

Claude Code

Documentación Completa

By Anthropic

Fecha de producción : 2026-03-17

Una herramienta de codificación agéntica en su terminal,
que entiende su código y le ayuda a programar más rápido.

Compilado de code.claude.com/docs/es
Para la versión más reciente, visite el sitio web.

Generated by: github.com/bepsvpt/cc-docs

Table of Contents

Part 1: Getting Started	5
Descripción general de Claude Code	
Inicio rápido	
Comenzar con la aplicación de escritorio	
Configurar Claude Code	25
Cómo funciona Claude Code	35
Part 2: Core Usage	45
Extender Claude Code	45
Modo interactivo	57
Mejores prácticas para Claude Code	
Flujos de trabajo comunes	94
Part 3: Commands & Reference	
Referencia de CLI	119
Personalizar atajos de teclado	
Part 4: Configuration	142
Configurar permisos	142
Cómo Claude recuerda su proyecto	
Configuración de Claude Code	165
Configuración del modelo	216
Estilos de salida	225
Acelera las respuestas con el modo rápido	
Gestionar costos de manera efectiva	
Part 5: Extensibility	241
Referencia de hooks	241
Automatizar flujos de trabajo con hooks	
Conectar Claude Code a herramientas mediante MCP	
Ampliar Claude con skills	369
Crear plugins	393
Referencia de plugins	406
Crear y distribuir un marketplace de plugins	
Descubra e instale plugins pregenerados a través de mercados	
Part 6: Advanced & Automation	471
Crear subagentes personalizados	

Orquestrar equipos de sesiones de Claude Code	
Ejecutar Claude Code mediante programación	
Continúa sesiones locales desde cualquier dispositivo con Control Remoto	
Ejecutar prompts en un horario	
Code Review	530

Part 7: CI/CD & Integrations

Claude Code GitHub Actions	536
Claude Code GitLab CI/CD	556

Part 8: IDE & Platform Integration 570

Usar Claude Code en VS Code	570
JetBrains IDEs	588
Usar Claude Code Desktop	593
Usar Claude Code con Chrome (beta)	
Claude Code en Slack	625
Claude Code en la web	632

Part 9: Security & Privacy

Seguridad	653
Sandboxing	659
checkpoint	670
Uso de datos	673
Legal y cumplimiento	679
Retención cero de datos	681

Part 10: Enterprise & Monitoring 685

Rastrear el uso del equipo con análisis	
Monitoreo	693
Descripción general de implementación empresarial	
Configurar la configuración administrada por servidor (beta pública)	

Part 11: Cloud Providers 723

Claude Code en Amazon Bedrock	
Claude Code en Google Vertex AI	
Claude Code en Microsoft Foundry	
Configuración de la puerta de enlace LLM	
Configuración de red empresarial	

Part 12: Environment Setup 748

Contenedores de desarrollo	748
Optimiza tu configuración de terminal	

Personaliza tu línea de estado

Part 13: Troubleshooting & Changelog

786

Solución de problemas

786

Part 1: Getting Started

Descripción general de Claude Code

Claude Code es una herramienta de codificación agencial que lee tu base de código, edita archivos, ejecuta comandos e integra con tus herramientas de desarrollo. Disponible en tu terminal, IDE, aplicación de escritorio y navegador.

Claude Code es un asistente de codificación impulsado por IA que te ayuda a construir características, corregir errores y automatizar tareas de desarrollo. Entiende tu base de código completa y puede trabajar en múltiples archivos y herramientas para lograr las cosas.

Comenzar

Elige tu entorno para comenzar. La mayoría de las superficies requieren una [suscripción a Claude](#) o una cuenta de [Anthropic Console](#). La CLI de Terminal y VS Code también admiten [proveedores de terceros](#).

Terminal

La CLI completa para trabajar con Claude Code directamente en tu terminal. Edita archivos, ejecuta comandos y gestiona tu proyecto completo desde la línea de comandos.

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Luego inicia Claude Code en cualquier proyecto:

```
cd your-project  
claude
```

Se te pedirá que inicies sesión en el primer uso. ¡Eso es todo! [Continúa con el Inicio rápido](#)
→

Tip:

Consulta [configuración avanzada](#) para opciones de instalación, actualizaciones manuales o instrucciones de desinstalación. Visita [solución de problemas](#) si encuentras problemas.

VS Code

La extensión de VS Code proporciona diffs en línea, menciones @, revisión de planes e historial de conversación directamente en tu editor.

- [Instalar para VS Code](#)
- [Instalar para Cursor](#)

O busca “Claude Code” en la vista de Extensiones (`Cmd+Shift+X` en Mac, `Ctrl+Shift+X` en Windows/Linux). Después de instalar, abre la Paleta de Comandos (`Cmd+Shift+P` / `Ctrl+Shift+P`), escribe “Claude Code” y selecciona **Abrir en Nueva Pestaña**.

[Comenzar con VS Code →](#)

Aplicación de escritorio

Una aplicación independiente para ejecutar Claude Code fuera de tu IDE o terminal. Revisa diffs visualmente, ejecuta múltiples sesiones lado a lado, programa tareas recurrentes e inicia sesiones en la nube.

Descarga e instala:

- [macOS](#) (Intel y Apple Silicon)
- [Windows](#) (x64)
- [Windows ARM64](#) (solo sesiones remotas)

Después de instalar, lanza Claude, inicia sesión y haz clic en la pestaña **Code** para comenzar a codificar. Se requiere una [suscripción de pago](#).

[Obtén más información sobre la aplicación de escritorio →](#)

Web

Ejecuta Claude Code en tu navegador sin configuración local. Inicia tareas de larga duración y vuelve cuando estén listas, trabaja en repositorios que no tienes localmente o ejecuta múltiples tareas en paralelo. Disponible en navegadores de escritorio y la aplicación Claude iOS.

Comienza a codificar en claude.ai/code.

[Comenzar en la web →](#)

JetBrains

Un plugin para IntelliJ IDEA, PyCharm, WebStorm y otros IDEs de JetBrains con visualización de diff interactiva y compartición de contexto de selección.

Instala el [plugin Claude Code](#) desde el Marketplace de JetBrains y reinicia tu IDE.

[Comenzar con JetBrains →](#)

Lo que puedes hacer

Aquí hay algunas de las formas en que puedes usar Claude Code:

Claude Code maneja las tareas tediosas que consumen tu día: escribir pruebas para código sin probar, corregir errores de lint en un proyecto, resolver conflictos de fusión, actualizar dependencias y escribir notas de lanzamiento.

```
claude "write tests for the auth module, run them, and fix any failures"
```

Describe lo que quieres en lenguaje natural. Claude Code planifica el enfoque, escribe el código en múltiples archivos y verifica que funcione.

Para errores, pega un mensaje de error o describe el síntoma. Claude Code rastrea el problema a través de tu base de código, identifica la causa raíz e implementa una solución. Consulta [flujos de trabajo comunes](#) para más ejemplos.

Claude Code funciona directamente con git. Prepara cambios, escribe mensajes de commit, crea ramas y abre solicitudes de extracción.

```
claude "commit my changes with a descriptive message"
```

En CI, puedes automatizar la revisión de código y la clasificación de problemas con [GitHub Actions](#) o [GitLab CI/CD](#).

El [Protocolo de Contexto de Modelo \(MCP\)](#) es un estándar abierto para conectar herramientas de IA a fuentes de datos externas. Con MCP, Claude Code puede leer tus documentos de diseño en Google Drive, actualizar tickets en Jira, extraer datos de Slack o usar tu propia herramienta personalizada.

[CLAUDE.md](#) es un archivo markdown que añades a la raíz de tu proyecto que Claude Code lee al inicio de cada sesión. Úsalo para establecer estándares de codificación, decisiones de arquitectura, librerías preferidas y listas de verificación de revisión. Claude también construye [memoria automática](#) mientras trabaja, guardando aprendizajes como comandos de compilación e insights de depuración en sesiones sin que escribas nada.

Crea [comandos personalizados](#) para empaquetar flujos de trabajo repetibles que tu equipo pueda compartir, como `/review-pr` o `/deploy-staging`.

[Hooks](#) te permiten ejecutar comandos de shell antes o después de acciones de Claude Code, como auto-formateo después de cada edición de archivo o ejecutar lint antes de un commit.

Genera [múltiples agentes de Claude Code](#) que trabajen en diferentes partes de una tarea simultáneamente. Un agente líder coordina el trabajo, asigna subtareas y fusiona resultados.

Para flujos de trabajo completamente personalizados, el [Agent SDK](#) te permite construir tus propios agentes impulsados por las herramientas y capacidades de Claude Code, con control total sobre orquestación, acceso a herramientas y permisos.

Claude Code es componible y sigue la filosofía de Unix. Canaliza registros en él, ejecútalo en CI o encadénalo con otras herramientas:

```
## Monitor logs and get alerted
tail -f app.log | claude -p "Slack me if you see any anomalies"

## Automate translations in CI
claude -p "translate new strings into French and raise a PR for review"

## Bulk operations across files
git diff main --name-only | claude -p "review these changed files for security issues"
```

Consulta la [referencia de CLI](#) para el conjunto completo de comandos y banderas.

Las sesiones no están vinculadas a una única superficie. Mueve el trabajo entre entornos a medida que cambia tu contexto:

- Aléjate de tu escritorio y sigue trabajando desde tu teléfono o cualquier navegador con [Control Remoto](#)
- Inicia una tarea de larga duración en la [web](#) o [aplicación iOS](#), luego extráela a tu terminal con `/teleport`
- Entrega una sesión de terminal a la [aplicación de Escritorio](#) con `/desktop` para revisión visual de diff
- Enruta tareas desde el chat del equipo: menciona `@Claude` en [Slack](#) con un informe de error y obtén una solicitud de extracción de vuelta

Usa Claude Code en todas partes

Cada superficie se conecta al mismo motor subyacente de Claude Code, por lo que tus archivos CLAUDE.md, configuración y servidores MCP funcionan en todos ellos.

Más allá de los entornos [Terminal](#), [VS Code](#), [JetBrains](#), [Desktop](#) y [Web](#) anteriores, Claude Code se integra con flujos de trabajo de CI/CD, chat y navegador:

Quiero...	Mejor opción
Continuar una sesión local desde mi teléfono u otro dispositivo	Control Remoto
Iniciar una tarea localmente, continuar en móvil	Web o aplicación Claude iOS
Automatizar revisiones de PR y clasificación de problemas	GitHub Actions o GitLab CI/CD
Obtener revisión de código automática en cada PR	Revisión de Código de GitHub
Enrutar informes de errores de Slack a solicitudes de extracción	Slack
Depurar aplicaciones web en vivo	Chrome
Construir agentes personalizados para tus propios flujos de trabajo	Agent SDK

Próximos pasos

Una vez que hayas instalado Claude Code, estas guías te ayudan a profundizar.

- [Inicio rápido](#): recorre tu primera tarea real, desde explorar una base de código hasta confirmar una solución
- [Almacena instrucciones y memorias](#): dale a Claude instrucciones persistentes con archivos CLAUDE.md y memoria automática
- [Flujos de trabajo comunes](#) y [mejores prácticas](#): patrones para obtener lo máximo de Claude Code
- [Configuración](#): personaliza Claude Code para tu flujo de trabajo
- [Solución de problemas](#): soluciones para problemas comunes

- code.claude.com: demostraciones, precios y detalles del producto

Inicio rápido

¡Bienvenido a Claude Code!

Esta guía de inicio rápido te permitirá usar asistencia de codificación impulsada por IA en pocos minutos. Al final, comprenderás cómo usar Claude Code para tareas comunes de desarrollo.

Antes de comenzar

Asegúrate de tener:

- Una terminal o símbolo del sistema abiertos
 - Si nunca has usado la terminal antes, consulta la [guía de terminal](#)
- Un proyecto de código con el que trabajar
- Una [suscripción a Claude](#) (Pro, Max, Teams o Enterprise), una cuenta de [Claude Console](#), o acceso a través de un [proveedor de nube compatible](#)

Note:

Esta guía cubre la CLI de terminal. Claude Code también está disponible en la [web](#), como una [aplicación de escritorio](#), en [VS Code](#) e [IDEs de JetBrains](#), en [Slack](#), y en CI/CD con [GitHub Actions](#) y [GitLab](#). Consulta [todas las interfaces](#).

Paso 1: Instalar Claude Code

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Paso 2: Inicia sesión en tu cuenta

Claude Code requiere una cuenta para usarse. Cuando inicies una sesión interactiva con el comando `claude`, deberás iniciar sesión:

```
claude
## Se te pedirá que inicies sesión en el primer uso
```

```
/login
## Sigue las indicaciones para iniciar sesión con tu cuenta
```

Puedes iniciar sesión usando cualquiera de estos tipos de cuenta:

- [Claude Pro, Max, Teams o Enterprise](#) (recomendado)
- [Claude Console](#) (acceso a API con créditos prepagados). En el primer inicio de sesión, se crea automáticamente un espacio de trabajo “Claude Code” en la Consola para el seguimiento centralizado de costos.
- [Amazon Bedrock, Google Vertex AI o Microsoft Foundry](#) (proveedores de nube empresariales)

Una vez que hayas iniciado sesión, tus credenciales se almacenan y no necesitarás iniciar sesión nuevamente. Para cambiar de cuenta más tarde, usa el comando `/login`.

Paso 3: Inicia tu primera sesión

Abre tu terminal en cualquier directorio de proyecto e inicia Claude Code:

```
cd /path/to/your/project
claude
```

Verás la pantalla de bienvenida de Claude Code con la información de tu sesión, conversaciones recientes y las últimas actualizaciones. Escribe `/help` para ver los comandos disponibles o `/resume` para continuar una conversación anterior.

Tip:

Después de iniciar sesión (Paso 2), tus credenciales se almacenan en tu sistema. Obtén más información en [Gestión de credenciales](#).

Paso 4: Haz tu primera pregunta

Comencemos por entender tu base de código. Intenta uno de estos comandos:

¿qué hace este proyecto?

Claude analizará tus archivos y proporcionará un resumen. También puedes hacer preguntas más específicas:

¿qué tecnologías usa este proyecto?

¿dónde está el punto de entrada principal?

explica la estructura de carpetas

También puedes preguntarle a Claude sobre sus propias capacidades:

¿qué puede hacer Claude Code?

¿cómo creo skills personalizados en Claude Code?

¿puede Claude Code trabajar con Docker?

Note:

Claude Code lee los archivos de tu proyecto según sea necesario. No tienes que agregar contexto manualmente.

Paso 5: Realiza tu primer cambio de código

Ahora hagamos que Claude Code haga algo de codificación real. Intenta una tarea simple:

agrega una función hello world al archivo principal

Claude Code hará lo siguiente:

1. Encontrará el archivo apropiado

2. Te mostrará los cambios propuestos
3. Pedirá tu aprobación
4. Realizará la edición

Note:

Claude Code siempre pide permiso antes de modificar archivos. Puedes aprobar cambios individuales o habilitar el modo “Aceptar todo” para una sesión.

Paso 6: Usa Git con Claude Code

Claude Code hace que las operaciones de Git sean conversacionales:

¿qué archivos he cambiado?

confirma mis cambios con un mensaje descriptivo

También puedes solicitar operaciones de Git más complejas:

crea una nueva rama llamada feature/quickstart

muéstrame los últimos 5 commits

ayúdame a resolver conflictos de fusión

Paso 7: Corrige un error o agrega una función

Claude es competente en depuración e implementación de funciones.

Describe lo que deseas en lenguaje natural:

agrega validación de entrada al formulario de registro de usuarios

O corrige problemas existentes:


```
hay un error donde los usuarios pueden enviar formularios vacíos - corrígelo
```

Claude Code hará lo siguiente:

- Localizará el código relevante
- Comprenderá el contexto
- Implementará una solución
- Ejecutará pruebas si están disponibles

Paso 8: Prueba otros flujos de trabajo comunes

Hay varias formas de trabajar con Claude:

Refactorizar código

```
refactoriza el módulo de autenticación para usar async/await en lugar de callbacks
```

Escribir pruebas

```
escribe pruebas unitarias para las funciones de calculadora
```

Actualizar documentación

```
actualiza el README con instrucciones de instalación
```

Revisión de código

```
revisa mis cambios y sugiere mejoras
```

Tip:

Habla con Claude como lo harías con un colega útil. Describe lo que deseas lograr y te ayudará a llegar allí.

Comandos esenciales

Aquí están los comandos más importantes para el uso diario:

Comando	Qué hace	Ejemplo
<code>claude</code>	Inicia el modo interactivo	<code>claude</code>
<code>claude "task"</code>	Ejecuta una tarea única	<code>claude "fix the build error"</code>
<code>claude -p "query"</code>	Ejecuta una consulta única y luego sale	<code>claude -p "explain this function"</code>
<code>claude -c</code>	Continúa la conversación más reciente en el directorio actual	<code>claude -c</code>
<code>claude -r</code>	Reanuda una conversación anterior	<code>claude -r</code>
<code>claude commit</code>	Crea un commit de Git	<code>claude commit</code>
<code>/clear</code>	Borra el historial de conversación	<code>/clear</code>
<code>/help</code>	Muestra los comandos disponibles	<code>/help</code>
<code>exit</code> o Ctrl+C	Salir de Claude Code	<code>exit</code>

Consulta la [referencia de CLI](#) para obtener una lista completa de comandos.

Consejos profesionales para principiantes

Para más información, consulta [mejores prácticas](#) y [flujos de trabajo comunes](#).

En lugar de: “corrige el error”

Intenta: “corrige el error de inicio de sesión donde los usuarios ven una pantalla en blanco después de ingresar credenciales incorrectas”

Divide tareas complejas en pasos:

1. crea una nueva tabla de base de datos para perfiles de usuario
2. crea un endpoint de API para obtener y actualizar perfiles de usuario
3. construye una página web que permita a los usuarios ver y editar su información

Antes de hacer cambios, deja que Claude entienda tu código:

analiza el esquema de la base de datos

construye un panel que muestre los productos que nuestros clientes del Reino Unido devuelven con mayor frecuencia

- Presiona `?` para ver todos los atajos de teclado disponibles
- Usa Tab para completar comandos
- Presiona `↑` para el historial de comandos
- Escribe `/` para ver todos los comandos y skills

¿Qué sigue?

Ahora que has aprendido lo básico, explora funciones más avanzadas:

- [Cómo funciona Claude Code](#) Comprende el bucle de agente, las herramientas integradas y cómo Claude Code interactúa con tu proyecto
- [Mejores prácticas](#) Obtén mejores resultados con indicaciones efectivas y configuración de proyecto
- [Flujos de trabajo comunes](#) Guías paso a paso para tareas comunes
- [Extiende Claude Code](#) Personaliza con CLAUDE.md, skills, hooks, MCP y más

Obtener ayuda

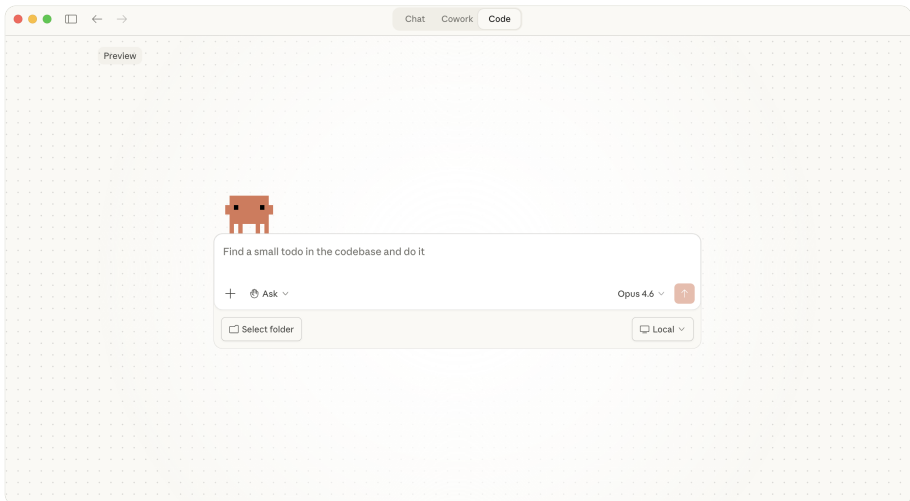
- **En Claude Code:** Escribe `/help` o pregunta “¿cómo...”
- **Documentación:** ¡Estás aquí! Explora otras guías
- **Comunidad:** Únete a nuestro [Discord](#) para consejos y soporte

Comenzar con la aplicación de escritorio

Instale Claude Code en el escritorio e inicie su primera sesión de codificación

La aplicación de escritorio te proporciona Claude Code con una interfaz gráfica: revisión visual de diferencias, vista previa de aplicaciones en vivo, monitoreo de PR de GitHub con fusión automática, sesiones paralelas con aislamiento de Git worktree, tareas programadas y la capacidad de ejecutar tareas de forma remota. No se requiere terminal.

Esta página te guía a través de la instalación de la aplicación e iniciando tu primera sesión. Si ya estás configurado, consulta [Usar Claude Code Desktop](#) para la referencia completa.



La interfaz de Claude Code Desktop mostrando la pestaña Code seleccionada, con un cuadro de solicitud, selector de modo de permisos establecido en Pedir permisos, selector de modelo, selector de carpeta y opción de entorno local

La aplicación de escritorio tiene tres pestañas:

- **Chat:** Conversación general sin acceso a archivos, similar a claude.ai.
- **Cowork:** Un agente autónomo de fondo que trabaja en tareas en una VM en la nube con su propio entorno. Puede funcionar de forma independiente mientras realizas otro trabajo.
- **Code:** Un asistente de codificación interactivo con acceso directo a tus archivos locales. Revisas y apruebas cada cambio en tiempo real.

Chat y Cowork se tratan en los [artículos de soporte de Claude Desktop](#). Esta página se enfoca en la pestaña **Code**.

Note:

Claude Code requiere una [suscripción Pro, Max, Teams o Enterprise](#).

Instalar

Step 1: Descargar la aplicación

Descarga Claude para tu plataforma.

- [macOS](#) Compilación universal para Intel y Apple Silicon
- [Windows](#) Para procesadores x64

Para Windows ARM64, [descarga aquí](#).

Linux no es compatible actualmente.

Step 2: Iniciar sesión

Abre Claude desde tu carpeta de Aplicaciones (macOS) o menú Inicio (Windows). Inicia sesión con tu cuenta de Anthropic.

Step 3: Abre la pestaña Code

Haz clic en la pestaña **Code** en el centro superior. Si hacer clic en Code te solicita actualizar, necesitas [suscribirte a un plan de pago](#) primero. Si te solicita iniciar sesión en línea, completa el inicio de sesión y reinicia la aplicación. Si ves un error 403, consulta [solución de problemas de autenticación](#).

La aplicación de escritorio incluye Claude Code. No necesitas instalar Node.js o la CLI por separado. Para usar **claude** desde la terminal, instala la CLI por separado. Consulta [Comenzar con la CLI](#).

Inicia tu primera sesión

Con la pestaña Code abierta, elige un proyecto y dale a Claude algo que hacer.

Step 1: Elige un entorno y carpeta

Selecciona **Local** para ejecutar Claude en tu máquina usando tus archivos directamente. Haz clic en **Seleccionar carpeta** y elige tu directorio de proyecto.

Tip:

Comienza con un proyecto pequeño que conozcas bien. Es la forma más rápida de ver qué puede hacer Claude Code. En Windows, [Git](#) debe estar instalado para que las sesiones locales funcionen. La mayoría de las Macs incluyen Git de forma predeterminada.

También puedes seleccionar:

- **Remote:** Ejecuta sesiones en la infraestructura en la nube de Anthropic que continúan incluso si cierras la aplicación. Las sesiones remotas utilizan la misma infraestructura que [Claude Code en la web](#).
- **SSH:** Conéctate a una máquina remota a través de SSH (tus propios servidores, VMs en la nube o contenedores de desarrollo). Claude Code debe estar instalado en la máquina remota.

Step 2: Elige un modelo

Selecciona un modelo del menú desplegable junto al botón de envío. Consulta [modelos](#) para una comparación de Opus, Sonnet y Haiku. No puedes cambiar el modelo después de que la sesión comience.

Step 3: Dile a Claude qué hacer

Escribe lo que quieres que Claude haga:

- Encuentra un comentario TODO y corrígelo
- Agrega pruebas para la función principal
- Crea un CLAUDE.md con instrucciones para este repositorio de código

Una [sesión](#) es una conversación con Claude sobre tu código. Cada sesión rastrea su propio contexto y cambios, para que puedas trabajar en múltiples tareas sin que se interfieran entre sí.

Step 4: Revisa y acepta cambios

De forma predeterminada, la pestaña Code comienza en [modo Pedir permisos](#), donde Claude propone cambios y espera tu aprobación antes de aplicarlos. Verás:

1. Una [vista de diferencias](#) que muestra exactamente qué cambiará en cada archivo
2. Botones Aceptar/Rechazar para aprobar o rechazar cada cambio
3. Actualizaciones en tiempo real mientras Claude trabaja en tu solicitud

Si rechazas un cambio, Claude te preguntará cómo te gustaría proceder de manera diferente. Tus archivos no se modifican hasta que aceptes.

¿Y ahora qué?

Has realizado tu primera edición. Para la referencia completa de todo lo que Desktop puede hacer, consulta [Usar Claude Code Desktop](#). Aquí hay algunas cosas para probar a continuación.

Interrompe y dirige. Puedes interrumpir a Claude en cualquier momento. Si va por el camino equivocado, haz clic en el botón de parada o escribe tu corrección y presiona **Enter**. Claude detiene lo que está haciendo y se ajusta según tu entrada. No tienes que esperar a que termine o empezar de nuevo.

Dale a Claude más contexto. Escribe `@nombre_de_archivo` en el cuadro de solicitud para incluir un archivo específico en la conversación, adjunta imágenes y PDFs usando el botón de adjuntos, o arrastra y suelta archivos directamente en la solicitud. Cuanto más contexto tenga Claude, mejores serán los resultados. Consulta [Agregar archivos y contexto](#).

Usa skills para tareas repetibles. Escribe `/` o haz clic en `+ →` **Slash commands** para explorar [comandos integrados](#), [skills personalizados](#) y skills de plugins. Los skills son solicitudes reutilizables que puedes invocar cuando las necesites, como listas de verificación de revisión de código o pasos de implementación.

Revisa cambios antes de confirmar. Después de que Claude edita archivos, aparece un indicador `+12 -1`. Haz clic en él para abrir la [vista de diferencias](#), revisa modificaciones archivo por archivo y comenta en líneas específicas. Claude lee tus comentarios y revisa. Haz clic en **Revisar código** para que Claude evalúe las diferencias y deje sugerencias en línea.

Ajusta cuánto control tienes. Tu [modo de permisos](#) controla el equilibrio. Pedir permisos (predeterminado) requiere aprobación antes de cada edición. Auto aceptar ediciones acepta automáticamente ediciones de archivos para una iteración más rápida. Plan mode permite que Claude trace un enfoque sin tocar ningún archivo, lo cual es útil antes de una refactorización grande.

Agrega plugins para más capacidades. Haz clic en el botón `+` junto al cuadro de solicitud y selecciona **Plugins** para explorar e instalar [plugins](#) que agregan skills, agentes, MCP servers y más.

Vista previa de tu aplicación. Haz clic en el menú desplegable **Preview** para ejecutar tu servidor de desarrollo directamente en el escritorio. Claude puede ver la aplicación en ejecución, probar puntos finales, inspeccionar registros e iterar en lo que ve. Consulta [Vista previa de tu aplicación](#).

Rastrea tu solicitud de extracción. Después de abrir un PR, Claude Code monitorea los resultados de las verificaciones de CI y puede corregir automáticamente fallos o fusionar el PR una vez que todas las verificaciones pasen. Consulta [Monitorear estado de solicitud de extracción](#).

Pon a Claude en un horario. Configura [tareas programadas](#) para ejecutar Claude automáticamente de forma recurrente: una revisión de código diaria cada mañana, una auditoría de dependencias semanal, o un resumen que extrae de tus herramientas conectadas.

Escala cuando estés listo. Abre [sesiones paralelas](#) desde la barra lateral para trabajar en múltiples tareas a la vez, cada una en su propio Git worktree. Envía [trabajo de larga duración a la nube](#) para que continúe incluso si cierras la aplicación, o [continúa una sesión en la web o en tu IDE](#) si una tarea toma más tiempo del esperado. [Conecta herramientas externas](#) como GitHub, Slack y Linear para unir tu flujo de trabajo.

¿Vienes de la CLI?

Desktop ejecuta el mismo motor que la CLI con una interfaz gráfica. Puedes ejecutar ambos simultáneamente en el mismo proyecto, y comparten configuración (archivos CLAUDE.md, MCP servers, hooks, skills y configuración). Para una comparación completa de características, equivalentes de banderas y lo que no está disponible en Desktop, consulta [Comparación de CLI](#).

Qué sigue

- [Usar Claude Code Desktop](#): modos de permisos, sesiones paralelas, vista de diferencias, conectores y configuración empresarial
- [Solución de problemas](#): soluciones a errores comunes y problemas de configuración
- [Mejores prácticas](#): consejos para escribir solicitudes efectivas y aprovechar al máximo Claude Code
- [Flujos de trabajo comunes](#): tutoriales para depuración, refactorización, pruebas y más

Configurar Claude Code

Instale, autentíquese e inicie el uso de Claude Code en su máquina de desarrollo.

Requisitos del sistema

- **Sistema operativo:**

- macOS 13.0+
- Windows 10 1809+ o Windows Server 2019+ ([consulte las notas de configuración](#))
- Ubuntu 20.04+
- Debian 10+
- Alpine Linux 3.19+ ([dependencias adicionales requeridas](#))

- **Hardware:** 4 GB+ de RAM

- **Red:** Se requiere conexión a Internet (consulte [configuración de red](#))

- **Shell:** Funciona mejor en Bash o Zsh

- **Ubicación:** [Países compatibles con Anthropic](#)

Dependencias adicionales

- **ripgrep:** Generalmente incluido con Claude Code. Si la búsqueda falla, consulte [solución de problemas de búsqueda](#).
- **Node.js 18+:** Solo se requiere para [instalación npm obsoleta](#)

Instalación

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Después de que se complete el proceso de instalación, navegue a su proyecto e inicie Claude Code:

```
cd your-awesome-project
claude
```

Si encuentra algún problema durante la instalación, consulte la [guía de solución de problemas](#).

Tip:

Ejecute `claude doctor` después de la instalación para verificar su tipo de instalación y versión.

Configuración específica de la plataforma

Windows: Ejecute Claude Code de forma nativa (requiere [Git Bash](#)) o dentro de WSL. Se admiten tanto WSL 1 como WSL 2, pero WSL 1 tiene soporte limitado y no admite características como el sandboxing de herramientas Bash.

Alpine Linux y otras distribuciones basadas en musl/uClibc:

El instalador nativo en Alpine y otras distribuciones basadas en musl/uClibc requiere `libgcc`, `libstdc++` y `ripgrep`. Instale estos usando el gestor de paquetes de su distribución, luego establezca `USE_BUILTIN_RIPGREP=0`.

En Alpine:

```
apk add libgcc libstdc++ ripgrep
```

Autenticación

Para individuos

1. **Plan Claude Pro o Max** (recomendado): Suscríbase al [plan Pro o Max](#) de Claude para obtener una suscripción unificada que incluya tanto Claude Code como Claude en la web. Administre su cuenta en un solo lugar e inicie sesión con su cuenta de Claude.ai.
2. **Claude Console:** Conéctese a través de [Claude Console](#) y complete el proceso de OAuth. Requiere facturación activa en la Consola de Anthropic. Se crea automáticamente un espacio de trabajo “Claude Code” para el seguimiento de uso y la gestión de costos. No puede crear claves de API para el espacio de trabajo de Claude Code; está dedicado exclusivamente al uso de Claude Code.

Para equipos y organizaciones

1. **Claude for Teams o Enterprise** (recomendado): Suscríbase a [Claude for Teams](#) o [Claude for Enterprise](#) para facturación centralizada, gestión de equipos y acceso tanto a Claude Code como a Claude en la web. Los miembros del equipo inician sesión con sus cuentas de Claude.ai.
2. **Claude Console con facturación de equipo**: Configure una organización compartida de [Claude Console](#) con facturación de equipo. Invite a los miembros del equipo y asigne roles para el seguimiento de uso.
3. **Proveedores de nube**: Configure Claude Code para usar [Amazon Bedrock](#), [Google Vertex AI](#) o [Microsoft Foundry](#) para implementaciones con su infraestructura de nube existente.

Instalar una versión específica

El instalador nativo acepta un número de versión específico o un canal de lanzamiento (`latest` o `stable`). El canal que elija en el momento de la instalación se convierte en su predeterminado para actualizaciones automáticas. Consulte [Configurar canal de lanzamiento](#) para obtener más información.

Para instalar la versión más reciente (predeterminada):

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

Para instalar la versión estable:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s stable
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) stable
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd stable &&  
del install.cmd
```

Para instalar un número de versión específico:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s 1.0.58
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) 1.0.58
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd 1.0.58 &&  
del install.cmd
```

Integridad binaria y firma de código

- Los checksums SHA256 para todas las plataformas se publican en los manifiestos de lanzamiento, actualmente ubicados en <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json> (ejemplo: reemplace {VERSION} con 2.0.30)
- Los binarios firmados se distribuyen para las siguientes plataformas:
 - macOS: Firmado por “Anthropic PBC” y notariado por Apple
 - Windows: Firmado por “Anthropic, PBC”

Instalación NPM (obsoleta)

La instalación de NPM está obsoleta. Use el método de [instalación nativa](#) cuando sea posible. Para migrar una instalación npm existente a nativa, ejecute `claude install`.

Instalación global de npm

```
npm install -g @anthropic-ai/claude-code
```

Warning:

NO use `sudo npm install -g` ya que esto puede causar problemas de permisos y riesgos de seguridad. Si encuentra errores de permisos, consulte [solución de problemas de permisos](#) para obtener soluciones recomendadas.

Configuración de Windows

Opción 1: Claude Code dentro de WSL

- Se admiten tanto WSL 1 como WSL 2
- WSL 2 admite [sandboxing](#) para mayor seguridad. WSL 1 no admite sandboxing.

Opción 2: Claude Code en Windows nativo con Git Bash

- Requiere [Git para Windows](#)
- Para instalaciones portátiles de Git, especifique la ruta a su `bash.exe`:

```
$env:CLAUDE_CODE_GIT_BASH_PATH="C:\Program Files\Git\bin\bash.exe"
```

Actualizar Claude Code

Actualizaciones automáticas

Claude Code se mantiene automáticamente actualizado para asegurar que tenga las características más recientes y correcciones de seguridad.

- **Comprobaciones de actualización:** Se realizan al inicio y periódicamente mientras se ejecuta
- **Proceso de actualización:** Descarga e instala automáticamente en segundo plano
- **Notificaciones:** Verá una notificación cuando se instalen las actualizaciones

- **Aplicar actualizaciones:** Las actualizaciones surten efecto la próxima vez que inicie Claude Code

Note:

Las instalaciones de Homebrew y WinGet no se actualizan automáticamente. Use `brew upgrade claude-code` o `winget upgrade Anthropic.ClaudeCode` para actualizar manualmente.

Problema conocido: Claude Code puede notificarle sobre actualizaciones antes de que la nueva versión esté disponible en estos gestores de paquetes. Si una actualización falla, espere e intente más tarde.

Configurar canal de lanzamiento

Configure qué canal de lanzamiento sigue Claude Code tanto para actualizaciones automáticas como para `claude update` con la configuración `autoUpdatesChannel`:

- `"latest"` (predeterminado): Reciba nuevas características tan pronto como se lancen
- `"stable"`: Use una versión que típicamente tiene aproximadamente una semana de antigüedad, omitiendo lanzamientos con regresiones importantes

Configure esto a través de `/config` → **Auto-update channel**, o agréguelo a su [archivo settings.json](#):

```
{
  "autoUpdatesChannel": "stable"
}
```

Para implementaciones empresariales, puede aplicar un canal de lanzamiento consistente en toda su organización usando [configuraciones administradas](#).

Deshabilitar actualizaciones automáticas

Establezca la variable de entorno `DISABLE_AUTOUPDATER` en su shell o [archivo settings.json](#):

```
export DISABLE_AUTOUPDATER=1
```

Actualizar manualmente

```
claude update
```

Desinstalar Claude Code

Si necesita desinstalar Claude Code, siga las instrucciones para su método de instalación.

Instalación nativa

Elimine el binario de Claude Code y los archivos de versión:

macOS, Linux, WSL:

```
rm -f ~/.local/bin/claude  
rm -rf ~/.local/share/claude
```

Windows PowerShell:

```
Remove-Item -Path "$env:USERPROFILE\.local\bin\claude.exe" -Force  
Remove-Item -Path "$env:USERPROFILE\.local\share\claude" -Recurse -Force
```

Windows CMD:

```
del "%USERPROFILE%\local\bin\claude.exe"  
rmdir /s /q "%USERPROFILE%\local\share\claude"
```

Instalación de Homebrew

```
brew uninstall --cask claude-code
```

Instalación de WinGet

```
winget uninstall Anthropic.ClaudeCode
```


Instalación de NPM

```
npm uninstall -g @anthropic-ai/claude-code
```

Limpiar archivos de configuración (opcional)

Warning:

Eliminar archivos de configuración eliminará toda su configuración, herramientas permitidas, configuraciones del servidor MCP e historial de sesiones.

Para eliminar la configuración y datos en caché de Claude Code:

macOS, Linux, WSL:

```
## Eliminar configuración de usuario y estado
rm -rf ~/.claude
rm ~/.claude.json

## Eliminar configuración específica del proyecto (ejecute desde su directorio de
proyecto)
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell:

```
## Eliminar configuración de usuario y estado
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## Eliminar configuración específica del proyecto (ejecute desde su directorio de
proyecto)
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD:

REM Eliminar configuración de usuario y estado

```
rmdir /s /q "%USERPROFILE%\.claude"
```

```
del "%USERPROFILE%\.claude.json"
```

REM Eliminar configuración específica del proyecto (ejecute desde su directorio de proyecto)

```
rmdir /s /q ".claude"
```

```
del ".mcp.json"
```

Cómo funciona Claude Code

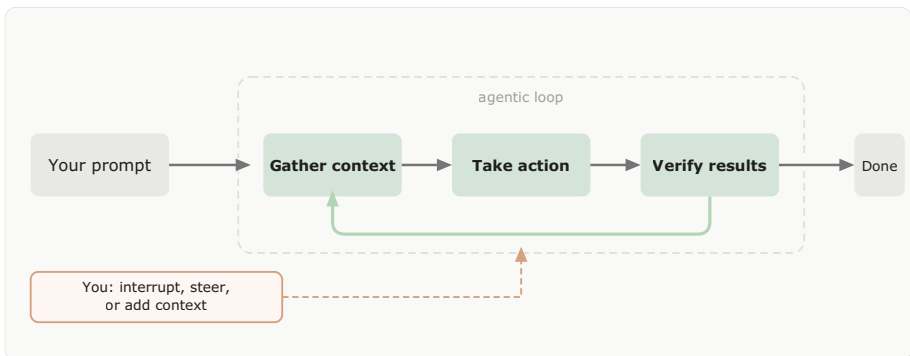
Comprenda el bucle agéntico, las herramientas integradas y cómo Claude Code interactúa con su proyecto.

Claude Code es un asistente agéntico que se ejecuta en su terminal. Aunque destaca en codificación, puede ayudarle con cualquier cosa que pueda hacer desde la línea de comandos: escribir documentación, ejecutar compilaciones, buscar archivos, investigar temas y más.

Esta guía cubre la arquitectura principal, las capacidades integradas y [consejos para trabajar efectivamente](#). Para tutoriales paso a paso, consulte [Flujos de trabajo comunes](#). Para características de extensibilidad como skills, MCP y hooks, consulte [Extender Claude Code](#).

El bucle agéntico

Cuando le da una tarea a Claude, trabaja a través de tres fases: **recopilar contexto**, **tomar acción** y **verificar resultados**. Estas fases se mezclan entre sí. Claude utiliza herramientas en todo momento, ya sea buscando archivos para entender su código, editando para hacer cambios o ejecutando pruebas para verificar su trabajo.



El bucle agéntico: Su indicación lleva a Claude a recopilar contexto, tomar acción, verificar resultados y repetir hasta completar la tarea. Puede interrumpir en cualquier momento.

El bucle se adapta a lo que pregunta. Una pregunta sobre su base de código podría necesitar solo recopilación de contexto. Una corrección de errores cicla a través de las tres fases repetidamente. Una refactorización podría implicar una verificación extensa. Claude decide qué requiere cada paso basándose en lo que aprendió del paso anterior, encadenando docenas de acciones juntas y corrigiendo el curso en el camino.

Usted también es parte de este bucle. Puede interrumpir en cualquier momento para dirigir a Claude en una dirección diferente, proporcionar contexto adicional o pedirle que intente un enfoque diferente. Claude trabaja de forma autónoma pero permanece receptivo a su entrada.

El bucle agéntico está impulsado por dos componentes: [modelos](#) que razonan y [herramientas](#) que actúan. Claude Code sirve como el **arnés agéntico** alrededor de Claude: proporciona las herramientas, la gestión del contexto y el entorno de ejecución que convierten un modelo de lenguaje en un agente de codificación capaz.

Modelos

Claude Code utiliza modelos Claude para entender su código y razonar sobre tareas. Claude puede leer código en cualquier idioma, entender cómo se conectan los componentes y determinar qué necesita cambiar para lograr su objetivo. Para tareas complejas, divide el trabajo en pasos, los ejecuta y se ajusta basándose en lo que aprende.

[Múltiples modelos](#) están disponibles con diferentes compensaciones. Sonnet maneja bien la mayoría de tareas de codificación. Opus proporciona un razonamiento más fuerte para decisiones arquitectónicas complejas. Cambie con `/model` durante una sesión o comience con `claude --model <name>`.

Cuando esta guía dice “Claude elige” o “Claude decide”, es el modelo el que está haciendo el razonamiento.

Herramientas

Las herramientas son lo que hace que Claude Code sea agéntico. Sin herramientas, Claude solo puede responder con texto. Con herramientas, Claude puede actuar: leer su código, editar archivos, ejecutar comandos, buscar en la web e interactuar con servicios externos. Cada uso de herramienta devuelve información que se retroalimenta en el bucle, informando la siguiente decisión de Claude.

Las herramientas integradas generalmente se dividen en cinco categorías, cada una representando un tipo diferente de agencia.

Categoría	Lo que Claude puede hacer
Operaciones de archivo	Leer archivos, editar código, crear nuevos archivos, renombrar y reorganizar
Búsqueda	Encontrar archivos por patrón, buscar contenido con regex, explorar bases de código
Ejecución	Ejecutar comandos de shell, iniciar servidores, ejecutar pruebas, usar git
Web	Buscar en la web, obtener documentación, buscar mensajes de error
Inteligencia de código	Ver errores de tipo y advertencias después de ediciones, saltar a definiciones, encontrar referencias (requiere plugins de inteligencia de código)

Estas son las capacidades principales. Claude también tiene herramientas para generar subagents, hacerle preguntas y otras tareas de orquestación. Consulte [Herramientas disponibles para Claude](#) para la lista completa.

Claude elige qué herramientas usar basándose en su indicación y lo que aprende en el camino. Cuando dice “arreglar las pruebas fallidas”, Claude podría:

1. Ejecutar el conjunto de pruebas para ver qué está fallando
2. Leer la salida de error
3. Buscar los archivos de código fuente relevantes
4. Leer esos archivos para entender el código
5. Editar los archivos para arreglar el problema
6. Ejecutar las pruebas nuevamente para verificar

Cada uso de herramienta le da a Claude nueva información que informa el siguiente paso. Este es el bucle agentico en acción.

Extender las capacidades base: Las herramientas integradas son la base. Puede extender lo que Claude sabe con [skills](#), conectarse a servicios externos con [MCP](#), automatizar flujos de trabajo con [hooks](#) y delegar tareas a [subagents](#). Estas extensiones forman una capa encima del bucle agentico principal. Consulte [Extender Claude Code](#) para orientación sobre cómo elegir la extensión correcta para sus necesidades.

A qué puede acceder Claude

Esta guía se enfoca en la terminal. Claude Code también se ejecuta en [VS Code](#), [IDEs de JetBrains](#) y otros entornos.

Cuando ejecuta `claude` en un directorio, Claude Code obtiene acceso a:

- **Su proyecto.** Archivos en su directorio y subdirectorios, más archivos en otros lugares con su permiso.
- **Su terminal.** Cualquier comando que pueda ejecutar: herramientas de compilación, git, gestores de paquetes, utilidades del sistema, scripts. Si puede hacerlo desde la línea de comandos, Claude también puede.
- **Su estado de git.** Rama actual, cambios sin confirmar e historial de confirmaciones recientes.
- **Su `CLAUDE.md`.** Un archivo markdown donde almacena instrucciones específicas del proyecto, convenciones y contexto que Claude debe conocer en cada sesión.
- **Memoria automática.** Aprendizajes que Claude guarda automáticamente mientras trabaja, como patrones de proyecto y sus preferencias. Las primeras 200 líneas de `MEMORY.md` se cargan al inicio de cada sesión.
- **Extensiones que configure.** [Servidores MCP](#) para servicios externos, [skills](#) para flujos de trabajo, [subagents](#) para trabajo delegado y [Claude en Chrome](#) para interacción del navegador.

Debido a que Claude ve todo su proyecto, puede trabajar en él. Cuando le pide a Claude que “arregle el error de autenticación”, busca archivos relevantes, lee múltiples archivos para entender el contexto, realiza ediciones coordinadas en ellos, ejecuta pruebas para verificar la corrección y confirma los cambios si lo solicita. Esto es diferente de los asistentes de código en línea que solo ven el archivo actual.

Entornos e interfaces

El bucle agentico, las herramientas y las capacidades descritas anteriormente son iguales en todas partes donde use Claude Code. Lo que cambia es dónde se ejecuta el código y cómo interactúa con él.

Entornos de ejecución

Claude Code se ejecuta en tres entornos, cada uno con diferentes compensaciones para dónde se ejecuta su código.

Entorno	Dónde se ejecuta el código	Caso de uso
Local	Su máquina	Predeterminado. Acceso completo a sus archivos, herramientas y entorno
Nube	VMs administradas por Anthropic	Delegar tareas, trabajar en repositorios que no tiene localmente
Control remoto	Su máquina, controlada desde un navegador	Usar la interfaz web mientras mantiene todo local

Interfaces

Puede acceder a Claude Code a través de la terminal, la [aplicación de escritorio](#), [extensiones de IDE](#), [claude.ai/code](#), [Control remoto](#), [Slack](#) y [canalizaciones CI/CD](#). La interfaz determina cómo ve e interactúa con Claude, pero el bucle agentico subyacente es idéntico. Consulte [Usar Claude Code en todas partes](#) para la lista completa.

Trabajar con sesiones

Claude Code guarda su conversación localmente mientras trabaja. Cada mensaje, uso de herramienta y resultado se almacena, lo que permite [rebobinar](#), [reanudar y bifurcar](#) sesiones. Antes de que Claude realice cambios de código, también toma una instantánea de los archivos afectados para que pueda revertir si es necesario.

Las sesiones son independientes. Cada nueva sesión comienza con una ventana de contexto nueva, sin el historial de conversación de sesiones anteriores. Claude puede persistir aprendizajes entre sesiones usando [memoria automática](#), y puede agregar sus propias instrucciones persistentes en [CLAUDE.md](#).

Trabajar entre ramas

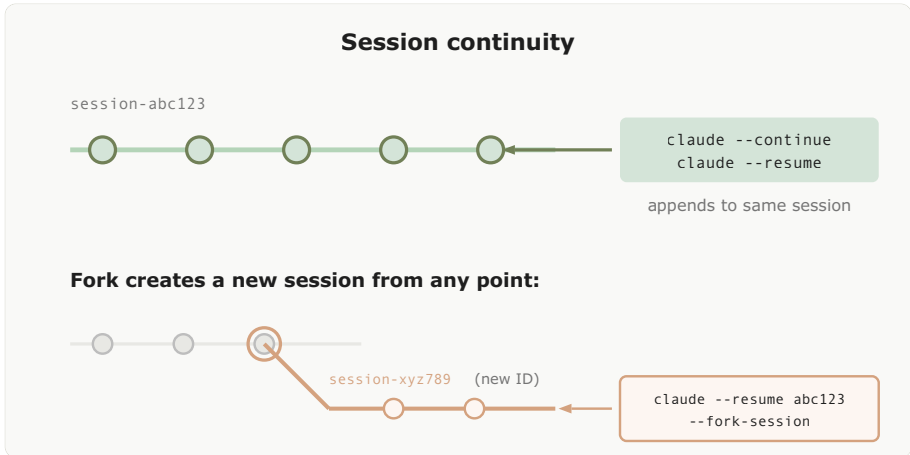
Cada conversación de Claude Code es una sesión vinculada a su directorio actual. Cuando reanuda, solo ve sesiones de ese directorio.

Claude ve los archivos de su rama actual. Cuando cambia de rama, Claude ve los archivos de la nueva rama, pero el historial de conversación permanece igual. Claude recuerda lo que discutió incluso después de cambiar de rama.

Dado que las sesiones están vinculadas a directorios, puede ejecutar sesiones paralelas de Claude Code usando [git worktrees](#), que crean directorios separados para ramas individuales.

Reanudar o bifurcar sesiones

Cuando reanuda una sesión con `claude --continue` o `claude --resume`, continúa donde lo dejó usando el mismo ID de sesión. Los nuevos mensajes se agregan a la conversación existente. Su historial de conversación completo se restaura, pero los permisos con alcance de sesión no. Deberá volver a aprobarlos.



Continuidad de sesión: reanudar continúa la misma sesión, bifurcar crea una nueva rama con un nuevo ID.

Para ramificar e intentar un enfoque diferente sin afectar la sesión original, use la bandera `--fork-session`:

```
claude --continue --fork-session
```

Esto crea un nuevo ID de sesión mientras preserva el historial de conversación hasta ese punto. La sesión original permanece sin cambios. Como reanudar, las sesiones bifurcadas no heredan permisos con alcance de sesión.

Misma sesión en múltiples terminales: Si reanuda la misma sesión en múltiples terminales, ambos terminales escriben en el mismo archivo de sesión. Los mensajes de ambos se intercalan, como dos personas escribiendo en el mismo cuaderno. Nada se corrompe, pero la conversación se vuelve confusa. Cada terminal solo ve sus propios mensajes durante la sesión, pero si reanuda esa sesión más tarde, verá todo intercalado. Para trabajo paralelo desde el mismo punto de partida, use `--fork-session` para dar a cada terminal su propia sesión limpia.

La ventana de contexto

La ventana de contexto de Claude contiene el historial de conversación, contenidos de archivos, salidas de comandos, [CLAUDE.md](#), skills cargados e instrucciones del sistema. A medida que trabaja, el contexto se llena. Claude se compacta automáticamente, pero las instrucciones del principio de la conversación pueden perderse. Coloque reglas persistentes en CLAUDE.md y ejecute `/context` para ver qué está usando espacio.

Cuando el contexto se llena

Claude Code gestiona el contexto automáticamente a medida que se acerca al límite. Primero borra salidas de herramientas más antiguas, luego resume la conversación si es necesario. Sus solicitudes y fragmentos de código clave se preservan; las instrucciones detalladas del principio de la conversación pueden perderse. Coloque reglas persistentes en CLAUDE.md en lugar de depender del historial de conversación.

Para controlar qué se preserva durante la compactación, agregue una sección “Compact Instructions” a CLAUDE.md o ejecute `/compact` con un enfoque (como `/compact focus on the API changes`).

Ejecute `/context` para ver qué está usando espacio. Los servidores MCP agregan definiciones de herramientas a cada solicitud, por lo que algunos servidores pueden consumir contexto significativo antes de que comience a trabajar. Ejecute `/mcp` para verificar costos por servidor.

Gestionar contexto con skills y subagents

Más allá de la compactación, puede usar otras características para controlar qué se carga en el contexto.

[Skills](#) se cargan bajo demanda. Claude ve descripciones de skills al inicio de la sesión, pero el contenido completo solo se carga cuando se usa un skill. Para skills que invoca manualmente, establezca `disable-model-invocation: true` para mantener descripciones fuera del contexto hasta que las necesite.

[Subagents](#) obtienen su propio contexto nuevo, completamente separado de su conversación principal. Su trabajo no infla su contexto. Cuando terminan, devuelven un resumen. Este aislamiento es por qué los subagents ayudan con sesiones largas.

Consulte [costos de contexto](#) para lo que cuesta cada característica y [reducir uso de tokens](#) para consejos sobre gestión de contexto.

Manténgase seguro con checkpoints y permisos

Claude tiene dos mecanismos de seguridad: los checkpoints le permiten deshacer cambios de archivo y los permisos controlan qué puede hacer Claude sin preguntar.

Deshacer cambios con checkpoints

Cada edición de archivo es reversible. Antes de que Claude edite cualquier archivo, toma una instantánea del contenido actual. Si algo sale mal, presione **Esc** dos veces para rebobinar a un estado anterior, o pida a Claude que deshaga.

Los checkpoints son locales a su sesión, separados de git. Solo cubren cambios de archivo. Las acciones que afectan sistemas remotos (bases de datos, APIs, implementaciones) no pueden ser checkpointed, por lo que Claude pregunta antes de ejecutar comandos con efectos secundarios externos.

Controle qué puede hacer Claude

Presione **Shift+Tab** para ciclar a través de modos de permiso:

- **Predeterminado:** Claude pregunta antes de ediciones de archivo y comandos de shell
- **Auto-aceptar ediciones:** Claude edita archivos sin preguntar, aún pregunta por comandos
- **Plan Mode:** Claude usa solo herramientas de solo lectura, creando un plan que puede aprobar antes de la ejecución

También puede permitir comandos específicos en `.claude/settings.json` para que Claude no pregunte cada vez. Esto es útil para comandos confiables como `npm test` o `git status`. Los ajustes pueden tener alcance desde políticas de toda la organización hasta preferencias personales. Consulte [Permisos](#) para detalles.

Trabajar efectivamente con Claude Code

Estos consejos le ayudan a obtener mejores resultados de Claude Code.

Pida ayuda a Claude Code

Claude Code puede enseñarle cómo usarlo. Haga preguntas como “¿cómo configuro hooks?” o “¿cuál es la mejor manera de estructurar mi CLAUDE.md?” y Claude explicará.

Los comandos integrados también lo guían a través de la configuración:

- `/init` lo guía a través de la creación de un CLAUDE.md para su proyecto

- `/agents` lo ayuda a configurar subagents personalizados
- `/doctor` diagnostica problemas comunes con su instalación

Es una conversación

Claude Code es conversacional. No necesita indicaciones perfectas. Comience con lo que desea, luego refine:

```
Arreglar el error de inicio de sesión
```

[Claude investiga, intenta algo]

```
Eso no es del todo correcto. El problema está en el manejo de sesiones.
```

[Claude ajusta el enfoque]

Cuando el primer intento no es correcto, no comienza de nuevo. Itera.

Interrumpir y dirigir

Puede interrumpir a Claude en cualquier momento. Si va por el camino equivocado, simplemente escriba su corrección y presione Enter. Claude dejará de hacer lo que está haciendo y ajustará su enfoque basándose en su entrada. No tiene que esperar a que termine o comenzar de nuevo.

Sea específico desde el principio

Cuanto más precisa sea su indicación inicial, menos correcciones necesitará. Haga referencia a archivos específicos, mencione restricciones y señale patrones de ejemplo.

```
El flujo de pago está roto para usuarios con tarjetas vencidas.  
Verifique src/payments/ para el problema, especialmente la actualización de  
tokens.  
Escriba una prueba fallida primero, luego arrégla.
```

Las indicaciones vagas funcionan, pero pasará más tiempo dirigiendo. Las indicaciones específicas como la anterior a menudo tienen éxito en el primer intento.

Dé a Claude algo contra lo que verificar

Claude funciona mejor cuando puede verificar su propio trabajo. Incluya casos de prueba, pegue capturas de pantalla de la interfaz de usuario esperada o defina la salida que desea.

```
Implementar validateEmail. Casos de prueba: 'user@example.com' → true,  
'invalid' → false, 'user@.com' → false. Ejecute las pruebas después.
```

Para trabajo visual, pegue una captura de pantalla del diseño y pida a Claude que compare su implementación con ella.

Explorar antes de implementar

Para problemas complejos, separe la investigación de la codificación. Use `plan mode` (**Shift+Tab** dos veces) para analizar la base de código primero:

```
Leer src/auth/ y entender cómo manejamos sesiones.  
Luego crear un plan para agregar soporte OAuth.
```

Revise el plan, refínelo a través de la conversación, luego deje que Claude implemente. Este enfoque de dos fases produce mejores resultados que saltar directamente al código.

Delegue, no dicte

Piense en delegar a un colega capaz. Dé contexto y dirección, luego confíe en que Claude descubra los detalles:

```
El flujo de pago está roto para usuarios con tarjetas vencidas.  
El código relevante está en src/payments/. ¿Puede investigar y arreglarlo?
```

No necesita especificar qué archivos leer o qué comandos ejecutar. Claude lo descubre.

Qué sigue

- [Extender con características](#) Agregar Skills, conexiones MCP y comandos personalizados
- [Flujos de trabajo comunes](#) Guías paso a paso para tareas típicas

Part 2: Core Usage

Extender Claude Code

Comprenda cuándo usar [CLAUDE.md](#), [Skills](#), [subagents](#), [hooks](#), [MCP](#) y [plugins](#).

Claude Code combina un modelo que razona sobre su código con [herramientas integradas](#) para operaciones de archivos, búsqueda, ejecución y acceso web. Las herramientas integradas cubren la mayoría de las tareas de codificación. Esta guía cubre la capa de extensión: características que agrega para personalizar lo que Claude sabe, conectarlo a servicios externos y automatizar flujos de trabajo.

Note:

Para saber cómo funciona el bucle agéntico central, consulte [Cómo funciona Claude Code](#).

¿Nuevo en Claude Code? Comience con [CLAUDE.md](#) para convenciones de proyecto. Agregue otras extensiones según sea necesario.

Descripción general

Las extensiones se conectan a diferentes partes del bucle agéntico:

- [CLAUDE.md](#) agrega contexto persistente que Claude ve en cada sesión
- [Skills](#) agregan conocimiento reutilizable y flujos de trabajo invocables
- [MCP](#) conecta Claude a servicios y herramientas externas
- [Subagents](#) ejecutan sus propios bucles en contexto aislado, devolviendo resúmenes
- [Agent teams](#) coordinan múltiples sesiones independientes con tareas compartidas y mensajería punto a punto
- [Hooks](#) se ejecutan fuera del bucle completamente como scripts deterministas
- [Plugins](#) y [marketplaces](#) empaquetan y distribuyen estas características

[Skills](#) son la extensión más flexible. Una skill es un archivo markdown que contiene conocimiento, flujos de trabajo o instrucciones. Puede invocar skills con un comando como `/deploy`, o Claude puede cargarlas automáticamente cuando sea relevante. Las skills pueden ejecutarse en su conversación actual o en un contexto aislado a través de subagents.

Hacer coincidir características con su objetivo

Las características van desde contexto siempre activo que Claude ve en cada sesión, hasta capacidades bajo demanda que usted o Claude pueden invocar, hasta automatización en segundo plano que se ejecuta en eventos específicos. La tabla a continuación muestra qué está disponible y cuándo tiene sentido cada uno.

Caracte- rística	Qué hace	Cuándo usarlo	Ejemplo
CLAU- DE.md	Contexto persistente cargado en cada conversación	Convenciones de proyecto, reglas “siempre haz X”	“Usa pnpm, no npm. Ejecuta pruebas antes de hacer commit.”
Skill	Instrucciones, conocimiento y flujos de trabajo que Claude puede usar	Contenido reutilizable, documentos de referencia, tareas repetibles	<code>/deploy</code> ejecuta su lista de verificación de implementación; skill de documentos API con patrones de endpoint
Subagent	Contexto de ejecución aislado que devuelve resultados resumidos	Aislamiento de contexto, tareas paralelas, trabajadores especializados	Tarea de investigación que lee muchos archivos pero devuelve solo hallazgos clave
Agent teams	Coordinar múltiples sesiones independientes de Claude Code	Investigación paralela, desarrollo de nuevas características, depuración con hipótesis competidoras	Generar revisores para verificar seguridad, rendimiento y pruebas simultáneamente
MCP	Conectar a servicios externos	Datos o acciones externas	Consultar su base de datos, publicar en Slack, controlar un navegador
Hook	Script determinista que se ejecuta en eventos	Automatización predecible, sin LLM involucrado	Ejecutar ESLint después de cada edición de archivo

Plugins son la capa de empaquetamiento. Un plugin agrupa skills, hooks, subagents y servidores MCP en una única unidad instalable. Las skills de plugin tienen espacios de nombres (como `/my-plugin:review`) para que múltiples plugins puedan coexistir. Use plugins cuando desee reutilizar la misma configuración en múltiples repositorios o distribuir a otros a través de un [marketplace](#).

Comparar características similares

Algunas características pueden parecer similares. Aquí se explica cómo distinguirlas.

Skill vs Subagent

Las skills y los subagents resuelven problemas diferentes:

- **Skills** son contenido reutilizable que puede cargar en cualquier contexto
- **Subagents** son trabajadores aislados que se ejecutan separadamente de su conversación principal

Aspecto	Skill	Subagent
Qué es	Instrucciones, conocimiento o flujos de trabajo reutilizables	Trabajador aislado con su propio contexto
Beneficio clave	Compartir contenido entre contextos	Aislamiento de contexto. El trabajo ocurre por separado, solo devuelve resumen
Mejor para	Material de referencia, flujos de trabajo invocables	Tareas que leen muchos archivos, trabajo paralelo, trabajadores especializados

Las skills pueden ser de referencia o acción. Las skills de referencia proporcionan conocimiento que Claude usa en toda su sesión (como su guía de estilo de API). Las skills de acción le dicen a Claude que haga algo específico (como `/deploy` que ejecuta su flujo de trabajo de implementación).

Use un subagent cuando necesite aislamiento de contexto o cuando su ventana de contexto se esté llenando. El subagent podría leer docenas de archivos o ejecutar búsquedas extensas, pero su conversación principal solo recibe un resumen. Dado que el trabajo del subagent no consume su contexto principal, esto también es útil cuando no necesita que el trabajo intermedio permanezca visible. Los subagents personalizados pueden tener sus propias instrucciones y pueden precargar skills.

Pueden combinarse. Un subagent puede precargar skills específicas (campo `skills:`). Una skill puede ejecutarse en contexto aislado usando `context: fork` . Consulte [Skills](#) para obtener detalles.

CLAUDE.md vs Skill

Ambos almacenan instrucciones, pero se cargan de manera diferente y sirven propósitos diferentes.

Aspecto	CLAUDE.md	Skill
Se carga	Cada sesión, automáticamente	Bajo demanda
Puede incluir archivos	Sí, con importaciones <code>@path</code>	Sí, con importaciones <code>@path</code>
Puede desencadenar flujos de trabajo	No	Sí, con <code><name></code>
Mejor para	Reglas “siempre haz X”	Material de referencia, flujos de trabajo invocables

Póngalo en CLAUDE.md si Claude siempre debe saberlo: convenciones de codificación, comandos de compilación, estructura del proyecto, reglas “nunca hagas X”.

Póngalo en una skill si es material de referencia que Claude necesita a veces (documentos de API, guías de estilo) o un flujo de trabajo que desencadena con `<name>` (implementar, revisar, lanzar).

Regla general: Mantenga CLAUDE.md bajo 200 líneas. Si está creciendo, mueva contenido de referencia a skills o divida en archivos `.claude/rules/` .

CLAUDE.md vs Rules vs Skills

Los tres almacenan instrucciones, pero se cargan de manera diferente:

Aspecto	CLAUDE.md	<code>.claude/rules/</code>	Skill
Se carga	Cada sesión	Cada sesión, o cuando se abren archivos coincidentes	Bajo demanda, cuando se invoca o es relevante
Alcance	Proyecto completo	Puede estar limitado a rutas de archivo	Específico de tarea

Aspecto	CLAUDE.md	.claude/rules/	Skill
Mejor para	Convenciones y comandos de compilación centrales	Directrices específicas del idioma o directorio	Material de referencia, flujos de trabajo repetibles

Use **CLAUDE.md** para instrucciones que cada sesión necesita: comandos de compilación, convenciones de prueba, arquitectura del proyecto.

Use **rules** para mantener CLAUDE.md enfocado. Las rules con `frontmatter paths` solo se cargan cuando Claude trabaja con archivos coincidentes, ahorrando contexto.

Use **skills** para contenido que Claude solo necesita a veces, como documentación de API o una lista de verificación de implementación que desencadena con `<name>`.

Subagent vs Agent team

Ambos paralelizan el trabajo, pero son arquitectónicamente diferentes:

- **Subagents** se ejecutan dentro de su sesión e informan resultados de vuelta a su contexto principal
- **Agent teams** son sesiones independientes de Claude Code que se comunican entre sí

Aspecto	Subagent	Agent team
Contexto	Ventana de contexto propia; los resultados regresan al llamador	Ventana de contexto propia; completamente independiente
Comunicación	Informa resultados solo al agente principal	Los compañeros se envían mensajes directamente entre sí
Coordinación	El agente principal gestiona todo el trabajo	Lista de tareas compartida con auto-coordinación
Mejor para	Tareas enfocadas donde solo importa el resultado	Trabajo complejo que requiere discusión y colaboración
Costo de token	Menor: resultados resumidos de vuelta al contexto principal	Mayor: cada compañero es una instancia separada de Claude

Use un subagent cuando necesite un trabajador rápido y enfocado: investigar una pregunta, verificar una afirmación, revisar un archivo. El subagent hace el trabajo y devuelve un resumen. Su conversación principal se mantiene limpia.

Use un agent team cuando los compañeros necesiten compartir hallazgos, desafiarse mutuamente y coordinarse de forma independiente. Los agent teams son mejores para investigación con hipótesis competidoras, revisión de código paralela y desarrollo de nuevas características donde cada compañero posee una pieza separada.

Punto de transición: Si está ejecutando subagents paralelos pero alcanzando límites de contexto, o si sus subagents necesitan comunicarse entre sí, los agent teams son el siguiente paso natural.

Note:

Los agent teams son experimentales y están deshabilitados por defecto. Consulte [agent teams](#) para configuración y limitaciones actuales.

MCP vs Skill

MCP conecta Claude a servicios externos. Las skills extienden lo que Claude sabe, incluyendo cómo usar esos servicios de manera efectiva.

Aspecto	MCP	Skill
Qué es	Protocolo para conectar a servicios externos	Conocimiento, flujos de trabajo y material de referencia
Proporciona	Acceso a herramientas y datos	Conocimiento, flujos de trabajo, material de referencia
Ejemplos	Integración de Slack, consultas de base de datos, control de navegador	Lista de verificación de revisión de código, flujo de trabajo de implementación, guía de estilo de API

Estos resuelven problemas diferentes y funcionan bien juntos:

MCP le da a Claude la capacidad de interactuar con sistemas externos. Sin MCP, Claude no puede consultar su base de datos o publicar en Slack.

Skills le dan a Claude conocimiento sobre cómo usar esas herramientas de manera efectiva, además de flujos de trabajo que puede desencadenar con `/<name>`. Una skill podría incluir el esquema de base de datos de su equipo y patrones de consulta, o un flujo de trabajo `/post-to-slack` con las reglas de formato de mensaje de su equipo.

Ejemplo: Un servidor MCP conecta Claude a su base de datos. Una skill enseña a Claude su modelo de datos, patrones de consulta comunes y qué tablas usar para diferentes tareas.

Entender cómo se superponen las características

Las características se pueden definir en múltiples niveles: en todo el usuario, por proyecto, a través de plugins o mediante políticas administradas. También puede anidar archivos CLAUDE.md en subdirectorios o colocar skills en paquetes específicos de un monorepo. Cuando la misma característica existe en múltiples niveles, así es como se superponen:

- **Los archivos CLAUDE.md** son aditivos: todos los niveles contribuyen contenido al contexto de Claude simultáneamente. Los archivos de su directorio de trabajo y superior se cargan al iniciar; los subdirectorios se cargan mientras trabaja en ellos. Cuando las instrucciones entran en conflicto, Claude usa el juicio para reconciliarlas, con instrucciones más específicas típicamente teniendo precedencia. Consulte [cómo se cargan los archivos CLAUDE.md](#).
- **Las skills y subagents** se anulan por nombre: cuando el mismo nombre existe en múltiples niveles, una definición gana según la prioridad (administrado > usuario > proyecto para skills; administrado > bandera CLI > proyecto > usuario > plugin para subagents). Las skills de plugin tienen [espacios de nombres](#) para evitar conflictos. Consulte [descubrimiento de skills](#) y [alcance de subagent](#).
- **Los servidores MCP** se anulan por nombre: local > proyecto > usuario. Consulte [alcance de MCP](#).
- **Los hooks** se fusionan: todos los hooks registrados se disparan para sus eventos coincidentes independientemente de la fuente. Consulte [hooks](#).

Combinar características

Cada extensión resuelve un problema diferente: CLAUDE.md maneja contexto siempre activo, las skills manejan conocimiento bajo demanda y flujos de trabajo, MCP maneja conexiones externas, los subagents manejan aislamiento y los hooks manejan automatización. Las configuraciones reales las combinan según su flujo de trabajo.

Por ejemplo, podría usar CLAUDE.md para convenciones de proyecto, una skill para su flujo de trabajo de implementación, MCP para conectar a su base de datos y un hook para ejecutar linting después de cada edición. Cada característica maneja lo que hace mejor.

Patrón	Cómo funciona	Ejemplo
Skill + MCP	MCP proporciona la conexión; una skill enseña a Claude cómo usarla bien	MCP se conecta a su base de datos, una skill documenta su esquema y patrones de consulta
Skill + Subagent	Una skill genera subagents para trabajo paralelo	La skill <code>/audit</code> inicia subagents de seguridad, rendimiento y estilo que trabajan en contexto aislado
CLAUDE.md + Skills	CLAUDE.md contiene reglas siempre activas; las skills contienen material de referencia cargado bajo demanda	CLAUDE.md dice “sigue nuestras convenciones de API,” una skill contiene la guía de estilo de API completa
Hook + MCP	Un hook desencadena acciones externas a través de MCP	El hook post-edición envía una notificación de Slack cuando Claude modifica archivos críticos

Entender costos de contexto

Cada característica que agrega consume algo del contexto de Claude. Demasiado puede llenar su ventana de contexto, pero también puede agregar ruido que hace que Claude sea menos efectivo; las skills pueden no desencadenarse correctamente, o Claude puede perder de vista sus convenciones. Entender estos compromisos lo ayuda a construir una configuración efectiva.

Costo de contexto por característica

Cada característica tiene una estrategia de carga y costo de contexto diferentes:

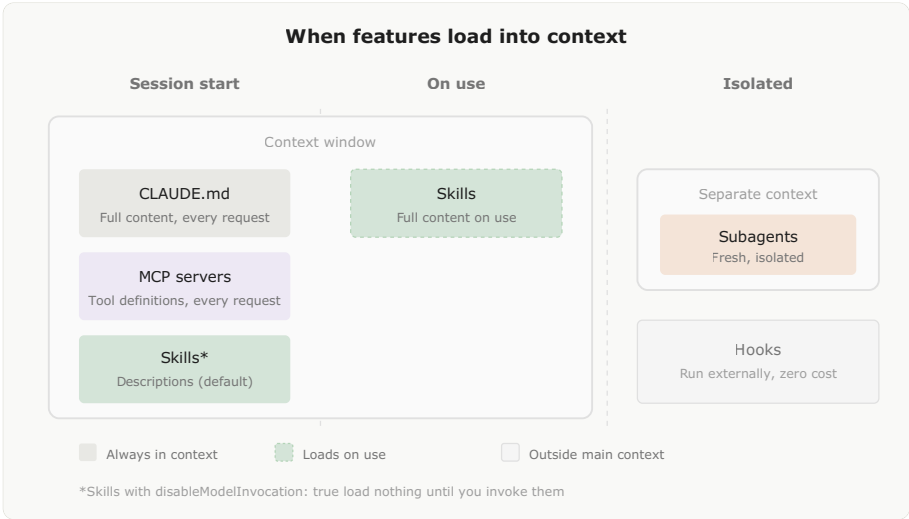
Característica	Cuándo se carga	Qué se carga	Costo de contexto
CLAUDE.md	Inicio de sesión	Contenido completo	Cada solicitud
Skills	Inicio de sesión + cuando se usa	Descripciones al inicio, contenido completo cuando se usa	Bajo (descripciones cada solicitud)*

Caracte- rística	Cuándo se carga	Qué se carga	Costo de contex- to
Servido- res MCP	Inicio de sesión	Todas las definiciones de herramientas y es- quemass	Cada solicitud
Subagen- ts	Cuando se generan	Contexto fresco con skills especificadas	Aislado de la sesión principal
Hooks	Al desencadenar	Nada (se ejecuta ex- ternamente)	Cero, a menos que el hook devuelva con- texto adicional

*Por defecto, las descripciones de skills se cargan al inicio de sesión para que Claude pueda decidir cuándo usarlas. Establezca `disable-model-invocation: true` en el front-matter de una skill para ocultarla de Claude completamente hasta que la invoque manual-mente. Esto reduce el costo de contexto a cero para las skills que solo desencadena usted mismo.

Entender cómo se cargan las características

Cada característica se carga en diferentes puntos de su sesión. Las pestañas a continuación explican cuándo se carga cada una y qué entra en contexto.



Carga de contexto: CLAUDE.md y MCP se cargan al inicio de sesión y permanecen en cada solicitud. Las skills cargan descripciones al inicio, contenido completo al invocar. Los subagents obtienen contexto aislado. Los hooks se ejecutan externamente.

CLAUDE.md

Cuándo: Inicio de sesión

Qué se carga: Contenido completo de todos los archivos CLAUDE.md (niveles administrado, usuario y proyecto).

Herencia: Claude lee archivos CLAUDE.md de su directorio de trabajo hasta la raíz, y descubre los anidados en subdirectorios mientras accede a esos archivos. Consulte [Cómo se cargan los archivos CLAUDE.md](#) para obtener detalles.

Tip:

Mantenga CLAUDE.md bajo ~500 líneas. Mueva material de referencia a skills, que se cargan bajo demanda.

Skills

Las skills son capacidades adicionales en el kit de herramientas de Claude. Pueden ser material de referencia (como una guía de estilo de API) o flujos de trabajo invocables que desencadena con `/<name>` (como `/deploy`). Claude Code se envía con [skills incluidas](#) como `/simplify`, `/batch` y `/debug` que funcionan de inmediato. También puede crear las suyas propias. Claude usa skills cuando es apropiado, o puede invocar una directamente.

Cuándo: Depende de la configuración de la skill. Por defecto, las descripciones se cargan al inicio de sesión y el contenido completo se carga cuando se usa. Para skills solo de usuario (`disable-model-invocation: true`), nada se carga hasta que las invoque.

Qué se carga: Para skills invocables por modelo, Claude ve nombres y descripciones en cada solicitud. Cuando invoca una skill con `/<name>` o Claude la carga automáticamente, el contenido completo se carga en su conversación.

Cómo Claude elige skills: Claude hace coincidir su tarea contra descripciones de skills para decidir cuáles son relevantes. Si las descripciones son vagas u se superponen, Claude puede cargar la skill incorrecta o perder una que ayudaría. Para decirle a Claude que use una skill específica, invóquela con `/<name>`. Las skills con `disable-model-invocation: true` son invisibles para Claude hasta que las invoque.

Costo de contexto: Bajo hasta que se use. Las skills solo de usuario tienen costo cero hasta que se invoquen.

En subagents: Las skills funcionan de manera diferente en subagents. En lugar de carga bajo demanda, las skills pasadas a un subagent se precarga completamente en su contexto al iniciar. Los subagents no heredan skills de la sesión principal; debe especificarlas explícitamente.

Tip:

Use `disable-model-invocation: true` para skills con efectos secundarios. Esto ahorra contexto y asegura que solo usted las desencadene.

Servidores MCP

Cuándo: Inicio de sesión.

Qué se carga: Todas las definiciones de herramientas y esquemas JSON de servidores conectados.

Costo de contexto: [Búsqueda de herramientas](#) (habilitada por defecto) carga herramientas MCP hasta el 10% del contexto y difiere el resto hasta que sea necesario.

Nota de confiabilidad: Las conexiones MCP pueden fallar silenciosamente a mitad de sesión. Si un servidor se desconecta, sus herramientas desaparecen sin advertencia. Claude puede intentar usar una herramienta que ya no existe. Si nota que Claude no puede usar una herramienta MCP a la que podía acceder anteriormente, verifique la conexión con `/mcp`.

Tip:

Ejecute `/mcp` para ver costos de token por servidor. Desconecte servidores que no esté usando activamente.

Subagents

Cuándo: Bajo demanda, cuando usted o Claude genera uno para una tarea.

Qué se carga: Contexto fresco y aislado que contiene:

- El prompt del sistema (compartido con el padre para eficiencia de caché)
- Contenido completo de skills listadas en el campo `skills:` del agente
- CLAUDE.md y estado de git (heredado del padre)
- Cualquier contexto que el agente principal pase en el prompt

Costo de contexto: Aislado de la sesión principal. Los subagents no heredan su historial de conversación o skills invocadas.

Tip:

Use subagents para trabajo que no necesita su contexto de conversación completo. Su aislamiento previene inflar su sesión principal.

Hooks

Cuándo: Al desencadenar. Los hooks se disparan en eventos de ciclo de vida específicos como ejecución de herramientas, límites de sesión, envío de prompt, solicitudes de permiso y compactación. Consulte [Hooks](#) para la lista completa.

Qué se carga: Nada por defecto. Los hooks se ejecutan como scripts externos.

Costo de contexto: Cero, a menos que el hook devuelva salida que se agregue como mensajes a su conversación.

Tip:

Los hooks son ideales para efectos secundarios (linting, logging) que no necesitan afectar el contexto de Claude.

Aprender más

Cada característica tiene su propia guía con instrucciones de configuración, ejemplos y opciones de configuración.

- [CLAUDE.md](#) Almacenar contexto de proyecto, convenciones e instrucciones
- [Skills](#) Dar a Claude experiencia de dominio y flujos de trabajo reutilizables
- [Subagents](#) Descargar trabajo a contexto aislado
- [Agent teams](#) Coordinar múltiples sesiones trabajando en paralelo
- [MCP](#) Conectar Claude a servicios externos
- [Hooks](#) Automatizar flujos de trabajo con hooks
- [Plugins](#) Empaquetar y compartir conjuntos de características
- [Marketplaces](#) Alojamiento y distribuir colecciones de plugins

Modo interactivo

Referencia completa de atajos de teclado, modos de entrada y características interactivas en sesiones de Claude Code.

Atajos de teclado

Note:

Los atajos de teclado pueden variar según la plataforma y la terminal. Presione `?` para ver los atajos disponibles en su entorno.

Usuarios de macOS: Los atajos de la tecla Option/Alt (`Alt+B` , `Alt+F` , `Alt+Y` , `Alt+M` , `Alt+P`) requieren configurar Option como Meta en su terminal:

- **iTerm2:** configuración → Perfiles → Teclas → establecer la tecla Option izquierda/derecha en “Esc+”
- **Terminal.app:** configuración → Perfiles → Teclado → marcar “Usar Option como tecla Meta”
- **VS Code:** configuración → Perfiles → Teclas → establecer la tecla Option izquierda/derecha en “Esc+”

Consulte [Configuración de terminal](#) para obtener más detalles.

Controles generales

Atajo	Descripción	Contexto
<code>Ctrl+C</code>	Cancelar entrada o generación actual	Interrupción estándar
<code>Ctrl+F</code>	Terminar todos los agentes en segundo plano. Presione dos veces en 3 segundos para confirmar	Control de agentes en segundo plano
<code>Ctrl+D</code>	Salir de la sesión de Claude Code	Señal EOF
<code>Ctrl+G</code>	Abrir en el editor de texto predeterminado	Editar su prompt o respuesta personalizada en su editor de texto predeterminado

Atajo	Descripción	Contexto
Ctrl+L	Limpiar pantalla de terminal	Mantiene el historial de conversación
Ctrl+O	Alternar salida detallada	Muestra el uso detallado de herramientas y ejecución
Ctrl+R	Búsqueda inversa del historial de comandos	Buscar a través de comandos anteriores de forma interactiva
Ctrl+V o Cmd+V (iTerm2) o Alt+V (Windows)	Pegar imagen desde el portapapeles	Pega una imagen o ruta a un archivo de imagen
Ctrl+B	Tareas en ejecución en segundo plano	Pone en segundo plano comandos bash y agentes. Los usuarios de Tmux presionan dos veces
Ctrl+T	Alternar lista de tareas	Mostrar u ocultar la lista de tareas en el área de estado de la terminal
Flechas izquierda/derecha	Ciclar a través de pestañas de diálogo	Navegar entre pestañas en diálogos de permisos y menús
Flechas arriba/abajo	Navegar por el historial de comandos	Recuperar entradas anteriores
Esc + Esc	Rebobinar o resumir	Restaurar código y/o conversación a un punto anterior, o resumir desde un mensaje seleccionado
Shift+Tab o Alt+M (algunas configuraciones)	Alternar modos de permisos	Cambiar entre Modo de Aceptación Automática, Plan Mode y modo normal.
Option+P (macOS) o Alt+P (Windows/Linux)	Cambiar modelo	Cambiar modelos sin borrar su prompt

Atajo	Descripción	Contexto
<code>Option+T</code> (macOS) o <code>Alt+T</code> (Windows/Linux)	Alternar pensamiento extendido	Habilitar o deshabilitar el modo de pensamiento extendido. Ejecute <code>/ terminal-setup</code> primero para habilitar este atajo

Edición de texto

Atajo	Descripción	Contexto
<code>Ctrl+K</code>	Eliminar hasta el final de la línea	Almacena el texto eliminado para pegarlo
<code>Ctrl+U</code>	Eliminar línea completa	Almacena el texto eliminado para pegarlo
<code>Ctrl+Y</code>	Pegar texto eliminado	Pegar texto eliminado con <code>Ctrl+K</code> o <code>Ctrl+U</code>
<code>Alt+Y</code> (después de <code>Ctrl+Y</code>)	Ciclar por el historial de pegado	Después de pegar, ciclar a través del texto eliminado anteriormente. Requiere Option como Meta en macOS
<code>Alt+B</code>	Mover cursor una palabra hacia atrás	Navegación de palabras. Requiere Option como Meta en macOS
<code>Alt+F</code>	Mover cursor una palabra hacia adelante	Navegación de palabras. Requiere Option como Meta en macOS

Tema y visualización

Atajo	Descripción	Contexto
<code>Ctrl+T</code>	Alternar resaltado de sintaxis para bloques de código	Solo funciona dentro del menú del selector <code>/ theme</code> . Controla si el código en las respuestas de Claude usa colores de sintaxis

Note:

El resaltado de sintaxis solo está disponible en la compilación nativa de Claude Code.

Entrada multilínea

Método	Atajo	Contexto
Escape rápido	<code>\ + Enter</code>	Funciona en todas las terminales
Predeterminado de macOS	<code>Option+Enter</code>	Predeterminado en macOS
Shift+Enter	<code>Shift+Enter</code>	Funciona sin configuración en iTerm2, WezTerm, Ghostty, Kitty
Secuencia de control	<code>Ctrl+J</code>	Carácter de salto de línea para multilínea
Modo de pegado	Pegar directamente	Para bloques de código, registros

Tip:

Shift+Enter funciona sin configuración en iTerm2, WezTerm, Ghostty y Kitty. Para otras terminales (VS Code, Alacritty, Zed, Warp), ejecute `/terminal-setup` para instalar el enlace.

Comandos rápidos

Atajo	Descripción	Notas
<code>/</code> al inicio	Comando o skill	Consulte comandos integrados y skills
<code>!</code> al inicio	Modo Bash	Ejecutar comandos directamente y agregar salida de ejecución a la sesión
<code>@</code>	Mención de ruta de archivo	Activar autocompletado de ruta de archivo

Comandos integrados

Escriba `/` en Claude Code para ver todos los comandos disponibles, o escriba `/` seguido de cualquier letra para filtrar. No todos los comandos son visibles para todos los usuarios. Algunos dependen de su plataforma, plan o entorno. Por ejemplo, `/desktop` solo aparece en macOS y Windows, `/upgrade` y `/privacy-settings` solo están disponibles en planes Pro y Max, y `/terminal-setup` está oculto cuando su terminal admite nativamente sus atajos de teclado.

Claude Code también incluye [skills agrupados](#) como `/simplify`, `/batch` y `/debug` que aparecen junto a comandos integrados cuando escribe `/`. Para crear sus propios comandos, consulte [skills](#).

En la tabla siguiente, `<arg>` indica un argumento requerido y `[arg]` indica uno opcional.

Comando	Propósito
<code>/add-dir</code> <code><path></code>	Agregar un nuevo directorio de trabajo a la sesión actual
<code>/agents</code>	Administrar configuraciones de agent
<code>/btw</code> <code><question></code>	Hacer una pregunta lateral rápida sin agregar a la conversación
<code>/chrome</code>	Configurar ajustes de Claude en Chrome
<code>/clear</code>	Limpiar historial de conversación y liberar contexto. Alias: <code>/reset</code> , <code>/new</code>
<code>/compact</code> <code>[instructions]</code>	Compactar conversación con instrucciones de enfoque opcionales
<code>/config</code>	Abrir la interfaz de Configuración para ajustar tema, modelo, estilo de salida y otras preferencias. Alias: <code>/settings</code>
<code>/context</code>	Visualizar el uso actual del contexto como una cuadrícula de colores
<code>/copy</code>	Copiar la última respuesta del asistente al portapapeles. Cuando hay bloques de código presentes, muestra un selector interactivo para seleccionar bloques individuales o la respuesta completa
<code>/cost</code>	Mostrar estadísticas de uso de tokens. Consulte guía de seguimiento de costos para detalles específicos de suscripción
<code>/desktop</code>	Continuar la sesión actual en la aplicación de escritorio de Claude Code. Solo macOS y Windows. Alias: <code>/app</code>

Comando	Propósito
<code>/diff</code>	Abrir un visor de diferencias interactivo que muestra cambios sin confirmar y diferencias por turno. Use las flechas izquierda/derecha para cambiar entre el diff de git actual y los turnos individuales de Claude, y arriba/abajo para examinar archivos
<code>/doctor</code>	Diagnosticar y verificar su instalación y configuración de Claude Code
<code>/exit</code>	Salir de la CLI. Alias: <code>/quit</code>
<code>/export [filename]</code>	Exportar la conversación actual como texto sin formato. Con un nombre de archivo, escribe directamente en ese archivo. Sin uno, abre un diálogo para copiar al portapapeles o guardar en un archivo
<code>/extra-usage</code>	Configurar uso extra para continuar trabajando cuando se alcanzan los límites de velocidad
<code>/fast [on off]</code>	Alternar modo rápido activado o desactivado
<code>/feedback [report]</code>	Enviar comentarios sobre Claude Code. Alias: <code>/bug</code>
<code>/fork [name]</code>	Crear una bifurcación de la conversación actual en este punto
<code>/help</code>	Mostrar ayuda y comandos disponibles
<code>/hooks</code>	Administrar configuraciones de hook para eventos de herramientas
<code>/ide</code>	Administrar integraciones de IDE y mostrar estado
<code>/init</code>	Inicializar proyecto con guía <code>CLAUDE.md</code>
<code>/insights</code>	Generar un informe que analice sus sesiones de Claude Code, incluidas áreas de proyecto, patrones de interacción y puntos de fricción
<code>/install-github-app</code>	Configurar la aplicación Claude GitHub Actions para un repositorio. Lo guía a través de la selección de un repositorio y la configuración de la integración
<code>/install-slack-app</code>	Instalar la aplicación Claude Slack. Abre un navegador para completar el flujo OAuth
<code>/keybindings</code>	Abrir o crear su archivo de configuración de atajos de teclado
<code>/login</code>	Iniciar sesión en su cuenta de Anthropic
<code>/logout</code>	Cerrar sesión de su cuenta de Anthropic

Comando	Propósito
<code>/mcp</code>	Administrar conexiones de servidor MCP y autenticación OAuth
<code>/memory</code>	Editar archivos de memoria <code>CLAUDE.md</code> , habilitar o deshabilitar auto-memory y ver entradas de auto-memory
<code>/mobile</code>	Mostrar código QR para descargar la aplicación móvil de Claude. Alias: <code>/ios</code> , <code>/android</code>
<code>/model [model]</code>	Seleccionar o cambiar el modelo de IA. Para modelos que lo admiten, use las flechas izquierda/derecha para ajustar el nivel de esfuerzo . El cambio entra en vigor inmediatamente sin esperar a que se complete la respuesta actual
<code>/passes</code>	Compartir una semana gratuita de Claude Code con amigos. Solo visible si su cuenta es elegible
<code>/permissions</code>	Ver o actualizar permisos . Alias: <code>/allowed-tools</code>
<code>/plan</code>	Entrar en Plan Mode directamente desde el prompt
<code>/plugin</code>	Administrar plugins de Claude Code
<code>/pr-comments [PR]</code>	Obtener y mostrar comentarios de una solicitud de extracción de GitHub. Detecta automáticamente el PR para la rama actual, o pase una URL o número de PR. Requiere la CLI <code>gh</code>
<code>/privacy-settings</code>	Ver y actualizar su configuración de privacidad. Solo disponible para suscriptores del plan Pro y Max
<code>/release-notes</code>	Ver el registro de cambios completo, con la versión más reciente más cercana a su prompt
<code>/reload-plugins</code>	Recargar todos los plugins activos para aplicar cambios pendientes sin reiniciar. Informa qué se cargó y anota los cambios que requieren un reinicio
<code>/remote-control</code>	Hacer que esta sesión esté disponible para control remoto desde claude.ai. Alias: <code>/rc</code>
<code>/remote-env</code>	Configurar el entorno remoto predeterminado para sesiones de teleport
<code>/rename [name]</code>	Renombrar la sesión actual. Sin un nombre, genera automáticamente uno a partir del historial de conversación
<code>/resume [session]</code>	Reanudar una conversación por ID o nombre, o abrir el selector de sesión. Alias: <code>/continue</code>

Comando	Propósito
<code>/review</code>	Obsoleto. Instale el plugin code-review en su lugar: <code>claude plugin install code-review@claude-code-marketplace</code>
<code>/rewind</code>	Rebobinar la conversación y/o código a un punto anterior, o resumir desde un mensaje seleccionado. Consulte checkpointing . Alias: <code>/checkpoint</code>
<code>/sandbox</code>	Alternar modo sandbox . Disponible solo en plataformas compatibles
<code>/security-review</code>	Analizar cambios pendientes en la rama actual para detectar vulnerabilidades de seguridad. Revisa el diff de git e identifica riesgos como inyección, problemas de autenticación y exposición de datos
<code>/skills</code>	Listar skills disponibles
<code>/stats</code>	Visualizar uso diario, historial de sesiones, rachas y preferencias de modelo
<code>/status</code>	Abrir la interfaz de Configuración (pestaña Estado) mostrando versión, modelo, cuenta y conectividad
<code>/statusline</code>	Configurar la línea de estado de Claude Code. Describa lo que desea, o ejecute sin argumentos para auto-configurar desde su prompt de shell
<code>/stickers</code>	Pedir pegatinas de Claude Code
<code>/tasks</code>	Listar y administrar tareas en segundo plano
<code>/terminal-setup</code>	Configurar atajos de teclado de terminal para Shift+Enter y otros atajos. Solo visible en terminales que lo necesitan, como VS Code, Alacritty o Warp
<code>/theme</code>	Cambiar el tema de color. Incluye variantes claras y oscuras, temas accesibles para daltónicos (daltonizados) y temas ANSI que usan la paleta de colores de su terminal
<code>/upgrade</code>	Abrir la página de actualización para cambiar a un nivel de plan superior
<code>/usage</code>	Mostrar límites de uso del plan y estado del límite de velocidad
<code>/vim</code>	Alternar entre modos de edición Vim y Normal

Prompts de MCP

Los servidores MCP pueden exponer prompts que aparecen como comandos. Estos usan el formato `/mcp__<server>__<prompt>` y se descubren dinámicamente desde servidores conectados. Consulte [prompts de MCP](#) para obtener detalles.

Modo editor Vim

Habilitar edición de estilo vim con el comando `/vim` o configurar permanentemente a través de `/config`.

Cambio de modo

Comando	Acción	Desde modo
<code>Esc</code>	Entrar en modo NORMAL	INSERT
<code>i</code>	Insertar antes del cursor	NORMAL
<code>I</code>	Insertar al principio de la línea	NORMAL
<code>a</code>	Insertar después del cursor	NORMAL
<code>A</code>	Insertar al final de la línea	NORMAL
<code>o</code>	Abrir línea debajo	NORMAL
<code>O</code>	Abrir línea arriba	NORMAL

Navegación (modo NORMAL)

Comando	Acción
<code>h / j / k / l</code>	Mover izquierda/abajo/arriba/derecha
<code>w</code>	Siguiente palabra
<code>e</code>	Final de palabra
<code>b</code>	Palabra anterior
<code>0</code>	Principio de línea
<code>\$</code>	Final de línea
<code>^</code>	Primer carácter no en blanco
<code>gg</code>	Principio de entrada
<code>G</code>	Final de entrada
<code>f{char}</code>	Saltar a la siguiente ocurrencia del carácter
<code>F{char}</code>	Saltar a la ocurrencia anterior del carácter

Comando	Acción
<code>t{char}</code>	Saltar justo antes de la siguiente ocurrencia del carácter
<code>T{char}</code>	Saltar justo después de la ocurrencia anterior del carácter
<code>;</code>	Repetir último movimiento f/F/t/T
<code>,</code>	Repetir último movimiento f/F/t/T en orden inverso

Note:

En modo normal de vim, si el cursor está al principio o al final de la entrada y no puede moverse más, las flechas de dirección navegan por el historial de comandos en su lugar.

Edición (modo NORMAL)

Comando	Acción
<code>x</code>	Eliminar carácter
<code>dd</code>	Eliminar línea
<code>D</code>	Eliminar hasta el final de la línea
<code>dw / de / db</code>	Eliminar palabra/hasta el final/hacia atrás
<code>cc</code>	Cambiar línea
<code>C</code>	Cambiar hasta el final de la línea
<code>cw / ce / cb</code>	Cambiar palabra/hasta el final/hacia atrás
<code>yy / Y</code>	Yanquear (copiar) línea
<code>yw / ye / yb</code>	Yanquear palabra/hasta el final/hacia atrás
<code>p</code>	Pegar después del cursor
<code>P</code>	Pegar antes del cursor
<code>>></code>	Indentar línea
<code><<</code>	Desindentación de línea
<code>J</code>	Unir líneas

Comando	Acción
.	Repetir último cambio

Objetos de texto (modo NORMAL)

Los objetos de texto funcionan con operadores como **d**, **c** e **y**:

Co-mando	Acción
iw / aw	Palabra interior/alrededor
iW / aW	PALABRA interior/alrededor (delimitada por espacios en blanco)
i" / a"	Comillas dobles interior/alrededor
i' / a'	Comillas simples interior/alrededor
i(/ a(	Paréntesis interior/alrededor
i[/ a[	Corchetes interior/alrededor
i{ / a{	Llaves interior/alrededor

Historial de comandos

Claude Code mantiene el historial de comandos para la sesión actual:

- El historial de entrada se almacena por directorio de trabajo
- El historial de entrada se reinicia cuando ejecuta **/clear** para iniciar una nueva sesión. La conversación de la sesión anterior se conserva y se puede reanudar.
- Use las flechas arriba/abajo para navegar (consulte los atajos de teclado anteriores)
- **Nota:** la expansión del historial (**!**) está deshabilitada de forma predeterminada

Búsqueda inversa con Ctrl+R

Presione **Ctrl+R** para buscar de forma interactiva a través de su historial de comandos:

1. **Iniciar búsqueda:** presione **Ctrl+R** para activar la búsqueda inversa del historial
2. **Escribir consulta:** ingrese texto para buscar en comandos anteriores. El término de búsqueda se resalta en los resultados coincidentes
3. **Navegar coincidencias:** presione **Ctrl+R** nuevamente para ciclar a través de coincidencias más antiguas

4. Aceptar coincidencia:

- Presione **Tab** o **Esc** para aceptar la coincidencia actual y continuar editando
- Presione **Enter** para aceptar y ejecutar el comando inmediatamente

5. Cancelar búsqueda:

- Presione **Ctrl+C** para cancelar y restaurar su entrada original
- Presione **Backspace** en búsqueda vacía para cancelar

La búsqueda muestra comandos coincidentes con el término de búsqueda resaltado, para que pueda encontrar y reutilizar entradas anteriores.

Comandos bash en segundo plano

Claude Code admite la ejecución de comandos bash en segundo plano, lo que le permite continuar trabajando mientras se ejecutan procesos de larga duración.

Cómo funciona el segundo plano

Cuando Claude Code ejecuta un comando en segundo plano, ejecuta el comando de forma asíncrona e inmediatamente devuelve un ID de tarea en segundo plano. Claude Code puede responder a nuevos prompts mientras el comando continúa ejecutándose en segundo plano.

Para ejecutar comandos en segundo plano, puede:

- Solicitar a Claude Code que ejecute un comando en segundo plano
- Presionar Ctrl+B para mover una invocación de herramienta Bash regular al segundo plano. (Los usuarios de Tmux deben presionar Ctrl+B dos veces debido a la tecla de prefijo de tmux).

Características clave:

- La salida se almacena en búfer y Claude puede recuperarla usando la herramienta TaskOutput
- Las tareas en segundo plano tienen ID únicos para seguimiento y recuperación de salida
- Las tareas en segundo plano se limpian automáticamente cuando Claude Code sale

Para deshabilitar toda la funcionalidad de tareas en segundo plano, establezca la variable de entorno `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` en `1`. Consulte [Variables de entorno](#) para obtener detalles.

Comandos comúnmente puestos en segundo plano:

- Herramientas de compilación (webpack, vite, make)
- Administradores de paquetes (npm, yarn, pnpm)
- Ejecutores de pruebas (jest, pytest)
- Servidores de desarrollo
- Procesos de larga duración (docker, terraform)

Modo Bash con prefijo **!**

Ejecutar comandos bash directamente sin pasar por Claude prefijando su entrada con **!**:

```
! npm test
! git status
! ls -la
```

Modo Bash:

- Agrega el comando y su salida al contexto de conversación
- Muestra progreso y salida en tiempo real
- Admite el mismo segundo plano **Ctrl+B** para comandos de larga duración
- No requiere que Claude interprete o apruebe el comando
- Admite autocompletado basado en historial: escriba un comando parcial y presione **Tab** para completar desde comandos **!** anteriores en el proyecto actual
- Salir con **Escape**, **Backspace** o **Ctrl+U** en un prompt vacío

Esto es útil para operaciones rápidas de shell mientras se mantiene el contexto de conversación.

Sugerencias de prompt

Cuando abre una sesión por primera vez, aparece un comando de ejemplo atenuado en la entrada del prompt para ayudarlo a comenzar. Claude Code elige esto del historial de git de su proyecto, por lo que refleja archivos en los que ha estado trabajando recientemente.

Después de que Claude responde, las sugerencias continúan apareciendo según su historial de conversación, como un paso de seguimiento de una solicitud de varias partes o una continuación natural de su flujo de trabajo.

- Presione **Tab** para aceptar la sugerencia, o presione **Enter** para aceptar y enviar
- Comience a escribir para descartarla

La sugerencia se ejecuta como una solicitud en segundo plano que reutiliza el caché de prompt de la conversación principal, por lo que el costo adicional es mínimo. Claude Code omite la generación de sugerencias cuando el caché está frío para evitar costos innecesarios.

Las sugerencias se omiten automáticamente después del primer turno de una conversación, en modo no interactivo y en Plan Mode.

Para deshabilitar completamente las sugerencias de prompt, establezca la variable de entorno o alterne la configuración en `/config`:

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

Preguntas laterales con `/btw`

Use `/btw` para hacer una pregunta rápida sobre su trabajo actual sin agregar al historial de conversación. Esto es útil cuando desea una respuesta rápida pero no desea saturar el contexto principal o desviar a Claude de una tarea de larga duración.

```
/btw what was the name of that config file again?
```

Las preguntas laterales tienen visibilidad completa de la conversación actual, por lo que puede preguntar sobre código que Claude ya ha leído, decisiones que tomó anteriormente, o cualquier otra cosa de la sesión. La pregunta y la respuesta son efímeras: aparecen en una superposición descartable y nunca entran en el historial de conversación.

- **Disponible mientras Claude está trabajando:** puede ejecutar `/btw` incluso mientras Claude está procesando una respuesta. La pregunta lateral se ejecuta de forma independiente y no interrumpe el turno principal.
- **Sin acceso a herramientas:** las preguntas laterales responden solo a partir de lo que ya está en contexto. Claude no puede leer archivos, ejecutar comandos o buscar al responder una pregunta lateral.
- **Respuesta única:** no hay turnos de seguimiento. Si necesita una conversación de ida y vuelta, use un prompt normal en su lugar.
- **Bajo costo:** la pregunta lateral reutiliza el caché de prompt de la conversación principal, por lo que el costo adicional es mínimo.

Presione **Espacio**, **Enter** o **Escape** para descartar la respuesta y volver al prompt.

`/btw` es lo opuesto a un `subagent`: ve su conversación completa pero no tiene herramientas, mientras que un `subagent` tiene herramientas completas pero comienza con un contexto vacío. Use `/btw` para preguntar sobre lo que Claude ya sabe de esta sesión; use un `subagent` para descubrir algo nuevo.

Lista de tareas

Cuando trabaja en trabajo complejo y de varios pasos, Claude crea una lista de tareas para rastrear el progreso. Las tareas aparecen en el área de estado de su terminal con indicadores que muestran qué está pendiente, en progreso o completado.

- Presione `Ctrl+T` para alternar la vista de la lista de tareas. La pantalla muestra hasta 10 tareas a la vez
- Para ver todas las tareas o borrarlas, pregunte a Claude directamente: “show me all tasks” o “clear all tasks”
- Las tareas persisten en compactaciones de contexto, ayudando a Claude a mantenerse organizado en proyectos más grandes
- Para compartir una lista de tareas entre sesiones, establezca `CLAUDE_CODE_TASK_LIST_ID` para usar un directorio nombrado en `~/.claude/tasks/`:
`CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- Para revertir a la lista TODO anterior, establezca `CLAUDE_CODE_ENABLE_TASKS=false`.

Estado de revisión de PR

Cuando trabaja en una rama con una solicitud de extracción abierta, Claude Code muestra un enlace de PR en el pie de página (por ejemplo, “PR #446”). El enlace tiene un subrayado de color que indica el estado de revisión:

- Verde: aprobado
- Amarillo: revisión pendiente
- Rojo: cambios solicitados
- Gris: borrador
- Púrpura: fusionado

`Cmd+click` (Mac) o `Ctrl+click` (Windows/Linux) en el enlace para abrir la solicitud de extracción en su navegador. El estado se actualiza automáticamente cada 60 segundos.

Note:

El estado de PR requiere que la CLI `gh` esté instalada y autenticada (`gh auth login`).

Ver también

- [Skills](#) - Prompts personalizados y flujos de trabajo
- [Checkpointing](#) - Rebobinar ediciones de Claude y restaurar estados anteriores
- [Referencia de CLI](#) - Banderas y opciones de línea de comandos
- [Configuración](#) - Opciones de configuración
- [Gestión de memoria](#) - Administración de archivos CLAUDE.md

Mejores prácticas para Claude Code

Consejos y patrones para aprovechar al máximo Claude Code, desde configurar su entorno hasta escalar entre sesiones paralelas.

Claude Code es un entorno de codificación agencial. A diferencia de un chatbot que responde preguntas y espera, Claude Code puede leer sus archivos, ejecutar comandos, hacer cambios y trabajar autónomamente a través de problemas mientras usted observa, redirige o se aleja completamente.

Esto cambia cómo trabaja. En lugar de escribir código usted mismo y pedirle a Claude que lo revise, describe lo que desea y Claude descubre cómo construirlo. Claude explora, planifica e implementa.

Pero esta autonomía aún viene con una curva de aprendizaje. Claude trabaja dentro de ciertas restricciones que necesita entender.

Esta guía cubre patrones que han demostrado ser efectivos en los equipos internos de Anthropic y para ingenieros que usan Claude Code en varios códigos base, lenguajes y entornos. Para saber cómo funciona el bucle agencial bajo el capó, consulte [Cómo funciona Claude Code](#).

La mayoría de las mejores prácticas se basan en una restricción: la ventana de contexto de Claude se llena rápidamente y el rendimiento se degrada a medida que se llena.

La ventana de contexto de Claude contiene toda su conversación, incluido cada mensaje, cada archivo que Claude lee y cada salida de comando. Sin embargo, esto puede llenarse rápidamente. Una única sesión de depuración o exploración de código base podría generar y consumir decenas de miles de tokens.

Esto importa porque el rendimiento del LLM se degrada a medida que se llena el contexto. Cuando la ventana de contexto se está llenando, Claude puede comenzar a “olvidar” instrucciones anteriores o cometer más errores. La ventana de contexto es el recurso más importante a gestionar. Rastree el uso de contexto continuamente con una [línea de estado personalizada](#), y consulte [Reducir el uso de tokens](#) para estrategias sobre cómo reducir el uso de tokens.

Dé a Claude una forma de verificar su trabajo

Tip:

Incluya pruebas, capturas de pantalla o salidas esperadas para que Claude pueda verificarse a sí mismo. Esta es la cosa de mayor apalancamiento que puede hacer.

Claude funciona dramáticamente mejor cuando puede verificar su propio trabajo, como ejecutar pruebas, comparar capturas de pantalla y validar salidas.

Sin criterios de éxito claros, podría producir algo que se vea bien pero que en realidad no funcione. Usted se convierte en el único bucle de retroalimentación, y cada error requiere su atención.

Estrategia	Antes	Después
Proporcionar criterios de verificación	<i>“implementar una función que valide direcciones de correo electrónico”</i>	<i>“escribir una función validateEmail. casos de prueba de ejemplo: user@example.com es verdadero, inválido es falso, user@.com es falso. ejecutar las pruebas después de implementar”</i>
Verificar cambios de UI visualmente	<i>“hacer que el panel de control se vea mejor”</i>	<i>“[pegar captura de pantalla] implementar este diseño. tomar una captura de pantalla del resultado y compararla con la original. listar diferencias y corregirlas”</i>
Abordar causas raíz, no síntomas	<i>“la compilación está fallando”</i>	<i>“la compilación falla con este error: [pegar error]. corregirlo y verificar que la compilación tenga éxito. abordar la causa raíz, no suprimir el error”</i>

Los cambios de UI se pueden verificar usando la [extensión Claude en Chrome](#). Abre nuevas pestañas en su navegador, prueba la UI e itera hasta que el código funcione.

Su verificación también puede ser un conjunto de pruebas, un linter o un comando Bash que verifique la salida. Invierta en hacer que su verificación sea sólida.

Explore primero, luego planifique, luego codifique

Tip:

Separe la investigación y la planificación de la implementación para evitar resolver el problema incorrecto.

Dejar que Claude salte directamente a la codificación puede producir código que resuelve el problema incorrecto. Use [Plan Mode](#) para separar la exploración de la ejecución.

El flujo de trabajo recomendado tiene cuatro fases:

Step 1: Explorar

Ingrese Plan Mode. Claude lee archivos y responde preguntas sin hacer cambios.

```
read /src/auth and understand how we handle sessions and login.  
also look at how we manage environment variables for secrets.
```

Step 2: Planificar

Pida a Claude que cree un plan de implementación detallado.

```
I want to add Google OAuth. What files need to change?  
What's the session flow? Create a plan.
```

Presione **Ctrl+G** para abrir el plan en su editor de texto para edición directa antes de que Claude continúe.

Step 3: Implementar

Vuelva al Modo Normal y deje que Claude codifique, verificando contra su plan.

```
implement the OAuth flow from your plan. write tests for the  
callback handler, run the test suite and fix any failures.
```

Step 4: Confirmar

Pida a Claude que confirme con un mensaje descriptivo y cree un PR.

```
commit with a descriptive message and open a PR
```

Plan Mode es útil, pero también agrega sobrecarga.

Para tareas donde el alcance es claro y la corrección es pequeña (como corregir un error tipográfico, agregar una línea de registro o renombrar una variable) pida a Claude que lo haga directamente.

La planificación es más útil cuando no está seguro del enfoque, cuando el cambio modifica múltiples archivos o cuando no está familiarizado con el código que se está modificando. Si pudiera describir el diff en una oración, omita el plan.

Proporcione contexto específico en sus indicaciones

Tip:

Cuanto más precisas sean sus instrucciones, menos correcciones necesitará.

Claude puede inferir intención, pero no puede leer su mente. Haga referencia a archivos específicos, mencione restricciones y señale patrones de ejemplo.

Estrategia	Antes	Después
Delimitar la tarea. Especifique qué archivo, qué escenario y preferencias de prueba.	<i>“agregar pruebas para foo.py”</i>	<i>“escribir una prueba para foo.py cubriendo el caso extremo donde el usuario ha cerrado sesión. evitar mocks.”</i>
Señalar fuentes. Dirija a Claude a la fuente que puede responder una pregunta.	<i>“¿por qué ExecutionFactory tiene una API tan extraña?”</i>	<i>“revisar el historial de git de ExecutionFactory y resumir cómo su API llegó a ser así”</i>

Estrategia	Antes	Después
Hacer referencia a patrones existentes. Señale a Claude los patrones en su código base.	<i>“agregar un widget de calendario”</i>	<i>“ver cómo se implementan los widgets existentes en la página de inicio para entender los patrones. HotDogWidget.php es un buen ejemplo. seguir el patrón para implementar un nuevo widget de calendario que permita al usuario seleccionar un mes y paginar hacia adelante/atrás para elegir un año. construir desde cero sin bibliotecas que no sean las ya utilizadas en el código base.”</i>
Describir el síntoma. Proporcione el síntoma, la ubicación probable y qué significa “corregido”.	<i>“corregir el error de inicio de sesión”</i>	<i>“los usuarios informan que el inicio de sesión falla después del agotamiento de la sesión. verificar el flujo de autenticación en src/auth/, especialmente la actualización de tokens. escribir una prueba fallida que reproduzca el problema, luego corregirlo”</i>

Las indicaciones vagas pueden ser útiles cuando está explorando y puede permitirse corregir el curso. Una indicación como `"¿qué mejoraría en este archivo?"` puede revelar cosas en las que no habría pensado en preguntar.

Proporcionar contenido enriquecido

Tip:

Use `@` para hacer referencia a archivos, pegue capturas de pantalla/imágenes o canalice datos directamente.

Puede proporcionar datos enriquecidos a Claude de varias maneras:

- **Haga referencia a archivos con @** en lugar de describir dónde vive el código. Claude lee el archivo antes de responder.
- **Pegue imágenes directamente.** Copie/pegue o arrastre y suelte imágenes en la indicación.
- **Proporcione URLs** para documentación y referencias de API. Use `/permissions` para permitir dominios de uso frecuente.
- **Canalice datos** ejecutando `cat error.log | claude` para enviar contenidos de archivo directamente.
- **Deje que Claude obtenga lo que necesita.** Diga a Claude que extraiga contexto por sí mismo usando comandos Bash, herramientas MCP o leyendo archivos.

Configure su entorno

Algunos pasos de configuración hacen que Claude Code sea significativamente más efectivo en todas sus sesiones. Para una descripción general completa de las características de extensión y cuándo usar cada una, consulte [Extender Claude Code](#).

Escriba un CLAUDE.md efectivo

Tip:

Ejecute `/init` para generar un archivo CLAUDE.md inicial basado en la estructura de su proyecto actual, luego refine con el tiempo.

CLAUDE.md es un archivo especial que Claude lee al inicio de cada conversación. Incluya comandos Bash, estilo de código y reglas de flujo de trabajo. Esto le da a Claude contexto persistente que no puede inferir solo del código.

El comando `/init` analiza su código base para detectar sistemas de compilación, marcos de prueba y patrones de código, dándole una base sólida para refinar.

No hay un formato requerido para archivos CLAUDE.md, pero manténgalo corto y legible por humanos. Por ejemplo:

```
## Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from 'bar')

## Workflow
- Be sure to typecheck when you're done making a series of code changes
- Prefer running single tests, and not the whole test suite, for performance
```

CLAUDE.md se carga cada sesión, así que solo incluya cosas que se apliquen ampliamente. Para conocimiento de dominio o flujos de trabajo que solo son relevantes a veces, use [skills](#) en su lugar. Claude los carga bajo demanda sin inflar cada conversación.

Manténgalo conciso. Para cada línea, pregúntese: “¿Causaría que Claude cometiera errores si elimino esto?” Si no, elimínelo. Los archivos CLAUDE.md inflados hacen que Claude ignore sus instrucciones reales.

✔ Incluir	✖ Excluir
Comandos Bash que Claude no puede adivinar	Cualquier cosa que Claude pueda descubrir leyendo código
Reglas de estilo de código que difieren de los valores predeterminados	Convenciones de lenguaje estándar que Claude ya conoce
Instrucciones de prueba y ejecutores de prueba preferidos	Documentación detallada de API (enlace a documentos en su lugar)
Etiqueta del repositorio (nomenclatura de rama, convenciones de PR)	Información que cambia frecuentemente
Decisiones arquitectónicas específicas de su proyecto	Explicaciones largas o tutoriales
Peculiaridades del entorno de desarrollo (variables de entorno requeridas)	Prácticas evidentes por sí solas como “escribir código limpio”
Errores comunes o comportamientos no obvios	

Si Claude sigue haciendo algo que no desea a pesar de tener una regla en su contra, el archivo probablemente sea demasiado largo y la regla se está perdiendo. Si Claude le hace preguntas que se responden en CLAUDE.md, la redacción podría ser ambigua. Trate CLAUDE.md como código: revíselo cuando las cosas salgan mal, elimine regularmente y pruebe cambios observando si el comportamiento de Claude realmente cambia.

Puede ajustar instrucciones agregando énfasis (por ejemplo, “**IMPORTANTE**” o “**DEBE**”) para mejorar la adherencia. Verifique CLAUDE.md en git para que su equipo pueda contribuir. El archivo se compone en valor con el tiempo.

Los archivos CLAUDE.md pueden importar archivos adicionales usando la sintaxis `@path/to/import`:

```
See @README.md for project overview and @package.json for available npm commands.
```

```
## Additional Instructions
```

- Git workflow: @docs/git-instructions.md
- Personal overrides: @~/ .claude/my-project-instructions.md

Puede colocar archivos CLAUDE.md en varias ubicaciones:

- **Carpeta de inicio** (`~/ .claude/CLAUDE.md`): se aplica a todas las sesiones de Claude
- **Raíz del proyecto** (`./CLAUDE.md`): verificar en git para compartir con su equipo
- **Directorios principales**: útil para monorepos donde tanto `root/CLAUDE.md` como `root/foo/CLAUDE.md` se extraen automáticamente
- **Directorios secundarios**: Claude extrae archivos CLAUDE.md secundarios bajo demanda cuando trabaja con archivos en esos directorios

Configurar permisos

Tip:

Use `/permissions` para permitir comandos seguros o `/sandbox` para aislamiento a nivel del SO. Esto reduce interrupciones mientras lo mantiene en control.

De forma predeterminada, Claude Code solicita permiso para acciones que podrían modificar su sistema: escrituras de archivo, comandos Bash, herramientas MCP, etc. Esto es seguro pero tedioso. Después de la décima aprobación, realmente no está revisando, solo está haciendo clic. Hay dos formas de reducir estas interrupciones:

- **Listas de permisos**: permitir herramientas específicas que sabe que son seguras (como `npm run lint` o `git commit`)
- **Sandboxing**: habilitar aislamiento a nivel del SO que restrinja el acceso al sistema de archivos y red, permitiendo que Claude trabaje más libremente dentro de límites definidos

Alternativamente, use `--dangerously-skip-permissions` para omitir todas las verificaciones de permiso para flujos de trabajo contenidos como corregir errores de lint o generar código repetitivo.

Warning:

Dejar que Claude ejecute comandos arbitrarios puede resultar en pérdida de datos, corrupción del sistema o exfiltración de datos a través de inyección de indicaciones. Solo use `--dangerously-skip-permissions` en un sandbox sin acceso a internet.

Lea más sobre [configurar permisos](#) y [habilitar sandboxing](#).

Usar herramientas CLI

Tip:

Diga a Claude Code que use herramientas CLI como `gh`, `aws`, `gcloud` y `sentry-cli` cuando interactúe con servicios externos.

Las herramientas CLI son la forma más eficiente en contexto de interactuar con servicios externos. Si usa GitHub, instale la CLI `gh`. Claude sabe cómo usarla para crear problemas, abrir solicitudes de extracción y leer comentarios. Sin `gh`, Claude aún puede usar la API de GitHub, pero las solicitudes no autenticadas a menudo alcanzan límites de velocidad.

Claude también es efectivo en aprender herramientas CLI que no conoce. Intente indicaciones como `Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.`

Conectar servidores MCP

Tip:

Ejecute `claude mcp add` para conectar herramientas externas como Notion, Figma o su base de datos.

Con [servidores MCP](#), puede pedir a Claude que implemente características desde rastreadores de problemas, consulte bases de datos, analice datos de monitoreo, integre diseños de Figma y automatice flujos de trabajo.

Configurar hooks

Tip:

Use hooks para acciones que deben suceder cada vez sin excepciones.

[Hooks](#) ejecutan scripts automáticamente en puntos específicos del flujo de trabajo de Claude. A diferencia de las instrucciones CLAUDE.md que son consultivas, los hooks son deterministas y garantizan que la acción suceda.

Claude puede escribir hooks para usted. Intente indicaciones como *“Write a hook that runs eslint after every file edit”* o *“Write a hook that blocks writes to the migrations folder.”* Ejecute `/hooks` para configuración interactiva, o edite `.claude/settings.json` directamente.

Crear skills

Tip:

Cree archivos `SKILL.md` en `.claude/skills/` para dar a Claude conocimiento de dominio y flujos de trabajo reutilizables.

[Skills](#) extienden el conocimiento de Claude con información específica de su proyecto, equipo o dominio. Claude los aplica automáticamente cuando son relevantes, o puede invocarlos directamente con `/skill-name`.

Cree una skill agregando un directorio con un `SKILL.md` a `.claude/skills/`:

```
---
name: api-conventions
description: REST API design conventions for our services
---

## API Conventions

- Use kebab-case for URL paths
- Use camelCase for JSON properties
- Always include pagination for list endpoints
- Version APIs in the URL path (/v1/, /v2/)
```

Las skills también pueden definir flujos de trabajo repetibles que invoca directamente:

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Analyze and fix the GitHub issue: $ARGUMENTS.

1. Use `gh issue view` to get the issue details
2. Understand the problem described in the issue
3. Search the codebase for relevant files
4. Implement the necessary changes to fix the issue
5. Write and run tests to verify the fix
6. Ensure code passes linting and type checking
7. Create a descriptive commit message
8. Push and create a PR
```

Ejecute `/fix-issue 1234` para invocarlo. Use `disable-model-invocation: true` para flujos de trabajo con efectos secundarios que desea activar manualmente.

Crear subagents personalizados

Tip:

Defina asistentes especializados en `.claude/agents/` que Claude pueda delegar para tareas aisladas.

[Subagents](#) se ejecutan en su propio contexto con su propio conjunto de herramientas permitidas. Son útiles para tareas que leen muchos archivos o necesitan enfoque especializado sin saturar su conversación principal.

```
---  
name: security-reviewer  
description: Reviews code for security vulnerabilities  
tools: Read, Grep, Glob, Bash  
model: opus  
---  
  
You are a senior security engineer. Review code for:  
- Injection vulnerabilities (SQL, XSS, command injection)  
- Authentication and authorization flaws  
- Secrets or credentials in code  
- Insecure data handling  
  
Provide specific line references and suggested fixes.
```

Diga a Claude que use subagents explícitamente: *“Use a subagent to review this code for security issues.”*

Instalar plugins

Tip:

Ejecute `/plugin` para explorar el marketplace. Los plugins agregan skills, herramientas e integraciones sin configuración.

[Plugins](#) agrupan skills, hooks, subagents y servidores MCP en una única unidad instalable de la comunidad y Anthropic. Si trabaja con un lenguaje tipado, instale un [plugin de inteligencia de código](#) para dar a Claude navegación de símbolos precisa y detección automática de errores después de ediciones.

Para orientación sobre cómo elegir entre skills, subagents, hooks y MCP, consulte [Extender Claude Code](#).

Comuníquese efectivamente

La forma en que se comunica con Claude Code impacta significativamente la calidad de los resultados.

Haga preguntas sobre el código base

Tip:

Haga a Claude preguntas que haría a un ingeniero senior.

Al incorporarse a un nuevo código base, use Claude Code para aprender y explorar. Puede hacer a Claude el mismo tipo de preguntas que haría a otro ingeniero:

- ¿Cómo funciona el registro?
- ¿Cómo hago un nuevo punto final de API?
- ¿Qué hace `async move { ... }` en la línea 134 de `foo.rs`?
- ¿Qué casos extremos maneja `CustomerOnboardingFlowImpl`?
- ¿Por qué este código llama a `foo()` en lugar de `bar()` en la línea 333?

Usar Claude Code de esta manera es un flujo de trabajo de incorporación efectivo, mejorando el tiempo de rampa y reduciendo la carga en otros ingenieros. No se requiere indicación especial: haga preguntas directamente.

Deje que Claude lo entreviste

Tip:

Para características más grandes, deje que Claude lo entreviste primero. Comience con una indicación mínima y pida a Claude que lo entreviste usando la herramienta

`AskUserQuestion`.

Claude hace preguntas sobre cosas que podría no haber considerado, incluyendo implementación técnica, UI/UX, casos extremos y compensaciones.

I want to build [brief description]. Interview me in detail using the `AskUserQuestion` tool.

Ask about technical implementation, UI/UX, edge cases, concerns, and tradeoffs. Don't ask obvious questions, dig into the hard parts I might not have considered.

Keep interviewing until we've covered everything, then write a complete spec to `SPEC.md`.

Una vez que la especificación esté completa, inicie una sesión nueva para ejecutarla. La nueva sesión tiene contexto limpio enfocado completamente en la implementación, y tiene una especificación escrita para hacer referencia.

Gestione su sesión

Las conversaciones son persistentes y reversibles. ¡Úselo a su favor!

Corrija el curso temprano y a menudo

Tip:

Corrija a Claude tan pronto como note que se desvía del camino.

Los mejores resultados provienen de bucles de retroalimentación ajustados. Aunque Claude ocasionalmente resuelve problemas perfectamente en el primer intento, corregirlo rápidamente generalmente produce mejores soluciones más rápido.

- **Esc** : detener a Claude a mitad de acción con la tecla **Esc** . El contexto se preserva, para que pueda redirigir.
- **Esc + Esc** o **/rewind** : presione **Esc** dos veces o ejecute **/rewind** para abrir el menú de rebobinado y restaurar la conversación anterior y el estado del código, o resumir desde un mensaje seleccionado.
- **"Undo that"** : haga que Claude revierta sus cambios.
- **/clear** : restablecer contexto entre tareas no relacionadas. Las sesiones largas con contexto irrelevante pueden reducir el rendimiento.

Si ha corregido a Claude más de dos veces en el mismo problema en una sesión, el contexto está saturado de enfoques fallidos. Ejecute **/clear** e inicie de nuevo con una indicación más específica que incorpore lo que aprendió. Una sesión limpia con una indicación mejor casi siempre supera una sesión larga con correcciones acumuladas.

Gestione el contexto agresivamente

Tip:

Ejecute **/clear** entre tareas no relacionadas para restablecer el contexto.

Claude Code compacta automáticamente el historial de conversación cuando se acerca a los límites de contexto, lo que preserva código importante y decisiones mientras libera espacio.

Durante sesiones largas, la ventana de contexto de Claude puede llenarse con conversación irrelevante, contenidos de archivo y comandos. Esto puede reducir el rendimiento y a veces distraer a Claude.

- Use `/clear` frecuentemente entre tareas para restablecer completamente la ventana de contexto
- Cuando se activa la compactación automática, Claude resume lo que más importa, incluyendo patrones de código, estados de archivo y decisiones clave
- Para más control, ejecute `/compact <instructions>`, como `/compact Focus on the API changes`
- Para compactar solo parte de la conversación, use `Esc + Esc` o `/rewind`, seleccione un punto de control de mensaje y elija **Summarize from here**. Esto condensa mensajes desde ese punto hacia adelante mientras mantiene el contexto anterior intacto.
- Personalice el comportamiento de compactación en CLAUDE.md con instrucciones como `"When compacting, always preserve the full list of modified files and any test commands"` para asegurar que el contexto crítico sobreviva a la resumición
- Para preguntas rápidas que no necesitan permanecer en contexto, use `/btw`. La respuesta aparece en una superposición descartable y nunca entra en el historial de conversación, para que pueda verificar un detalle sin aumentar el contexto.

Use subagents para investigación

Tip:

Delegue investigación con `"use subagents to investigate X"`. Exploran en un contexto separado, manteniendo su conversación principal limpia para la implementación.

Dado que el contexto es su restricción fundamental, los subagents son una de las herramientas más poderosas disponibles. Cuando Claude investiga un código base, lee muchos archivos, todos los cuales consumen su contexto. Los subagents se ejecutan en ventanas de contexto separadas e informan resúmenes:

```
Use subagents to investigate how our authentication system handles token
refresh, and whether we have any existing OAuth utilities I should reuse.
```

El subagent explora el código base, lee archivos relevantes e informa hallazgos, todo sin saturar su conversación principal.

También puede usar subagents para verificación después de que Claude implemente algo:

```
use a subagent to review this code for edge cases
```

Rebobine con puntos de control

Tip:

Cada acción que Claude realiza crea un punto de control. Puede restaurar conversación, código o ambos a cualquier punto de control anterior.

Claude automáticamente crea puntos de control antes de cambios. Presione Escape dos veces o ejecute `/rewind` para abrir el menú de rebobinado. Puede restaurar solo conversación, restaurar solo código, restaurar ambos o resumir desde un mensaje seleccionado. Consulte [Checkpointing](#) para detalles.

En lugar de planificar cuidadosamente cada movimiento, puede decirle a Claude que intente algo arriesgado. Si no funciona, rebobine e intente un enfoque diferente. Los puntos de control persisten entre sesiones, para que pueda cerrar su terminal y aún rebobinar más tarde.

Warning:

Los puntos de control solo rastrean cambios realizados *por Claude*, no procesos externos. Esto no es un reemplazo para git.

Reanudar conversaciones

Tip:

Ejecute `claude --continue` para continuar donde lo dejó, o `--resume` para elegir entre sesiones recientes.

Claude Code guarda conversaciones localmente. Cuando una tarea abarca múltiples sesiones, no tiene que re-explicar el contexto:


```

claude --continue      # Resume the most recent conversation
claude --resume        # Select from recent conversations

```

Use `/rename` para dar a las sesiones nombres descriptivos como `"oauth-migration"` o `"debugging-memory-leak"` para que pueda encontrarlas más tarde. Trate las sesiones como ramas: diferentes flujos de trabajo pueden tener contextos separados y persistentes.

Automatica y escale

Una vez que sea efectivo con un Claude, multiplique su salida con sesiones paralelas, modo no interactivo y patrones de abanico.

Todo hasta ahora asume un humano, un Claude y una conversación. Pero Claude Code escala horizontalmente. Las técnicas en esta sección muestran cómo puede hacer más.

Ejecutar modo no interactivo

Tip:

Use `claude -p "prompt"` en CI, hooks previos a la confirmación o scripts. Agregue `--output-format stream-json` para salida JSON de transmisión.

Con `claude -p "your prompt"`, puede ejecutar Claude de forma no interactiva, sin una sesión. El modo no interactivo es cómo integra Claude en canalizaciones de CI, hooks previos a la confirmación o cualquier flujo de trabajo automatizado. Los formatos de salida le permiten analizar resultados mediante programación: texto sin formato, JSON o JSON de transmisión.

```

## One-off queries
claude -p "Explain what this project does"

## Structured output for scripts
claude -p "List all API endpoints" --output-format json

## Streaming for real-time processing
claude -p "Analyze this log file" --output-format stream-json

```

Ejecutar múltiples sesiones de Claude

Tip:

Ejecute múltiples sesiones de Claude en paralelo para acelerar el desarrollo, ejecutar experimentos aislados o iniciar flujos de trabajo complejos.

Hay tres formas principales de ejecutar sesiones paralelas:

- **Aplicación de escritorio Claude Code:** Gestione múltiples sesiones locales visualmente. Cada sesión obtiene su propio worktree aislado.
- **Claude Code en la web:** Ejecutar en la infraestructura en la nube segura de Anthropic en máquinas virtuales aisladas.
- **Equipos de agentes:** Coordinación automatizada de múltiples sesiones con tareas compartidas, mensajería y un líder de equipo.

Más allá de paralelizar el trabajo, múltiples sesiones habilitan flujos de trabajo enfocados en la calidad. Un contexto nuevo mejora la revisión de código ya que Claude no estará sesgado hacia el código que acaba de escribir.

Por ejemplo, use un patrón Escritor/Revisor:

Sesión A (Escritor)	Sesión B (Revisor)
Implement a rate limiter for our API endpoints	
	Review the rate limiter implementation in @src/middleware/rateLimiter.ts. Look for edge cases, race conditions, and consistency with our existing middleware patterns.
Here's the review feedback: [Session B output]. Address these issues.	

Puede hacer algo similar con pruebas: haga que un Claude escriba pruebas, luego otro escriba código para pasarlas.

Abanico a través de archivos

Tip:

Recorra tareas llamando a `claude -p` para cada una. Use `--allowedTools` para permisos de alcance para operaciones por lotes.

Para migraciones o análisis grandes, puede distribuir trabajo entre muchas invocaciones paralelas de Claude:

Step 1: Generar una lista de tareas

Haga que Claude enumere todos los archivos que necesitan migración (por ejemplo, `list all 2,000 Python files that need migrating`)

Step 2: Escribir un script para recorrer la lista

```
for file in $(cat files.txt); do
  claude -p "Migrate $file from React to Vue. Return OK or FAIL." \
    --allowedTools "Edit,Bash(git commit *)"
done
```

Step 3: Probar en algunos archivos, luego ejecutar a escala

Refine su indicación basada en lo que sale mal con los primeros 2-3 archivos, luego ejecute en el conjunto completo. La bandera `--allowedTools` restringe lo que Claude puede hacer, lo que importa cuando está ejecutando desatendido.

También puede integrar Claude en canalizaciones de datos/procesamiento existentes:

```
claude -p "<your prompt>" --output-format json | your_command
```

Use `--verbose` para depuración durante el desarrollo, y apáguelo en producción.

Evite patrones de falla comunes

Estos son errores comunes. Reconocerlos temprano ahorra tiempo:

- **La sesión de todo incluido.** Comienza con una tarea, luego pregunta a Claude algo no relacionado, luego vuelve a la primera tarea. El contexto está lleno de información irrelevante. > **Solución:** `/clear` entre tareas no relacionadas.

- **Corrección una y otra vez.** Claude hace algo mal, lo corrige, sigue siendo incorrecto, lo corrige de nuevo. El contexto está contaminado con enfoques fallidos. > **Solución:** Después de dos correcciones fallidas, `/clear` y escriba una indicación inicial mejor incorporando lo que aprendió.
 - **El CLAUDE.md sobre especificado.** Si su CLAUDE.md es demasiado largo, Claude ignora la mitad porque las reglas importantes se pierden en el ruido. > **Solución:** Elimine sin piedad. Si Claude ya hace algo correctamente sin la instrucción, elimínelo o conviértalo en un hook.
 - **La brecha de confianza-luego-verificación.** Claude produce una implementación que se ve plausible pero no maneja casos extremos. > **Solución:** Siempre proporcione verificación (pruebas, scripts, capturas de pantalla). Si no puede verificarlo, no lo envíe.
 - **La exploración infinita.** Pide a Claude que “investigue” algo sin delimitarlo. Claude lee cientos de archivos, llenando el contexto. > **Solución:** Delimite investigaciones estrechamente o use subagents para que la exploración no consuma su contexto principal.
-

Desarrolle su intuición

Los patrones en esta guía no están grabados en piedra. Son puntos de partida que funcionan bien en general, pero podrían no ser óptimos para cada situación.

A veces *debería* dejar que el contexto se acumule porque está profundo en un problema complejo y el historial es valioso. A veces debería omitir la planificación y dejar que Claude lo descubra porque la tarea es exploratoria. A veces una indicación vaga es exactamente lo correcto porque desea ver cómo Claude interpreta el problema antes de limitarlo.

Preste atención a lo que funciona. Cuando Claude produce una salida excelente, note lo que hizo: la estructura de la indicación, el contexto que proporcionó, el modo en que estaba. Cuando Claude lucha, pregúntese por qué. ¿Fue el contexto demasiado ruidoso? ¿La indicación demasiado vaga? ¿La tarea demasiado grande para un pase?

Con el tiempo, desarrollará intuición que ninguna guía puede capturar. Sabrá cuándo ser específico y cuándo ser abierto, cuándo planificar y cuándo explorar, cuándo limpiar contexto y cuándo dejarlo acumular.

Recursos relacionados

- [Cómo funciona Claude Code](#): el bucle agencial, herramientas y gestión de contexto
- [Extender Claude Code](#): skills, hooks, MCP, subagents y plugins

- [Flujos de trabajo comunes](#): recetas paso a paso para depuración, pruebas, PRs y más
- [CLAUDE.md](#): almacenar convenciones de proyecto y contexto persistente

Flujos de trabajo comunes

Guías paso a paso para explorar bases de código, corregir errores, refactorizar, probar y otras tareas cotidianas con Claude Code.

Esta página cubre flujos de trabajo prácticos para el desarrollo cotidiano: explorar código desconocido, depuración, refactorización, escritura de pruebas, creación de solicitudes de extracción y gestión de sesiones. Cada sección incluye ejemplos de indicaciones que puede adaptar a sus propios proyectos. Para patrones y consejos de nivel superior, consulte [Mejores prácticas](#).

Comprender nuevas bases de código

Obtener una descripción general rápida de la base de código

Supongamos que acaba de unirse a un nuevo proyecto y necesita comprender su estructura rápidamente.

Step 1: Navegue al directorio raíz del proyecto

```
cd /path/to/project
```

Step 2: Inicie Claude Code

```
claude
```

Step 3: Solicite una descripción general de alto nivel

```
dame una descripción general de esta base de código
```

Step 4: Profundice en componentes específicos

```
explica los patrones de arquitectura principales utilizados aquí
```

¿cuáles son los modelos de datos clave?

¿cómo se maneja la autenticación?

Tip:

Consejos:

- Comience con preguntas amplias, luego reduzca a áreas específicas
- Pregunte sobre convenciones de codificación y patrones utilizados en el proyecto
- Solicite un glosario de términos específicos del proyecto

Encontrar código relevante

Supongamos que necesita localizar código relacionado con una característica o funcionalidad específica.

Step 1: Pida a Claude que encuentre archivos relevantes

encuentra los archivos que manejan la autenticación de usuarios

Step 2: Obtenga contexto sobre cómo interactúan los componentes

¿cómo funcionan juntos estos archivos de autenticación?

Step 3: Comprenda el flujo de ejecución

rastrear el proceso de inicio de sesión de front-end a base de datos

Tip:

Consejos:

- Sea específico sobre lo que está buscando
- Utilice el lenguaje del dominio del proyecto
- Instale un [complemento de inteligencia de código](#) para su lenguaje para dar a Claude una navegación precisa de “ir a definición” y “buscar referencias”

Corregir errores de manera eficiente

Supongamos que ha encontrado un mensaje de error y necesita encontrar y corregir su origen.

Step 1: Comparta el error con Claude

```
estoy viendo un error cuando ejecuto npm test
```

Step 2: Solicite recomendaciones de corrección

```
sugiere algunas formas de corregir el @ts-ignore en user.ts
```

Step 3: Aplique la corrección

```
actualiza user.ts para agregar la verificación nula que sugeriste
```

Tip:

Consejos:

- Dígale a Claude el comando para reproducir el problema y obtener un seguimiento de pila
- Mencione cualquier paso para reproducir el error
- Hágle saber a Claude si el error es intermitente o consistente

Refactorizar código

Supongamos que necesita actualizar código antiguo para utilizar patrones y prácticas modernas.

Step 1: Identifique código heredado para refactorización

```
encuentra el uso de API obsoleta en nuestra base de código
```

Step 2: Obtenga recomendaciones de refactorización

sugiere cómo refactorizar `utils.js` para usar características modernas de JavaScript

Step 3: Aplique los cambios de manera segura

refactoriza `utils.js` para usar características ES2024 manteniendo el mismo comportamiento

Step 4: Verifique la refactorización

ejecuta pruebas para el código refactorizado

Tip:

Consejos:

- Pida a Claude que explique los beneficios del enfoque moderno
- Solicite que los cambios mantengan la compatibilidad hacia atrás cuando sea necesario
- Realice la refactorización en incrementos pequeños y comprobables

Usar subagentes especializados

Supongamos que desea utilizar subagentes de IA especializados para manejar tareas específicas de manera más efectiva.

Step 1: Ver subagentes disponibles

`/agents`

Esto muestra todos los subagentes disponibles y le permite crear otros nuevos.

Step 2: Usar subagentes automáticamente

Claude Code delega automáticamente tareas apropiadas a subagentes especializados:

revisa mis cambios de código recientes para problemas de seguridad

ejecuta todas las pruebas y corrige cualquier fallo

Step 3: Solicitar explícitamente subagentes específicos

usa el subagente `code-reviewer` para verificar el módulo de autenticación

haz que el subagente `debugger` investigue por qué los usuarios no pueden iniciar sesión

Step 4: Crear subagentes personalizados para su flujo de trabajo

`/agents`

Luego seleccione “Crear nuevo subagente” y siga las indicaciones para definir:

- Un identificador único que describa el propósito del subagente (por ejemplo, `code-reviewer`, `api-designer`).
- Cuándo Claude debe usar este agente
- Qué herramientas puede acceder
- Un indicador del sistema que describa el rol y comportamiento del agente

Tip:

Consejos:

- Cree subagentes específicos del proyecto en `.claude/agents/` para compartir en equipo
- Utilice campos `description` descriptivos para habilitar la delegación automática
- Limite el acceso a herramientas a lo que cada subagente realmente necesita
- Consulte la [documentación de subagentes](#) para ejemplos detallados

Usar Plan Mode para análisis seguro de código

Plan Mode instruye a Claude para crear un plan analizando la base de código con operaciones de solo lectura, perfecto para explorar bases de código, planificar cambios complejos o revisar código de manera segura. En Plan Mode, Claude utiliza `AskUserQuestion` para recopilar requisitos y aclarar sus objetivos antes de proponer un plan.

Cuándo usar Plan Mode

- **Implementación de múltiples pasos:** Cuando su característica requiere hacer ediciones en muchos archivos
- **Exploración de código:** Cuando desea investigar la base de código a fondo antes de cambiar nada
- **Desarrollo interactivo:** Cuando desea iterar en la dirección con Claude

Cómo usar Plan Mode

Activar Plan Mode durante una sesión

Puede cambiar a Plan Mode durante una sesión usando **Shift+Tab** para ciclar a través de modos de permiso.

Si está en Normal Mode, **Shift+Tab** primero cambia a Auto-Accept Mode, indicado por `▶▶ accept edits on` en la parte inferior de la terminal. Un **Shift+Tab** posterior cambiará a Plan Mode, indicado por `|| plan mode on`.

Iniciar una nueva sesión en Plan Mode

Para iniciar una nueva sesión en Plan Mode, use la bandera `--permission-mode plan`:

```
claude --permission-mode plan
```

Ejecutar consultas “sin interfaz” en Plan Mode

También puede ejecutar una consulta en Plan Mode directamente con `-p` (es decir, en “modo sin interfaz”):

```
claude --permission-mode plan -p "Analiza el sistema de autenticación y sugiere mejoras"
```

Ejemplo: Planificar una refactorización compleja

```
claude --permission-mode plan
```

Necesito refactorizar nuestro sistema de autenticación para usar OAuth2. Crea un plan de migración detallado.

Claude analiza la implementación actual y crea un plan integral. Refine con seguimientos:

¿Qué hay sobre la compatibilidad hacia atrás?

¿Cómo deberíamos manejar la migración de la base de datos?

Tip:

Presione **Ctrl+G** para abrir el plan en su editor de texto predeterminado, donde puede editarlo directamente antes de que Claude continúe.

Configurar Plan Mode como predeterminado

```
// .claude/settings.json
{
  "permissions": {
    "defaultMode": "plan"
  }
}
```

Consulte la [documentación de configuración](#) para más opciones de configuración.

Trabajar con pruebas

Supongamos que necesita agregar pruebas para código no cubierto.

Step 1: Identifique código no probado

encuentra funciones en `NotificationsService.swift` que no están cubiertas por pruebas

Step 2: Genere andamiaje de prueba

agrega pruebas para el servicio de notificaciones

Step 3: Agregue casos de prueba significativos

agrega casos de prueba para condiciones de borde en el servicio de notificaciones

Step 4: Ejecute y verifique las pruebas

ejecuta las nuevas pruebas y corrige cualquier fallo

Claude puede generar pruebas que sigan los patrones y convenciones existentes de su proyecto. Al solicitar pruebas, sea específico sobre qué comportamiento desea verificar. Claude examina sus archivos de prueba existentes para coincidir con el estilo, marcos y patrones de afirmación ya en uso.

Para una cobertura integral, pida a Claude que identifique casos extremos que podría haber perdido. Claude puede analizar sus rutas de código y sugerir pruebas para condiciones de error, valores límite e entradas inesperadas que son fáciles de pasar por alto.

Crear solicitudes de extracción

Puede crear solicitudes de extracción pidiendo a Claude directamente (“crear una pr para mis cambios”), o guiar a Claude a través de ella paso a paso:

Step 1: Resuma sus cambios

resume los cambios que he hecho en el módulo de autenticación

Step 2: Genere una solicitud de extracción

crear una pr

Step 3: Revise y refine

mejora la descripción de PR con más contexto sobre las mejoras de seguridad

Cuando crea una PR usando `gh pr create`, la sesión se vincula automáticamente a esa PR. Puede reanudarla más tarde con `claude --from-pr <number>`.

Tip:

Revise la PR generada por Claude antes de enviarla y pida a Claude que destaque los riesgos potenciales o consideraciones.

Manejar documentación

Supongamos que necesita agregar o actualizar documentación para su código.

Step 1: Identifique código sin documentar

encuentra funciones sin comentarios JSDoc adecuados en el módulo de autenticación

Step 2: Genere documentación

agrega comentarios JSDoc a las funciones sin documentar en auth.js

Step 3: Revise y mejore

mejora la documentación generada con más contexto y ejemplos

Step 4: Verifique la documentación

verifica si la documentación sigue los estándares de nuestro proyecto

Tip:

Consejos:

- Especifique el estilo de documentación que desea (JSDoc, docstrings, etc.)
- Solicite ejemplos en la documentación
- Solicite documentación para API públicas, interfaces y lógica compleja

Trabajar con imágenes

Supongamos que necesita trabajar con imágenes en su base de código y desea la ayuda de Claude para analizar el contenido de la imagen.

Step 1: Agregue una imagen a la conversación

Puede usar cualquiera de estos métodos:

1. Arrastre y suelte una imagen en la ventana de Claude Code
2. Copie una imagen y péguela en la CLI con ctrl+v (No use cmd+v)
3. Proporcione una ruta de imagen a Claude. Por ejemplo, “Analiza esta imagen: /path/to/your/image.png”

Step 2: Pida a Claude que analice la imagen

¿Qué muestra esta imagen?

Describe los elementos de la interfaz de usuario en esta captura de pantalla

¿Hay algún elemento problemático en este diagrama?

Step 3: Usar imágenes para contexto

Aquí hay una captura de pantalla del error. ¿Qué lo está causando?

Este es nuestro esquema de base de datos actual. ¿Cómo deberíamos modificarlo para la nueva característica?

Step 4: Obtenga sugerencias de código del contenido visual

Generar CSS para coincidir con este mockup de diseño

¿Qué estructura HTML recrearía este componente?

Tip:

Consejos:

- Utilice imágenes cuando las descripciones de texto serían poco claras o engorrosas
- Incluya capturas de pantalla de errores, diseños de interfaz de usuario o diagramas para un mejor contexto
- Puede trabajar con múltiples imágenes en una conversación
- El análisis de imágenes funciona con diagramas, capturas de pantalla, mockups y más
- Cuando Claude hace referencia a imágenes (por ejemplo, `[Image #1]`), `Cmd+Click` (Mac) o `Ctrl+Click` (Windows/Linux) el enlace para abrir la imagen en su visor predeterminado

Archivos y directorios de referencia

Use `@` para incluir rápidamente archivos o directorios sin esperar a que Claude los lea.

Step 1: Haga referencia a un archivo único

Explica la lógica en `@src/utils/auth.js`

Esto incluye el contenido completo del archivo en la conversación.

Step 2: Haga referencia a un directorio

¿Cuál es la estructura de `@src/components`?

Esto proporciona un listado de directorio con información de archivo.

Step 3: Haga referencia a recursos MCP

Muéstrame los datos de `@github:repos/owner/repo/issues`

Esto obtiene datos de servidores MCP conectados usando el formato `@server:resource`. Consulte [recursos MCP](#) para más detalles.

Tip:

Consejos:

- Las rutas de archivo pueden ser relativas o absolutas
- Las referencias de archivo `@` agregan `CLAUDE.md` en el directorio del archivo y directorios principales al contexto
- Las referencias de directorio muestran listados de archivos, no contenidos
- Puede hacer referencia a múltiples archivos en un solo mensaje (por ejemplo, “`@file1.js` y `@file2.js`”)

Usar pensamiento extendido (thinking mode)

El [pensamiento extendido](#) está habilitado de forma predeterminada, dando a Claude espacio para razonar a través de problemas complejos paso a paso antes de responder. Este razonamiento es visible en modo detallado, que puede alternar con `Ctrl+0`.

Además, Opus 4.6 introduce razonamiento adaptativo: en lugar de un presupuesto de token de pensamiento fijo, el modelo asigna dinámicamente el pensamiento basado en su configuración de [nivel de esfuerzo](#). El pensamiento extendido y el razonamiento adaptativo funcionan juntos para darle control sobre qué tan profundamente Claude razona antes de responder.

El pensamiento extendido es particularmente valioso para decisiones arquitectónicas complejas, errores desafiantes, planificación de implementación de múltiples pasos y evaluación de compensaciones entre diferentes enfoques.

Note:

Frases como “pensar”, “pensar duro” y “pensar más” se interpretan como instrucciones de indicación regulares y no asignan tokens de pensamiento.

Configurar thinking mode

El pensamiento está habilitado de forma predeterminada, pero puede ajustarlo o deshabilitarlo.

Alcance	Cómo configurar	Detalles
Nivel de esfuerzo	Ajuste en <code>/model</code> o establezca <code>CLAUDE_CODE_EFFORT_LEVEL</code>	Control de profundidad de pensamiento para Opus 4.6 y Sonnet 4.6: bajo, medio, alto. Consulte Ajustar nivel de esfuerzo
Palabra clave <code>ultrathink</code>	Incluya “ultrathink” en cualquier lugar de su indicación	Establece el esfuerzo en alto para ese turno en Opus 4.6 y Sonnet 4.6. Útil para tareas únicas que requieren razonamiento profundo sin cambiar permanentemente su configuración de esfuerzo
Atajo de alternancia	Presione <code>Option+T</code> (macOS) o <code>Alt+T</code> (Windows/Linux)	Alterne el pensamiento activado/desactivado para la sesión actual (todos los modelos). Puede requerir configuración de terminal para habilitar atajos de tecla de opción
Predeterminado global	Use <code>/config</code> para alternar thinking mode	Establece su predeterminado en todos los proyectos (todos los modelos). Guardado como <code>alwaysThinkingEnabled</code> en <code>~/.claude/settings.json</code>
Presupuesto de token límite	Establezca la variable de entorno <code>MAX_THINKING_TOKENS</code>	Limite el presupuesto de pensamiento a un número específico de tokens (ignorado en Opus 4.6 a menos que se establezca en 0). Ejemplo: <code>export MAX_THINKING_TOKENS=10000</code>

Para ver el proceso de pensamiento de Claude, presione `Ctrl+0` para alternar el modo detallado y ver el razonamiento interno mostrado como texto gris en cursiva.

Cómo funciona el pensamiento extendido

El pensamiento extendido controla cuánto razonamiento interno realiza Claude antes de responder. Más pensamiento proporciona más espacio para explorar soluciones, analizar casos extremos y autocorregir errores.

Con Opus 4.6, el pensamiento utiliza razonamiento adaptativo: el modelo asigna dinámicamente tokens de pensamiento basados en el [nivel de esfuerzo](#) que selecciona (bajo, medio, alto). Esta es la forma recomendada de ajustar la compensación entre velocidad y profundidad de razonamiento.

Con otros modelos, el pensamiento utiliza un presupuesto fijo de hasta 31.999 tokens de su presupuesto de salida. Puede limitar esto con la variable de entorno

`MAX_THINKING_TOKENS`, o deshabilitar el pensamiento completamente a través de `/config` o la alternancia `Option+T` / `Alt+T`.

`MAX_THINKING_TOKENS` se ignora en Opus 4.6 y Sonnet 4.6, ya que el razonamiento adaptativo controla la profundidad del pensamiento en su lugar. La única excepción: establecer `MAX_THINKING_TOKENS=0` aún deshabilita el pensamiento completamente en cualquier modelo. Para deshabilitar el pensamiento adaptativo y revertir al presupuesto de pensamiento fijo, establezca `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Consulte [variables de entorno](#).

Warning:

Se le cobra por todos los tokens de pensamiento utilizados, aunque los modelos Claude 4 muestren pensamiento resumido

Reanudar conversaciones anteriores

Cuando inicia Claude Code, puede reanudar una sesión anterior:

- `claude --continue` continúa la conversación más reciente en el directorio actual
- `claude --resume` abre un selector de conversación o reanuda por nombre
- `claude --from-pr 123` reanuda sesiones vinculadas a una solicitud de extracción específica

Desde dentro de una sesión activa, use `/resume` para cambiar a una conversación diferente.

Las sesiones se almacenan por directorio de proyecto. El selector `/resume` muestra sesiones del mismo repositorio de git, incluidos worktrees.

Nombrar sus sesiones

Dé a las sesiones nombres descriptivos para encontrarlas más tarde. Esta es una mejor práctica cuando se trabaja en múltiples tareas o características.

Step 1: Nombre la sesión actual

Use `/rename` durante una sesión para darle un nombre memorable:

```
/rename auth-refactor
```

También puede renombrar cualquier sesión desde el selector: ejecute `/resume`, navegue a una sesión y presione `R`.

Step 2: Reanude por nombre más tarde

Desde la línea de comandos:

```
claude --resume auth-refactor
```

O desde dentro de una sesión activa:

```
/resume auth-refactor
```

Usar el selector de sesión

El comando `/resume` (o `claude --resume` sin argumentos) abre un selector de sesión interactivo con estas características:

Atajos de teclado en el selector:

Atajo	Acción
<code>↑ / ↓</code>	Navegue entre sesiones
<code>→ / ←</code>	Expandir o contraer sesiones agrupadas
<code>Enter</code>	Seleccione y reanude la sesión resaltada
<code>P</code>	Vista previa del contenido de la sesión
<code>R</code>	Renombre la sesión resaltada
<code>/</code>	Busque para filtrar sesiones

Atajo	Acción
A	Alterne entre directorio actual y todos los proyectos
B	Filtrar a sesiones de su rama de git actual
Esc	Salga del selector o modo de búsqueda

Organización de sesiones:

El selector muestra sesiones con metadatos útiles:

- Nombre de sesión o indicación inicial
- Tiempo transcurrido desde la última actividad
- Recuento de mensajes
- Rama de Git (si corresponde)

Las sesiones bifurcadas (creadas con `/rewind` o `--fork-session`) se agrupan bajo su sesión raíz, lo que facilita encontrar conversaciones relacionadas.

Tip:

Consejos:

- **Nombre sesiones temprano:** Use `/rename` cuando comience a trabajar en una tarea distinta, es mucho más fácil encontrar “payment-integration” que “explica esta función” más tarde
- Use `--continue` para acceso rápido a su conversación más reciente en el directorio actual
- Use `--resume session-name` cuando sepa qué sesión necesita
- Use `--resume` (sin nombre) cuando necesite examinar y seleccionar
- Para scripts, use `claude --continue --print "prompt"` para reanudar en modo no interactivo
- Presione **P** en el selector para obtener una vista previa de una sesión antes de reanudarla
- La conversación reanudada comienza con el mismo modelo y configuración que el original

Cómo funciona:

1. **Almacenamiento de conversación:** Todas las conversaciones se guardan automáticamente localmente con su historial de mensajes completo
2. **Deserialización de mensajes:** Al reanudar, se restaura el historial de mensajes completo para mantener el contexto
3. **Estado de herramienta:** El uso de herramientas y los resultados de la conversación anterior se conservan

4. **Restauración de contexto:** La conversación se reanuda con todo el contexto anterior intacto

Ejecutar sesiones paralelas de Claude Code con Git worktrees

Cuando trabaja en múltiples tareas a la vez, necesita que cada sesión de Claude tenga su propia copia de la base de código para que los cambios no choquen. Los worktrees de Git resuelven esto creando directorios de trabajo separados que cada uno tiene sus propios archivos y rama, mientras comparten el mismo historial de repositorio y conexiones remotas. Esto significa que puede tener a Claude trabajando en una característica en un worktree mientras corrige un error en otro, sin que ninguna sesión interfiera con la otra.

Use la bandera `--worktree` (`-w`) para crear un worktree aislado e iniciar Claude en él. El valor que pasa se convierte en el nombre del directorio worktree y el nombre de la rama:

```
## Inicie Claude en un worktree llamado "feature-auth"
## Crea .claude/worktrees/feature-auth/ con una nueva rama
claude --worktree feature-auth

## Inicie otra sesión en un worktree separado
claude --worktree bugfix-123
```

Si omite el nombre, Claude genera uno automáticamente:

```
## Auto-genera un nombre como "bright-running-fox"
claude --worktree
```

Los worktrees se crean en `<repo>/.claude/worktrees/<name>` y se ramifican desde la rama remota predeterminada. La rama worktree se nombra `worktree-<name>`.

También puede pedir a Claude que “trabaje en un worktree” o “inicie un worktree” durante una sesión, y lo creará automáticamente.

Worktrees de subagentes

Los subagentes también pueden usar aislamiento de worktree para trabajar en paralelo sin conflictos. Pida a Claude que “use worktrees para sus agentes” o configúrelo en un [subagente personalizado](#) agregando `isolation: worktree` al frontmatter del agente. Cada subagente obtiene su propio worktree que se limpia automáticamente cuando el subagente termina sin cambios.

Limpieza de worktree

Cuando sale de una sesión de worktree, Claude maneja la limpieza según si realizó cambios:

- **Sin cambios:** el worktree y su rama se eliminan automáticamente
- **Cambios o commits existen:** Claude le solicita que mantenga o elimine el worktree. Mantener preserva el directorio y la rama para que pueda regresar más tarde. Eliminar elimina el directorio worktree y su rama, descartando todos los cambios sin confirmar y commits

Para limpiar worktrees fuera de una sesión de Claude, use [gestión manual de worktree](#).

Tip:

Agregue `.claude/worktrees/` a su `.gitignore` para evitar que el contenido del worktree aparezca como archivos sin seguimiento en su repositorio principal.

Gestionar worktrees manualmente

Para más control sobre la ubicación del worktree y la configuración de la rama, cree worktrees con Git directamente. Esto es útil cuando necesita verificar una rama existente específica o colocar el worktree fuera del repositorio.

```
## Crear un worktree con una nueva rama
git worktree add ../project-feature-a -b feature-a

## Crear un worktree con una rama existente
git worktree add ../project-bugfix bugfix-123

## Inicie Claude en el worktree
cd ../project-feature-a && claude

## Limpiar cuando termine
git worktree list
git worktree remove ../project-feature-a
```

Obtenga más información en la [documentación oficial de Git worktree](#).

Tip:

Recuerde inicializar su entorno de desarrollo en cada nuevo worktree de acuerdo con la configuración de su proyecto. Dependiendo de su pila, esto podría incluir ejecutar instalación de dependencias (`npm install` , `yarn`), configurar entornos virtuales o seguir el proceso de configuración estándar de su proyecto.

Control de versiones no git

El aislamiento de worktree funciona con git de forma predeterminada. Para otros sistemas de control de versiones como SVN, Perforce o Mercurial, configure [hooks WorktreeCreate y WorktreeRemove](#) para proporcionar lógica personalizada de creación y limpieza de worktree. Cuando se configura, estos hooks reemplazan el comportamiento predeterminado de git cuando usa `--worktree` .

Para la coordinación automatizada de sesiones paralelas con tareas compartidas y mensajería, consulte [equipos de agentes](#).

Reciba notificaciones cuando Claude necesite su atención

Cuando inicia una tarea de larga duración y cambia a otra ventana, puede configurar notificaciones de escritorio para saber cuándo Claude termina o necesita su entrada. Esto utiliza el evento de hook `Notification hook event`, que se activa cada vez que Claude está esperando permiso, inactivo y listo para una nueva indicación, o completando autenticación.

Step 1: Abra el menú de hooks

Escriba `/hooks` y seleccione `Notification` de la lista de eventos.

Step 2: Configure el matcher

Seleccione `+ Match all (no filter)` para activarse en todos los tipos de notificación. Para notificar solo para eventos específicos, seleccione `+ Add new matcher...` e ingrese uno de estos valores:

Matcher	Se activa cuando
<code>permission_prompt</code>	Claude necesita que apruebe un uso de herramienta
<code>idle_prompt</code>	Claude está hecho y esperando su próxima indicación
<code>auth_success</code>	La autenticación se completa
<code>elicitation_dialog</code>	Claude le está haciendo una pregunta

Step 3: Agregue su comando de notificación

Seleccione `+ Add new hook...` e ingrese el comando para su SO:

macOS

Utiliza `osascript` para activar una notificación nativa de macOS a través de AppleScript:

```
osascript -e 'display notification "Claude Code needs your attention" with title "Claude Code"'
```

Linux

Utiliza `notify-send`, que viene preinstalado en la mayoría de escritorios Linux con un demonio de notificación:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

Utiliza PowerShell para mostrar un cuadro de mensaje nativo a través de .NET Windows Forms:

```
powershell.exe -Command  
"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');  
[System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention',  
'Claude Code')"
```

Step 4: Guardar en configuración de usuario

Seleccione **User settings** para aplicar la notificación en todos sus proyectos.

Para el tutorial completo con ejemplos de configuración JSON, consulte [Automatizar flujos de trabajo con hooks](#). Para el esquema de evento completo y tipos de notificación, consulte la [referencia de notificación](#).

Usar Claude como una utilidad de estilo unix

Agregue Claude a su proceso de verificación

Supongamos que desea usar Claude Code como un linter o revisor de código.

Agregue Claude a su script de compilación:

```
// package.json
{
  ...
  "scripts": {
    ...
    "lint:claude": "claude -p 'you are a linter. please look at the changes
vs. main and report any issues related to typos. report the filename and line
number on one line, and a description of the issue on the second line. do not
return any other text.'"
  }
}
```

Tip:

Consejos:

- Utilice Claude para revisión de código automatizada en su canalización CI/CD
- Personalice la indicación para verificar problemas específicos relevantes para su proyecto
- Considere crear múltiples scripts para diferentes tipos de verificación

Canalizar entrada, canalizar salida

Supongamos que desea canalizar datos a Claude y obtener datos en un formato estructurado.

Canalizar datos a través de Claude:

```
cat build-error.txt | claude -p 'concisely explain the root cause of this build
error' > output.txt
```

Tip:

Consejos:

- Utilice tuberías para integrar Claude en scripts de shell existentes
- Combine con otras herramientas Unix para flujos de trabajo poderosos
- Considere usar `-output-format` para salida estructurada

Controlar el formato de salida

Supongamos que necesita la salida de Claude en un formato específico, especialmente cuando integra Claude Code en scripts u otras herramientas.

Step 1: Usar formato de texto (predeterminado)

```
cat data.txt | claude -p 'summarize this data' --output-format text > summary.txt
```

Esto genera solo la respuesta de texto sin formato de Claude (comportamiento predeterminado).

Step 2: Usar formato JSON

```
cat code.py | claude -p 'analyze this code for bugs' --output-format json > analysis.json
```

Esto genera una matriz JSON de mensajes con metadatos incluyendo costo y duración.

Step 3: Usar formato JSON de transmisión

```
cat log.txt | claude -p 'parse this log file for errors' --output-format stream-json
```

Esto genera una serie de objetos JSON en tiempo real mientras Claude procesa la solicitud. Cada mensaje es un objeto JSON válido, pero la salida completa no es JSON válido si se concatena.

Tip:

Consejos:

- Utilice `--output-format text` para integraciones simples donde solo necesita la respuesta de Claude
- Utilice `--output-format json` cuando necesite el registro de conversación completo
- Utilice `--output-format stream-json` para salida en tiempo real de cada turno de conversación

Pregunte a Claude sobre sus capacidades

Claude tiene acceso integrado a su documentación y puede responder preguntas sobre sus propias características y limitaciones.

Preguntas de ejemplo

¿puede Claude Code crear solicitudes de extracción?

¿cómo maneja Claude Code los permisos?

¿qué skills están disponibles?

¿cómo uso MCP con Claude Code?

¿cómo configuro Claude Code para Amazon Bedrock?

¿cuáles son las limitaciones de Claude Code?

Note:

Claude proporciona respuestas basadas en documentación a estas preguntas. Para ejemplos ejecutables y demostraciones prácticas, consulte las secciones de flujo de trabajo específicas anteriores.

Tip:

Consejos:

- Claude siempre tiene acceso a la documentación más reciente de Claude Code, independientemente de la versión que esté utilizando
- Haga preguntas específicas para obtener respuestas detalladas
- Claude puede explicar características complejas como integración MCP, configuraciones empresariales y flujos de trabajo avanzados

Próximos pasos

- [Mejores prácticas](#) Patrones para obtener lo máximo de Claude Code
- [Cómo funciona Claude Code](#) Comprenda el bucle agente y la gestión de contexto
- [Extender Claude Code](#) Agregue skills, hooks, MCP, subagentes y complementos
- [Implementación de referencia](#) Clone nuestra implementación de referencia del contenedor de desarrollo

Part 3: Commands & Reference

Referencia de CLI

Referencia completa de la interfaz de línea de comandos de Claude Code, incluidos comandos y banderas.

Comandos CLI

Puede iniciar sesiones, canalizar contenido, reanudar conversaciones y administrar actualizaciones con estos comandos:

Comando	Descripción	Ejemplo
<code>claude</code>	Iniciar sesión interactiva	<code>claude</code>
<code>claude "query"</code>	Iniciar sesión interactiva con indicación inicial	<code>claude "explain this project"</code>
<code>claude -p "query"</code>	Consultar a través de SDK, luego salir	<code>claude -p "explain this function"</code>
<code>cat file claude -p "query"</code>	Procesar contenido canalizado	<code>cat logs.txt claude -p "explain"</code>
<code>claude -c</code>	Continuar la conversación más reciente en el directorio actual	<code>claude -c</code>
<code>claude -c -p "query"</code>	Continuar a través de SDK	<code>claude -c -p "Check for type errors"</code>
<code>claude -r "<session>" "query"</code>	Reanudar sesión por ID o nombre	<code>claude -r "auth-refactor" "Finish this PR"</code>
<code>claude update</code>	Actualizar a la versión más reciente	<code>claude update</code>

Comando	Descripción	Ejemplo
<code>claude auth login</code>	Inicie sesión en su cuenta de Anthropic. Use <code>--email</code> para rellenar previamente su dirección de correo electrónico y <code>--sso</code> para forzar la autenticación SSO	<code>claude auth login --email user@example.com --sso</code>
<code>claude auth logout</code>	Cerrar sesión en su cuenta de Anthropic	<code>claude auth logout</code>
<code>claude auth status</code>	Mostrar estado de autenticación como JSON. Use <code>--text</code> para salida legible por humanos. Sale con código 0 si ha iniciado sesión, 1 si no	<code>claude auth status</code>
<code>claude agents</code>	Enumerar todos los subagents configurados, agrupados por fuente	<code>claude agents</code>
<code>claude mcp</code>	Configurar servidores Model Context Protocol (MCP)	Consulte la documentación de Claude Code MCP .
<code>claude remote-control</code>	Iniciar una sesión de Remote Control para controlar Claude Code desde Claude.ai o la aplicación Claude mientras se ejecuta localmente. Consulte Remote Control para banderas	<code>claude remote-control</code>

Banderas CLI

Personalice el comportamiento de Claude Code con estas banderas de línea de comandos:

Bandera	Descripción	Ejemplo
<code>--add-dir</code>	Agregar directorios de trabajo adicionales para que Claude acceda (valida que cada ruta exista como directorio)	<code>claude --add-dir ../apps ../lib</code>
<code>--agent</code>	Especificar un agente para la sesión actual (anula la configuración <code>agent</code>)	<code>claude --agent my-custom-agent</code>

Bandera	Descripción	Ejemplo
<code>--agents</code>	Definir subagents personalizados dinámicamente a través de JSON (consulte a continuación el formato)	<pre>claude --agents '{"reviewer": {"description":"Reviews code","prompt":"You are a code reviewer"}}'</pre>
<code>--allow-dangerously-skip-permissions</code>	Habilitar el bypass de permisos como opción sin activarlo inmediatamente. Permite componer con <code>--permission-mode</code> (usar con cuidado)	<pre>claude --permission- mode plan --allow- dangerously-skip- permissions</pre>
<code>--allowedTools</code>	Herramientas que se ejecutan sin solicitar permiso. Consulte sintaxis de regla de permiso para coincidencia de patrones. Para restringir qué herramientas están disponibles, use <code>--tools</code> en su lugar	<pre>"Bash(git log *)" "Bash(git diff *)" "Read"</pre>
<code>--append-system-prompt</code>	Agregar texto personalizado al final del indicador del sistema predeterminado	<pre>claude --append- system-prompt "Always use TypeScript"</pre>
<code>--append-system-prompt-file</code>	Cargar texto de indicación del sistema adicional desde un archivo y agregar al indicador predeterminado	<pre>claude --append- system-prompt-file ./ extra-rules.txt</pre>
<code>--betas</code>	Encabezados beta para incluir en solicitudes de API (solo usuarios con clave API)	<pre>claude --betas interleaved-thinking</pre>
<code>--chrome</code>	Habilitar integración del navegador Chrome para automatización web y pruebas	<pre>claude --chrome</pre>
<code>--continue</code> , <code>-c</code>	Cargar la conversación más reciente en el directorio actual	<pre>claude --continue</pre>
<code>--dangerously-skip-permissions</code>	Omitir todos los indicadores de permiso (usar con cuidado)	<pre>claude --dangerously- skip-permissions</pre>

Bandera	Descripción	Ejemplo
<code>--debug</code>	Habilitar modo de depuración con filtrado de categoría opcional (por ejemplo, <code>"api,hooks"</code> o <code>"!statsig,!file"</code>)	<code>claude --debug "api,mcp"</code>
<code>--disable-slash-commands</code>	Deshabilitar todas las skills y comandos para esta sesión	<code>claude --disable-slash-commands</code>
<code>--disallowedTools</code>	Herramientas que se eliminan del contexto del modelo y no se pueden usar	<code>"Bash(git log *)"</code> <code>"Bash(git diff *)"</code> <code>"Edit"</code>
<code>--fallback-model</code>	Habilitar conmutación automática al modelo especificado cuando el modelo predeterminado está sobrecargado (solo modo de impresión)	<code>claude -p --fallback-model sonnet "query"</code>
<code>--fork-session</code>	Al reanudar, crear un nuevo ID de sesión en lugar de reutilizar el original (usar con <code>--resume</code> o <code>--continue</code>)	<code>claude --resume abc123 --fork-session</code>
<code>--from-pr</code>	Reanudar sesiones vinculadas a un PR específico de GitHub. Acepta un número de PR o URL. Las sesiones se vinculan automáticamente cuando se crean a través de <code>gh pr create</code>	<code>claude --from-pr 123</code>
<code>--ide</code>	Conectarse automáticamente al IDE al iniciar si exactamente un IDE válido está disponible	<code>claude --ide</code>
<code>--init</code>	Ejecutar hooks de inicialización e iniciar modo interactivo	<code>claude --init</code>
<code>--init-only</code>	Ejecutar hooks de inicialización y salir (sin sesión interactiva)	<code>claude --init-only</code>
<code>--include-partial-messages</code>	Incluir eventos de transmisión parcial en la salida (requiere <code>--print</code> y <code>--output-format=stream-json</code>)	<code>claude -p --output-format stream-json --include-partial-messages "query"</code>

Bandera	Descripción	Ejemplo
<code>--input-format</code>	Especificar formato de entrada para modo de impresión (opciones: <code>text</code> , <code>stream-json</code>)	<code>claude -p --output-format json --input-format stream-json</code>
<code>--json-schema</code>	Obtener salida JSON validada que coincida con un JSON Schema después de que el agente complete su flujo de trabajo (solo modo de impresión, consulte salidas estructuradas)	<code>claude -p --json-schema '{"type":"object","properties":{"...}}" "query"</code>
<code>--maintenance</code>	Ejecutar hooks de mantenimiento y salir	<code>claude --maintenance</code>
<code>--max-budget-usd</code>	Cantidad máxima en dólares a gastar en llamadas API antes de detener (solo modo de impresión)	<code>claude -p --max-budget-usd 5.00 "query"</code>
<code>--max-turns</code>	Limitar el número de turnos agentes (solo modo de impresión). Sale con un error cuando se alcanza el límite. Sin límite por defecto	<code>claude -p --max-turns 3 "query"</code>
<code>--mcp-config</code>	Cargar servidores MCP desde archivos JSON o cadenas (separados por espacios)	<code>claude --mcp-config ./mcp.json</code>
<code>--model</code>	Establece el modelo para la sesión actual con un alias para el modelo más reciente (<code>sonnet</code> u <code>opus</code>) o el nombre completo de un modelo	<code>claude --model claude-sonnet-4-6</code>
<code>--no-chrome</code>	Deshabilitar integración del navegador Chrome para esta sesión	<code>claude --no-chrome</code>
<code>--no-session-persistence</code>	Deshabilitar la persistencia de sesión para que las sesiones no se guarden en disco y no se puedan reanudar (solo modo de impresión)	<code>claude -p --no-session-persistence "query"</code>
<code>--output-format</code>	Especificar formato de salida para modo de impresión (opciones: <code>text</code> , <code>json</code> , <code>stream-json</code>)	<code>claude -p "query" --output-format json</code>

Bandera	Descripción	Ejemplo
<code>--permission-mode</code>	Comenzar en un modo de permiso especificado	<code>claude --permission-mode plan</code>
<code>--permission-prompt-tool</code>	Especificar una herramienta MCP para manejar indicadores de permiso en modo no interactivo	<code>claude -p --permission-prompt-tool mcp_auth_tool "query"</code>
<code>--plugin-dir</code>	Cargar plugins desde directorios solo para esta sesión (repetible)	<code>claude --plugin-dir ./my-plugins</code>
<code>--print</code> , <code>-p</code>	Imprimir respuesta sin modo interactivo (consulte la documentación de Agent SDK para detalles de uso programático)	<code>claude -p "query"</code>
<code>--remote</code>	Crear una nueva sesión web en claude.ai con la descripción de tarea proporcionada	<code>claude --remote "Fix the login bug"</code>
<code>--resume</code> , <code>-r</code>	Reanudar una sesión específica por ID o nombre, o mostrar un selector interactivo para elegir una sesión	<code>claude --resume auth-refactor</code>
<code>--session-id</code>	Usar un ID de sesión específico para la conversación (debe ser un UUID válido)	<code>claude --session-id "550e8400-e29b-41d4-a716-446655440000"</code>
<code>--setting-sources</code>	Lista separada por comas de fuentes de configuración a cargar (<code>user</code> , <code>project</code> , <code>local</code>)	<code>claude --setting-sources user,project</code>
<code>--settings</code>	Ruta a un archivo JSON de configuración o una cadena JSON para cargar configuración adicional desde	<code>claude --settings ./settings.json</code>
<code>--strict-mcp-config</code>	Usar solo servidores MCP de <code>--mcp-config</code> , ignorando todas las demás configuraciones de MCP	<code>claude --strict-mcp-config --mcp-config ./mcp.json</code>
<code>--system-prompt</code>	Reemplazar todo el indicador del sistema con texto personalizado	<code>claude --system-prompt "You are a Python expert"</code>

Bandera	Descripción	Ejemplo
<code>--system-prompt-file</code>	Cargar indicador del sistema desde un archivo, reemplazando el indicador predeterminado	<code>claude --system-prompt-file ./custom-prompt.txt</code>
<code>--teleport</code>	Reanudar una sesión web en su terminal local	<code>claude --teleport</code>
<code>--teammate-mode</code>	Establecer cómo se muestran los compañeros del equipo de agentes : <code>auto</code> (predeterminado), <code>in-process</code> o <code>tmux</code> . Consulte configurar equipos de agentes	<code>claude --teammate-mode in-process</code>
<code>--tools</code>	Restringir qué herramientas integradas puede usar Claude. Use <code>"</code> para deshabilitar todas, <code>"default"</code> para todas, o nombres de herramientas como <code>"Bash,Edit,Read"</code>	<code>claude --tools "Bash,Edit,Read"</code>
<code>--verbose</code>	Habilitar registro detallado, muestra salida completa turno por turno	<code>claude --verbose</code>
<code>--version</code> , <code>-v</code>	Mostrar el número de versión	<code>claude -v</code>
<code>--worktree</code> , <code>-w</code>	Iniciar Claude en un git worktree aislado en <code><repo>/claude/worktrees/<name></code> . Si no se proporciona un nombre, se genera automáticamente	<code>claude -w feature-auth</code>

Tip:

La bandera `--output-format json` es particularmente útil para scripting y automatización, permitiéndole analizar las respuestas de Claude mediante programación.

Formato de bandera de agentes

La bandera `--agents` acepta un objeto JSON que define uno o más subagentes personalizados. Cada subagent requiere un nombre único (como clave) y un objeto de definición con los siguientes campos:

Campo	Requerido	Descripción
<code>description</code>	Sí	Descripción en lenguaje natural de cuándo se debe invocar el subagent
<code>prompt</code>	Sí	El indicador del sistema que guía el comportamiento del subagent
<code>tools</code>	No	Matriz de herramientas específicas que el subagent puede usar, por ejemplo [<code>"Read"</code> , <code>"Edit"</code> , <code>"Bash"</code>] . Si se omite, hereda todas las herramientas. Admite sintaxis <code>Agent(agent_type)</code>
<code>disallowedTools</code>	No	Matriz de nombres de herramientas a negar explícitamente para este subagent
<code>model</code>	No	Alias de modelo a usar: <code>sonnet</code> , <code>opus</code> , <code>haiku</code> o <code>inherit</code> . Si se omite, por defecto es <code>inherit</code>
<code>skills</code>	No	Matriz de nombres de skill para precargar en el contexto del subagent
<code>mcpServers</code>	No	Matriz de servidores MCP para este subagent. Cada entrada es una cadena de nombre de servidor o un objeto <code>{name: config}</code>
<code>maxTurns</code>	No	Número máximo de turnos agentes antes de que el subagent se detenga

Ejemplo:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  },
  "debugger": {
    "description": "Debugging specialist for errors and test failures.",
    "prompt": "You are an expert debugger. Analyze errors, identify root causes,
and provide fixes."
  }
}'
```

Para más detalles sobre cómo crear y usar subagents, consulte la [documentación de subagents](#).

Banderas de indicador del sistema

Claude Code proporciona cuatro banderas para personalizar el indicador del sistema. Las cuatro funcionan tanto en modo interactivo como no interactivo.

Bandera	Comportamiento	Caso de uso
<code>--system-prompt</code>	Reemplaza todo el indicador predefinido	Control completo sobre el comportamiento e instrucciones de Claude
<code>--system-prompt-file</code>	Reemplaza con contenido del archivo	Cargar indicadores desde archivos para reproducibilidad y control de versiones
<code>--append-system-prompt</code>	Agrega al indicador predeterminado	Agregar instrucciones específicas mientras se mantiene el comportamiento predeterminado de Claude Code

Bandera	Comportamiento	Caso de uso
<code>--append-system-prompt-file</code>	Agrega contenido del archivo al indicador predeterminado	Cargar instrucciones adicionales desde archivos mientras se mantienen los valores predeterminados

Cuándo usar cada una:

- **`--system-prompt`** : usar cuando necesite control completo sobre el indicador del sistema de Claude. Esto elimina todas las instrucciones predeterminadas de Claude Code, dándole una pizarra en blanco.

```
claude --system-prompt "You are a Python expert who only writes type-annotated code"
```

- **`--system-prompt-file`** : usar cuando desee cargar un indicador personalizado desde un archivo, útil para la consistencia del equipo o plantillas de indicadores controladas por versiones.

```
claude --system-prompt-file ./prompts/code-review.txt
```

- **`--append-system-prompt`** : usar cuando desee agregar instrucciones específicas mientras mantiene intactas las capacidades predeterminadas de Claude Code. Esta es la opción más segura para la mayoría de los casos de uso.

```
claude --append-system-prompt "Always use TypeScript and include JSDoc comments"
```

- **`--append-system-prompt-file`** : usar cuando desee agregar instrucciones desde un archivo mientras mantiene los valores predeterminados de Claude Code. Útil para adiciones controladas por versiones.

```
claude --append-system-prompt-file ./prompts/style-rules.txt
```

`--system-prompt` y `--system-prompt-file` son mutuamente excluyentes. Las banderas de adición se pueden usar junto con cualquiera de las banderas de reemplazo.

Para la mayoría de los casos de uso, se recomienda `--append-system-prompt` o `--append-system-prompt-file` ya que preservan las capacidades integradas de Claude Code mientras agregan sus requisitos personalizados. Use `--system-prompt` o `--system-prompt-file` solo cuando necesite control completo sobre el indicador del sistema.

Ver también

- [Extensión de Chrome](#) - Automatización de navegador y pruebas web
- [Modo interactivo](#) - Atajos de teclado, modos de entrada y características interactivas
- [Guía de inicio rápido](#) - Introducción a Claude Code
- [Flujos de trabajo comunes](#) - Flujos de trabajo y patrones avanzados
- [Configuración](#) - Opciones de configuración
- [Documentación de Agent SDK](#) - Uso programático e integraciones

Personalizar atajos de teclado

Personaliza atajos de teclado en Claude Code con un archivo de configuración de `keybindings`.

Note:

Los atajos de teclado personalizables requieren Claude Code v2.1.18 o posterior. Verifique su versión con `claude --version`.

Claude Code admite atajos de teclado personalizables. Ejecute `/keybindings` para crear o abrir su archivo de configuración en `~/.claude/keybindings.json`.

Archivo de configuración

El archivo de configuración de `keybindings` es un objeto con un array `bindings`. Cada bloque especifica un contexto y un mapa de pulsaciones de teclas a acciones.

Note:

Los cambios en el archivo de `keybindings` se detectan y aplican automáticamente sin reiniciar Claude Code.

Campo	Descripción
<code>\$schema</code>	URL de esquema JSON opcional para autocompletado del editor
<code>\$docs</code>	URL de documentación opcional
<code>bindings</code>	Array de bloques de vinculación por contexto

Este ejemplo vincula `Ctrl+E` para abrir un editor externo en el contexto de chat, y desvincula `Ctrl+U`:

```
{
  "$schema": "https://www.schemastore.org/claude-code-keybindings.json",
  "$docs": "https://code.claude.com/docs/es/keybindings",
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+e": "chat:externalEditor",
        "ctrl+u": null
      }
    }
  ]
}
```

Contextos

Cada bloque de vinculación especifica un **contexto** donde se aplican los atajos:

Contexto	Descripción
Global	Se aplica en todas partes de la aplicación
Chat	Área principal de entrada de chat
Autocomplete	Menú de autocompletado está abierto
Settings	Menú de configuración (cierre solo con escape)
Confirmation	Diálogos de permiso y confirmación
Tabs	Componentes de navegación de pestañas
Help	Menú de ayuda es visible
Transcript	Visor de transcripción
HistorySearch	Modo de búsqueda de historial (Ctrl+R)
Task	Tarea de fondo está en ejecución
ThemePicker	Diálogo de selector de tema
Attachments	Navegación de barra de imagen/adjunto

Contexto	Descripción
Footer	Navegación de indicador de pie de página (tareas, equipos, diff)
MessageSelector	Selección de mensaje de diálogo de rebobinado y resumen
DiffDialog	Navegación del visor de diff
ModelPicker	Nivel de esfuerzo del selector de modelo
Select	Componentes genéricos de selección/lista
Plugin	Diálogo de plugin (examinar, descubrir, administrar)

Acciones disponibles

Las acciones siguen un formato `namespace:action` , como `chat:submit` para enviar un mensaje o `app:toggleTodos` para mostrar la lista de tareas. Cada contexto tiene acciones específicas disponibles.

Acciones de aplicación

Acciones disponibles en el contexto `Global` :

Acción	Predeterminado	Descripción
<code>app:interrupt</code>	Ctrl+C	Cancelar operación actual
<code>app:exit</code>	Ctrl+D	Salir de Claude Code
<code>app:toggleTodos</code>	Ctrl+T	Alternar visibilidad de lista de tareas
<code>app:toggleTranscript</code>	Ctrl+O	Alternar transcripción de-tallada

Acciones de historial

Acciones para navegar por el historial de comandos:

Acción	Predeterminado	Descripción
<code>history:search</code>	Ctrl+R	Abrir búsqueda de historial

Acción	Predeterminado	Descripción
<code>history:previous</code>	Arriba	Elemento de historial anterior
<code>history:next</code>	Abajo	Siguiente elemento de historial

Acciones de chat

Acciones disponibles en el contexto `Chat` :

Acción	Predeterminado	Descripción
<code>chat:cancel</code>	Escape	Cancelar entrada actual
<code>chat:cycleMode</code>	Shift+Tab*	Ciclar modos de permiso
<code>chat:modelPicker</code>	Cmd+P / Meta+P	Abrir selector de modelo
<code>chat:thinkingToggle</code>	Cmd+T / Meta+T	Alternar pensamiento extendido
<code>chat:submit</code>	Enter	Enviar mensaje
<code>chat:undo</code>	Ctrl+_	Deshacer última acción
<code>chat:externalEditor</code>	Ctrl+G	Abrir en editor externo
<code>chat:stash</code>	Ctrl+S	Guardar indicación actual
<code>chat:imagePaste</code>	Ctrl+V (Alt+V en Windows)	Pegar imagen

*En Windows sin modo VT (Node <24.2.0/<22.17.0, Bun <1.2.23), el valor predeterminado es Meta+M.

Acciones de autocompletado

Acciones disponibles en el contexto `Autocomplete` :

Acción	Predeterminado	Descripción
<code>autocomplete:accept</code>	Tab	Aceptar sugerencia
<code>autocomplete:dismiss</code>	Escape	Descartar menú
<code>autocomplete:previous</code>	Arriba	Sugerencia anterior

Acción	Predeterminado	Descripción
<code>autocomplete:next</code>	Abajo	Siguiente sugerencia

Acciones de confirmación

Acciones disponibles en el contexto `Confirmation` :

Acción	Predeterminado	Descripción
<code>confirm:yes</code>	Y, Enter	Confirmar acción
<code>confirm:no</code>	N, Escape	Rechazar acción
<code>confirm:previous</code>	Arriba	Opción anterior
<code>confirm:next</code>	Abajo	Siguiente opción
<code>confirm:nextField</code>	Tab	Siguiente campo
<code>confirm:previousField</code>	(sin vincular)	Campo anterior
<code>confirm:cycleMode</code>	Shift+Tab	Ciclar modos de permiso
<code>confirm:toggleExplanation</code>	Ctrl+E	Alternar explicación de permiso

Acciones de permiso

Acciones disponibles en el contexto `Confirmation` para diálogos de permiso:

Acción	Predeterminado	Descripción
<code>permission:toggleDebug</code>	Ctrl+D	Alternar información de depuración de permiso

Acciones de transcripción

Acciones disponibles en el contexto `Transcript` :

Acción	Predeterminado	Descripción
<code>transcript:toggleShowAll</code>	Ctrl+E	Alternar mostrar todo el contenido

Acción	Predeterminado	Descripción
<code>transcript:exit</code>	Ctrl+C, Escape	Salir de vista de transcripción

Acciones de búsqueda de historial

Acciones disponibles en el contexto `HistorySearch` :

Acción	Predeterminado	Descripción
<code>historySearch:next</code>	Ctrl+R	Siguiente coincidencia
<code>historySearch:accept</code>	Escape, Tab	Aceptar selección
<code>historySearch:cancel</code>	Ctrl+C	Cancelar búsqueda
<code>historySearch:execute</code>	Enter	Ejecutar comando seleccionado

Acciones de tarea

Acciones disponibles en el contexto `Task` :

Acción	Predeterminado	Descripción
<code>task:background</code>	Ctrl+B	Tarea de fondo actual

Acciones de tema

Acciones disponibles en el contexto `ThemePicker` :

Acción	Predeterminado	Descripción
<code>theme:toggleSyntaxHighlighting</code>	Ctrl+T	Alternar resaltado de sintaxis

Acciones de ayuda

Acciones disponibles en el contexto `Help` :

Acción	Predeterminado	Descripción
<code>help:dismiss</code>	Escape	Cerrar menú de ayuda

Acciones de pestañas

Acciones disponibles en el contexto **Tabs** :

Acción	Predeterminado	Descripción
<code>tabs:next</code>	Tab, Derecha	Siguiente pestaña
<code>tabs:previous</code>	Shift+Tab, Izquierda	Pestaña anterior

Acciones de adjuntos

Acciones disponibles en el contexto **Attachments** :

Acción	Predeterminado	Descripción
<code>attachments:next</code>	Derecha	Siguiente adjunto
<code>attachments:previous</code>	Izquierda	Adjunto anterior
<code>attachments:remove</code>	Retroceso, Suprimir	Eliminar adjunto seleccionado
<code>attachments:exit</code>	Abajo, Escape	Salir de barra de adjuntos

Acciones de pie de página

Acciones disponibles en el contexto **Footer** :

Acción	Predeterminado	Descripción
<code>footer:next</code>	Derecha	Siguiente elemento de pie de página
<code>footer:previous</code>	Izquierda	Elemento de pie de página anterior
<code>footer:openSelected</code>	Enter	Abrir elemento de pie de página seleccionado
<code>footer:clearSelection</code>	Escape	Limpiar selección de pie de página

Acciones del selector de mensajes

Acciones disponibles en el contexto **MessageSelector** :

Acción	Predeterminado	Descripción
<code>messageSelector:up</code>	Arriba, K, Ctrl+P	Mover hacia arriba en la lista
<code>messageSelector:down</code>	Abajo, J, Ctrl+N	Mover hacia abajo en la lista
<code>messageSelector:top</code>	Ctrl+Arriba, Shift+Arriba, Meta+Arriba, Shift+K	Saltar al inicio
<code>messageSelector:bottom</code>	Ctrl+Abajo, Shift+Abajo, Meta+Abajo, Shift+J	Saltar al final
<code>messageSelector:select</code>	Enter	Seleccionar mensaje

Acciones de diff

Acciones disponibles en el contexto `DiffDialog` :

Acción	Predeterminado	Descripción
<code>diff:dismiss</code>	Escape	Cerrar visor de diff
<code>diff:previousSource</code>	Izquierda	Fuente de diff anterior
<code>diff:nextSource</code>	Derecha	Siguiente fuente de diff
<code>diff:previousFile</code>	Arriba	Archivo anterior en diff
<code>diff:nextFile</code>	Abajo	Siguiente archivo en diff
<code>diff:viewDetails</code>	Enter	Ver detalles de diff
<code>diff:back</code>	(específico del contexto)	Volver atrás en visor de diff

Acciones del selector de modelo

Acciones disponibles en el contexto `ModelPicker` :

Acción	Predeterminado	Descripción
<code>modelPicker:decreaseEffort</code>	Izquierda	Disminuir nivel de esfuerzo

Acción	Predeterminado	Descripción
<code>modelPicker:increaseEffort</code>	Derecha	Aumentar nivel de esfuerzo

Acciones de selección

Acciones disponibles en el contexto `Select` :

Acción	Predeterminado	Descripción
<code>select:next</code>	Abajo, J, Ctrl+N	Siguiente opción
<code>select:previous</code>	Arriba, K, Ctrl+P	Opción anterior
<code>select:accept</code>	Enter	Aceptar selección
<code>select:cancel</code>	Escape	Cancelar selección

Acciones de plugin

Acciones disponibles en el contexto `Plugin` :

Acción	Predeterminado	Descripción
<code>plugin:toggle</code>	Espacio	Alternar selección de plugin
<code>plugin:install</code>	I	Instalar plugins seleccionados

Acciones de configuración

Acciones disponibles en el contexto `Settings` :

Acción	Predeterminado	Descripción
<code>settings:search</code>	/	Entrar en modo de búsqueda
<code>settings:retry</code>	R	Reintentar carga de datos de uso (en caso de error)

Sintaxis de pulsación de tecla

Modificadores

Use teclas modificadoras con el separador **+** :

- **ctrl** o **control** - Tecla Control
- **alt**, **opt**, u **option** - Tecla Alt/Opción
- **shift** - Tecla Shift
- **meta**, **cmd**, o **command** - Tecla Meta/Comando

Por ejemplo:

<code>ctrl+k</code>	Tecla única con modificador
<code>shift+tab</code>	Shift + Tab
<code>meta+p</code>	Comando/Meta + P
<code>ctrl+shift+c</code>	Múltiples modificadores

Letras mayúsculas

Una letra mayúscula independiente implica Shift. Por ejemplo, **K** es equivalente a **shift+k**. Esto es útil para atajos de estilo vim donde las teclas mayúsculas y minúsculas tienen significados diferentes.

Las letras mayúsculas con modificadores (por ejemplo, **ctrl+K**) se tratan como estilísticas y **no** implican Shift — **ctrl+K** es lo mismo que **ctrl+k**.

Acordes

Los acordes son secuencias de pulsaciones de teclas separadas por espacios:

```
ctrl+k ctrl+s  Presione Ctrl+K, suelte, luego Ctrl+S
```

Teclas especiales

- **escape** o **esc** - Tecla Escape
- **enter** o **return** - Tecla Enter
- **tab** - Tecla Tab
- **space** - Barra espaciadora
- **up**, **down**, **left**, **right** - Teclas de flecha
- **backspace**, **delete** - Teclas de eliminación

Desvinculación de atajos predeterminados

Establezca una acción en `null` para desvinculación de un atajo predeterminado:

```
{
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+s": null
      }
    }
  ]
}
```

Atajos reservados

Estos atajos no se pueden reasignar:

Atajo	Razón
Ctrl+C	Interrupción/cancelación codificada
Ctrl+D	Salida codificada

Conflictos de terminal

Algunos atajos pueden entrar en conflicto con multiplexores de terminal:

Atajo	Conflicto
Ctrl+B	Prefijo de tmux (presione dos veces para enviar)
Ctrl+A	Prefijo de GNU screen
Ctrl+Z	Suspensión de proceso Unix (SIGTSTP)

Interacción del modo Vim

Cuando el modo vim está habilitado (`/vim`), los keybindings y el modo vim funcionan de forma independiente:

- **Modo Vim** maneja la entrada a nivel de entrada de texto (movimiento del cursor, modos, movimientos)
- **Keybindings** manejan acciones a nivel de componente (alternar tareas, enviar, etc.)
- La tecla Escape en modo vim cambia de modo INSERT a NORMAL; no activa `chat:cancel`
- La mayoría de los atajos Ctrl+tecla pasan a través del modo vim al sistema de keybindings
- En modo NORMAL de vim, `?` muestra el menú de ayuda (comportamiento de vim)

Validación

Claude Code valida sus keybindings y muestra advertencias para:

- Errores de análisis (JSON o estructura inválida)
- Nombres de contexto inválidos
- Conflictos de atajos reservados
- Conflictos de multiplexor de terminal
- Vinculaciones duplicadas en el mismo contexto

Ejecute `/doctor` para ver cualquier advertencia de keybindings.

Part 4: Configuration

Configurar permisos

Controle lo que Claude Code puede acceder y hacer con reglas de permisos granulares, modos y políticas administradas.

Claude Code admite permisos granulares para que pueda especificar exactamente qué puede hacer el agente y qué no puede hacer. La configuración de permisos se puede registrar en el control de versiones y distribuir a todos los desarrolladores de su organización, así como personalizarse por desarrolladores individuales.

Sistema de permisos

Claude Code utiliza un sistema de permisos escalonado para equilibrar potencia y seguridad:

Tipo de herramienta	Ejemplo	Se requiere aprobación	Comportamiento de "Sí, no preguntar de nuevo"
Solo lectura	Lecturas de archivos, Grep	No	N/A
Comandos Bash	Ejecución de shell	Sí	Permanentemente por directorio de proyecto y comando
Modificación de archivos	Editar/escribir archivos	Sí	Hasta el final de la sesión

Administrar permisos

Puede ver y administrar los permisos de herramientas de Claude Code con `/permissions`. Esta interfaz de usuario enumera todas las reglas de permisos y el archivo `settings.json` del que se obtienen.

- Las reglas **Allow** permiten que Claude Code use la herramienta especificada sin aprobación manual.

- Las reglas **Ask** solicitan confirmación cada vez que Claude Code intenta usar la herramienta especificada.
- Las reglas **Deny** impiden que Claude Code use la herramienta especificada.

Las reglas se evalúan en orden: **deny** -> **ask** -> **allow**. La primera regla coincidente gana, por lo que las reglas de negación siempre tienen prioridad.

Modos de permisos

Claude Code admite varios modos de permisos que controlan cómo se aprueban las herramientas. Establezca `defaultMode` en sus [archivos de configuración](#):

Modo	Descripción
<code>default</code>	Comportamiento estándar: solicita permiso en el primer uso de cada herramienta
<code>acceptEdits</code>	Acepta automáticamente los permisos de edición de archivos para la sesión
<code>plan</code>	Plan Mode: Claude puede analizar pero no modificar archivos ni ejecutar comandos
<code>dontAsk</code>	Deniega automáticamente las herramientas a menos que estén preaprobadas a través de <code>/permissions</code> o reglas <code>permissions.allow</code>
<code>bypassPermissions</code>	Omite todos los avisos de permisos (requiere entorno seguro, ver advertencia a continuación)

Warning:

El modo `bypassPermissions` desactiva todas las comprobaciones de permisos. Use esto solo en entornos aislados como contenedores o máquinas virtuales donde Claude Code no pueda causar daño. Los administradores pueden evitar este modo estableciendo `disableBypassPermissionsMode` en `"disable"` en [configuración administrada](#).

Sintaxis de reglas de permisos

Las reglas de permisos siguen el formato `Tool` o `Tool(specifier)`.

Coincidir con todos los usos de una herramienta

Para coincidir con todos los usos de una herramienta, use solo el nombre de la herramienta sin paréntesis:

Regla	Efecto
<code>Bash</code>	Coincide con todos los comandos Bash
<code>WebFetch</code>	Coincide con todas las solicitudes de obtención web
<code>Read</code>	Coincide con todas las lecturas de archivos

`Bash(*)` es equivalente a `Bash` y coincide con todos los comandos Bash.

Usar especificadores para control granular

Agregue un especificador entre paréntesis para coincidir con usos específicos de herramientas:

Regla	Efecto
<code>Bash(npm run build)</code>	Coincide con el comando exacto <code>npm run build</code>
<code>Read(./env)</code>	Coincide con la lectura del archivo <code>.env</code> en el directorio actual
<code>WebFetch(domain:example.com)</code>	Coincide con solicitudes de obtención a <code>example.com</code>

Patrones de comodín

Las reglas de Bash admiten patrones glob con `*`. Los comodines pueden aparecer en cualquier posición del comando. Esta configuración permite comandos npm y git commit mientras bloquea git push:


```
{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)",
      "Bash(git * main)",
      "Bash(* --version)",
      "Bash(* --help *)"
    ],
    "deny": [
      "Bash(git push *)"
    ]
  }
}
```

El espacio antes de `*` importa: `Bash(ls *)` coincide con `ls -la` pero no con `ls -of`, mientras que `Bash(ls*)` coincide con ambos. La sintaxis de sufijo heredada `:*` es equivalente a `*` pero está depreciada.

Reglas de permisos específicas de herramientas

Bash

Las reglas de permisos de Bash admiten coincidencia de comodines con `*`. Los comodines pueden aparecer en cualquier posición del comando, incluyendo al principio, en el medio o al final:

- `Bash(npm run build)` coincide con el comando Bash exacto `npm run build`
- `Bash(npm run test *)` coincide con comandos Bash que comienzan con `npm run test`
- `Bash(npm *)` coincide con cualquier comando que comience con `npm`
- `Bash(* install)` coincide con cualquier comando que termine con `install`
- `Bash(git * main)` coincide con comandos como `git checkout main`, `git merge main`

Cuando `*` aparece al final con un espacio antes (como `Bash(ls *)`), aplica un límite de palabra, requiriendo que el prefijo sea seguido por un espacio o fin de cadena. Por ejemplo, `Bash(ls *)` coincide con `ls -la` pero no con `ls -of`. En contraste, `Bash(ls*)` sin espacio coincide con ambos `ls -la` y `ls -of` porque no hay restricción de límite de palabra.

Tip:

Claude Code es consciente de los operadores de shell (como `&&`) por lo que una regla de coincidencia de prefijo como `Bash(safe-cmd *)` no le dará permiso para ejecutar el comando `safe-cmd && other-cmd`.

Warning:

Los patrones de permisos de Bash que intentan restringir argumentos de comando son frágiles. Por ejemplo, `Bash(curl http://github.com/ *)` intenta restringir curl a URLs de GitHub, pero no coincidirá con variaciones como:

- Opciones antes de URL: `curl -X GET http://github.com/ ...`
- Protocolo diferente: `curl https://github.com/ ...`
- Redirecciones: `curl -L http://bit.ly/xyz` (redirige a github)
- Variables: `URL=http://github.com && curl $URL`
- Espacios adicionales: `curl http://github.com`

Para un filtrado de URL más confiable, considere:

- **Restringir herramientas de red de Bash:** use reglas de negación para bloquear `curl`, `wget` y comandos similares, luego use la herramienta WebFetch con permiso `WebFetch(domain:github.com)` para dominios permitidos
- **Usar hooks PreToolUse:** implemente un hook que valide URLs en comandos Bash y bloquee dominios no permitidos
- Instruir a Claude Code sobre sus patrones curl permitidos a través de CLAUDE.md

Tenga en cuenta que usar WebFetch solo no previene el acceso a la red. Si se permite Bash, Claude aún puede usar `curl`, `wget` u otras herramientas para alcanzar cualquier URL.

Read y Edit

Las reglas `Edit` se aplican a todas las herramientas integradas que editan archivos. Claude hace un esfuerzo de mejor intento para aplicar reglas `Read` a todas las herramientas integradas que leen archivos como Grep y Glob.

Las reglas Read y Edit siguen la especificación [gitignore](#) con cuatro tipos de patrones distintos:

Patrón	Significado	Ejemplo	Coincide
<code>//path</code>	Ruta absoluta desde la raíz del sistema de archivos	<code>Read(//Users/alice/secrets/**)</code>	<code>/Users/alice/secrets/**</code>

Patrón	Significado	Ejemplo	Coincide
<code>~/path</code>	Ruta desde el directorio home	<code>Read(~/Documents/ *.pdf)</code>	<code>/Users/alice/ Documents/*.pdf</code>
<code>/path</code>	Ruta relativa a la raíz del proyecto	<code>Edit(/src/**/ *.ts)</code>	<code><project root>/ src/**/*.ts</code>
<code>path</code> o <code>./path</code>	Ruta relativa al directorio actual	<code>Read(*.env)</code>	<code><cwd>/*.env</code>

Warning:

Un patrón como `/Users/alice/file` NO es una ruta absoluta. Es relativa a la raíz del proyecto. Use `//Users/alice/file` para rutas absolutas.

Ejemplos:

- `Edit(/docs/**)` : edita en `<project>/docs/` (NO `/docs/` y NO `<project>/.claude/docs/`)
- `Read(~/.zshrc)` : lee el `.zshrc` de su directorio home
- `Edit(/tmp/scratch.txt)` : edita la ruta absoluta `/tmp/scratch.txt`
- `Read(src/**)` : lee desde `<current-directory>/src/`

Note:

En patrones gitignore, `*` coincide con archivos en un solo directorio mientras que `**` coincide recursivamente en directorios. Para permitir todo acceso a archivos, use solo el nombre de la herramienta sin paréntesis: `Read` , `Edit` o `Write` .

WebFetch

- `WebFetch(domain:example.com)` coincide con solicitudes de obtención a `example.com`

MCP

- `mcp__puppeteer` coincide con cualquier herramienta proporcionada por el servidor `puppeteer` (nombre configurado en Claude Code)
- `mcp_puppeteer__*` sintaxis de comodín que también coincide con todas las herramientas del servidor `puppeteer`
- `mcp_puppeteer_puppeteer_navigate` coincide con la herramienta `puppeteer_navigate` proporcionada por el servidor `puppeteer`

Agent (subagents)

Use reglas `Agent(AgentName)` para controlar qué [subagents](#) puede usar Claude:

- `Agent(Explore)` coincide con el subagent Explore
- `Agent(Plan)` coincide con el subagent Plan
- `Agent(my-custom-agent)` coincide con un subagent personalizado llamado `my-custom-agent`

Agregue estas reglas a la matriz `deny` en su configuración o use la bandera CLI `--disallowedTools` para deshabilitar agentes específicos. Para deshabilitar el agente Explore:

```
{
  "permissions": {
    "deny": ["Agent(Explore)"]
  }
}
```

Extender permisos con hooks

Los [hooks de Claude Code](#) proporcionan una forma de registrar comandos de shell personalizados para realizar evaluación de permisos en tiempo de ejecución. Cuando Claude Code realiza una llamada de herramienta, los hooks `PreToolUse` se ejecutan antes del sistema de permisos, y la salida del hook puede determinar si aprobar o denegar la llamada de herramienta en lugar del sistema de permisos.

Directorios de trabajo

Por defecto, Claude tiene acceso a archivos en el directorio donde fue lanzado. Puede extender este acceso:

- **Durante el inicio:** use el argumento CLI `--add-dir <path>`
- **Durante la sesión:** use el comando `/add-dir`
- **Configuración persistente:** agregue a `additionalDirectories` en [archivos de configuración](#)

Los archivos en directorios adicionales siguen las mismas reglas de permisos que el directorio de trabajo original: se vuelven legibles sin avisos, y los permisos de edición de archivos siguen el modo de permisos actual.

Cómo interactúan los permisos con el sandboxing

Los permisos y el [sandboxing](#) son capas de seguridad complementarias:

- **Permisos** controlan qué herramientas puede usar Claude Code y qué archivos o dominios puede acceder. Se aplican a todas las herramientas (Bash, Read, Edit, WebFetch, MCP y otras).
- **Sandboxing** proporciona aplicación a nivel del SO que restringe el acceso del sistema de archivos y red de la herramienta Bash. Se aplica solo a comandos Bash y sus procesos secundarios.

Use ambos para defensa en profundidad:

- Las reglas de negación de permisos impiden que Claude intente acceder a recursos restringidos
- Las restricciones de sandbox previenen que comandos Bash alcancen recursos fuera de límites definidos, incluso si una inyección de solicitud omite la toma de decisiones de Claude
- Las restricciones del sistema de archivos en el sandbox usan reglas de negación Read y Edit, no configuración de sandbox separada
- Las restricciones de red combinan reglas de permisos WebFetch con la lista `allowedDomains` del sandbox

Configuración administrada

Para organizaciones que necesitan control centralizado sobre la configuración de Claude Code, los administradores pueden implementar configuración administrada que no puede ser anulada por configuración de usuario o proyecto. Estas configuraciones de política siguen el mismo formato que archivos de configuración regulares y se pueden entregar a través de políticas MDM/a nivel del SO, archivos de configuración administrada o [configuración administrada por servidor](#). Consulte [archivos de configuración](#) para mecanismos de entrega y ubicaciones de archivos.

Configuración solo administrada

Algunas configuraciones solo son efectivas en configuración administrada:

Configuración	Descripción
<code>disableBypassPermissionsMode</code>	Establezca en <code>"disable"</code> para evitar el modo <code>bypassPermissions</code> y la bandera <code>-dangerously-skip-permissions</code>

Configuración	Descripción
<code>allowManagedPermissionRulesOnly</code>	Cuando es <code>true</code> , evita que la configuración de usuario y proyecto defina reglas de permisos <code>allow</code> , <code>ask</code> o <code>deny</code> . Solo se aplican las reglas en configuración administrada
<code>allowManagedHooksOnly</code>	Cuando es <code>true</code> , evita la carga de hooks de usuario, proyecto y plugin. Solo se permiten hooks administrados y hooks SDK
<code>allowManagedMcpServersOnly</code>	Cuando es <code>true</code> , solo se respetan <code>allowedMcpServers</code> de configuración administrada. <code>deniedMcpServers</code> aún se fusiona de todas las fuentes. Consulte Configuración MCP administrada
<code>blockedMarketplaces</code>	Lista de bloqueo de fuentes de marketplace. Las fuentes bloqueadas se verifican antes de descargar, por lo que nunca tocan el sistema de archivos. Consulte restricciones de marketplace administradas
<code>sandbox.network.allowManagedDomainsOnly</code>	Cuando es <code>true</code> , solo se respetan <code>allowedDomains</code> y reglas de permiso <code>WebFetch(domain:...)</code> de configuración administrada. Los dominios no permitidos se bloquean automáticamente sin solicitar al usuario. Los dominios denegados aún se fusionan de todas las fuentes
<code>strictKnownMarketplaces</code>	Controla qué marketplaces de plugins pueden agregar los usuarios. Consulte restricciones de marketplace administradas
<code>allow_remote_sessions</code>	Cuando es <code>true</code> , permite a los usuarios iniciar Control Remoto y sesiones web . Por defecto es <code>true</code> . Establezca en <code>false</code> para evitar acceso a sesiones remotas

Precedencia de configuración

Las reglas de permisos siguen la misma [precedencia de configuración](#) que todas las demás configuraciones de Claude Code:

1. **Configuración administrada:** no puede ser anulada por ningún otro nivel, incluyendo argumentos de línea de comandos
2. **Argumentos de línea de comandos:** anulaciones de sesión temporal
3. **Configuración de proyecto local** (`.claude/settings.local.json`)
4. **Configuración de proyecto compartida** (`.claude/settings.json`)
5. **Configuración de usuario** (`~/.claude/settings.json`)

Si una herramienta se deniega en cualquier nivel, ningún otro nivel puede permitirla. Por ejemplo, una negación de configuración administrada no puede ser anulada por `--allowedTools`, y `--disallowedTools` puede agregar restricciones más allá de lo que define la configuración administrada.

Si un permiso se permite en configuración de usuario pero se deniega en configuración de proyecto, la configuración de proyecto tiene prioridad y el permiso se bloquea.

Configuraciones de ejemplo

Este [repositorio](#) incluye configuraciones de configuración inicial para escenarios de implementación comunes. Use estos como puntos de partida y ajústelos para que se adapten a sus necesidades.

Ver también

- [Configuración](#): referencia de configuración completa incluyendo la tabla de configuración de permisos
- [Sandboxing](#): aislamiento del sistema de archivos y red a nivel del SO para comandos Bash
- [Autenticación](#): configure el acceso de usuario a Claude Code
- [Seguridad](#): salvaguardas de seguridad y mejores prácticas
- [Hooks](#): automatice flujos de trabajo y extienda la evaluación de permisos

Cómo Claude recuerda su proyecto

Proporcione a Claude instrucciones persistentes con archivos `CLAUDE.md`, y permita que Claude acumule aprendizajes automáticamente con `auto memory`.

Cada sesión de Claude Code comienza con una ventana de contexto nueva. Dos mecanismos llevan el conocimiento entre sesiones:

- **Archivos `CLAUDE.md`:** instrucciones que usted escribe para dar a Claude contexto persistente
- **Auto memory:** notas que Claude escribe por sí mismo basadas en sus correcciones y preferencias

Esta página cubre cómo:

- [Escribir y organizar archivos `CLAUDE.md`](#)
- [Limitar reglas a tipos de archivo específicos](#) con `.claude/rules/`
- [Configurar `auto memory`](#) para que Claude tome notas automáticamente
- [Solucionar problemas](#) cuando las instrucciones no se siguen

CLAUDE.md vs auto memory

Claude Code tiene dos sistemas de memoria complementarios. Ambos se cargan al inicio de cada conversación. Claude los trata como contexto, no como configuración forzada. Cuanto más específicas y concisas sean sus instrucciones, más consistentemente Claude las seguirá.

	Archivos <code>CLAUDE.md</code>	Auto memory
Quién lo escribe	Usted	Claude
Qué contiene	Instrucciones y reglas	Aprendizajes y patrones
Alcance	Proyecto, usuario u organización	Por worktree
Se carga en	Cada sesión	Cada sesión (primeras 200 líneas)

	Archivos CLAUDE.md	Auto memory
Usar para	Estándares de codificación, flujos de trabajo, arquitectura del proyecto	Comandos de compilación, información de depuración, preferencias que Claude descubre

Use archivos CLAUDE.md cuando quiera guiar el comportamiento de Claude. Auto memory permite que Claude aprenda de sus correcciones sin esfuerzo manual.

Los subagents también pueden mantener su propia auto memory. Consulte [configuración de subagent](#) para obtener detalles.

Archivos CLAUDE.md

Los archivos CLAUDE.md son archivos markdown que dan a Claude instrucciones persistentes para un proyecto, su flujo de trabajo personal o toda su organización. Usted escribe estos archivos en texto plano; Claude los lee al inicio de cada sesión.

Elija dónde colocar los archivos CLAUDE.md

Los archivos CLAUDE.md pueden vivir en varios lugares, cada uno con un alcance diferente. Las ubicaciones más específicas tienen prioridad sobre las más amplias.

Alcan-ce	Ubicación	Propósito	Ejemplos de casos de uso	Compartido con
Políti-ca ges-tiona-da	• macOS: <code>/Library/Application Support/ClaudeCode/CLAUDE.md</code> • Linux y WSL: <code>/etc/claude-code/CLAUDE.md</code> • Windows: <code>C:\Program Files\ClaudeCode\CLAUDE.md</code>	Instrucciones de toda la organiza-ción gestionadas por TI/DevOps	Estándares de codificación de la empresa, polí-ticas de seguridad, re-quisitos de cum-plimiento	Todos los usua-rios de la organi-zación
Ins-truc-ciones del proyec-to	<code>./CLAUDE.md</code> o <code>./claude/CLAUDE.md</code>	Instrucciones compartidas por el equipo para el proyecto	Arquitectura del proyecto, estándar de codifi-cación, flujos de trabajo comunes	Miembros del equipo a través del control de versiones

Alcan-ce	Ubicación	Propósito	Ejemplos de casos de uso	Compartido con
Ins-truc-ciones del usu-a-rio	<code>~/.claude/CLAUDE.md</code>	Preferencias personales para todos los proyec-tos	Preferencias de estilo de código, atajos de herra-mientas perso-nales	Solo usted (to-dos los proyec-tos)

Los archivos CLAUDE.md en la jerarquía de directorios por encima del directorio de trabajo se cargan completamente al iniciar. Los archivos CLAUDE.md en subdirectorios se cargan bajo demanda cuando Claude lee archivos en esos directorios. Consulte [Cómo se cargan los archivos CLAUDE.md](#) para el orden de resolución completo.

Para proyectos grandes, puede dividir las instrucciones en archivos específicos de temas usando [reglas de proyecto](#). Las reglas le permiten limitar las instrucciones a tipos de archivo específicos o subdirectorios.

Configure un CLAUDE.md de proyecto

Un CLAUDE.md de proyecto se puede almacenar en `./CLAUDE.md` o `./.claude/CLAUDE.md`. Cree este archivo y agregue instrucciones que se apliquen a cualquiera que trabaje en el proyecto: comandos de compilación y prueba, estándares de codificación, decisiones arquitectónicas, convenciones de nomenclatura y flujos de trabajo comunes. Estas instrucciones se comparten con su equipo a través del control de versiones, así que enfóquese en estándares a nivel de proyecto en lugar de preferencias personales.

Tip:

Ejecute `/init` para generar un CLAUDE.md inicial automáticamente. Claude analiza su base de código y crea un archivo con comandos de compilación, instrucciones de prueba y convenciones de proyecto que descubre. Si ya existe un CLAUDE.md, `/init` sugiere mejoras en lugar de sobrescribirlo. Refine desde allí con instrucciones que Claude no descubriría por sí solo.

Escriba instrucciones efectivas

Los archivos CLAUDE.md se cargan en la ventana de contexto al inicio de cada sesión, consumiendo tokens junto con su conversación. Debido a que son contexto en lugar de configuración forzada, cómo escribe las instrucciones afecta qué tan confiablemente Claude las sigue. Las instrucciones específicas, concisas y bien estructuradas funcionan mejor.

Tamaño: apunte a menos de 200 líneas por archivo CLAUDE.md. Los archivos más largos consumen más contexto y reducen la adherencia. Si sus instrucciones están creciendo mucho, divídalas usando [importaciones](#) o archivos `.claude/rules/`.

Estructura: use encabezados markdown y viñetas para agrupar instrucciones relacionadas. Claude escanea la estructura de la misma manera que los lectores: las secciones organizadas son más fáciles de seguir que los párrafos densos.

Especificidad: escriba instrucciones que sean lo suficientemente concretas para verificar. Por ejemplo:

- “Usar indentación de 2 espacios” en lugar de “Formatear código correctamente”
- “Ejecutar `npm test` antes de confirmar” en lugar de “Probar sus cambios”
- “Los controladores de API viven en `src/api/handlers/`” en lugar de “Mantener los archivos organizados”

Consistencia: si dos reglas se contradicen entre sí, Claude puede elegir una arbitrariamente. Revise sus archivos CLAUDE.md, archivos CLAUDE.md anidados en subdirectorios y archivos `.claude/rules/` periódicamente para eliminar instrucciones desactualizadas o conflictivas. En monorepos, use `claudeMdExcludes` para omitir archivos CLAUDE.md de otros equipos que no sean relevantes para su trabajo.

Importar archivos adicionales

Los archivos CLAUDE.md pueden importar archivos adicionales usando la sintaxis `@path/to/import`. Los archivos importados se expanden y se cargan en contexto al iniciar junto con el CLAUDE.md que los referencia.

Se permiten rutas relativas y absolutas. Las rutas relativas se resuelven en relación con el archivo que contiene la importación, no con el directorio de trabajo. Los archivos importados pueden importar recursivamente otros archivos, con una profundidad máxima de cinco saltos.

Para incluir un README, package.json y una guía de flujo de trabajo, haga referencia a ellos con la sintaxis `@` en cualquier lugar de su CLAUDE.md:

```
Consulte @README para obtener una descripción general del proyecto y @package.json para los comandos npm disponibles para este proyecto.
```

```
## Instrucciones adicionales
- flujo de trabajo git @docs/git-instructions.md
```

Para preferencias personales que no desea registrar, importe un archivo desde su directorio de inicio. La importación va en el CLAUDE.md compartido, pero el archivo al que apunta permanece en su máquina:

```
## Preferencias individuales
- @~/ .claude/my-project-instructions.md
```

Warning:

La primera vez que Claude Code encuentra importaciones externas en un proyecto, muestra un diálogo de aprobación que enumera los archivos. Si rechaza, las importaciones permanecen deshabilitadas y el diálogo no aparece nuevamente.

Para un enfoque más estructurado para organizar instrucciones, consulte [.claude/rules/](#).

Cómo se cargan los archivos CLAUDE.md

Claude Code lee los archivos CLAUDE.md caminando hacia arriba en el árbol de directorios desde su directorio de trabajo actual, verificando cada directorio en el camino. Esto significa que si ejecuta Claude Code en `foo/bar/`, carga instrucciones de `foo/bar/CLAUDE.md` y `foo/CLAUDE.md`.

Claude también descubre archivos CLAUDE.md en subdirectorios bajo su directorio de trabajo actual. En lugar de cargarlos al iniciar, se incluyen cuando Claude lee archivos en esos subdirectorios.

Si trabaja en un monorepo grande donde se recogen archivos CLAUDE.md de otros equipos, use `claudeMdExcludes` para omitirlos.

Cargar desde directorios adicionales

La bandera `--add-dir` da a Claude acceso a directorios adicionales fuera de su directorio de trabajo principal. De forma predeterminada, los archivos `CLAUDE.md` de estos directorios no se cargan.

Para cargar también archivos `CLAUDE.md` de directorios adicionales, incluidos `CLAUDE.md`, `.claude/CLAUDE.md` y `.claude/rules/*.md`, establezca la variable de entorno `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`:

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

Organizar reglas con `.claude/rules/`

Para proyectos más grandes, puede organizar instrucciones en múltiples archivos usando el directorio `.claude/rules/`. Esto mantiene las instrucciones modulares y más fáciles de mantener para los equipos. Las reglas también pueden ser [limitadas a rutas de archivo específicas](#), por lo que solo se cargan en contexto cuando Claude trabaja con archivos coincidentes, reduciendo ruido y ahorrando espacio de contexto.

Note:

Las reglas se cargan en contexto cada sesión o cuando se abren archivos coincidentes. Para instrucciones específicas de tareas que no necesitan estar en contexto todo el tiempo, use [skills](#) en su lugar, que solo se cargan cuando las invoca o cuando Claude determina que son relevantes para su solicitud.

Configurar reglas

Coloque archivos markdown en el directorio `.claude/rules/` de su proyecto. Cada archivo debe cubrir un tema, con un nombre de archivo descriptivo como `testing.md` o `api-design.md`. Todos los archivos `.md` se descubren recursivamente, por lo que puede organizar reglas en subdirectorios como `frontend/` o `backend/`:

```
your-project/
├── .claude/
│   ├── CLAUDE.md          # Instrucciones principales del proyecto
│   └── rules/
│       ├── code-style.md  # Directrices de estilo de código
│       ├── testing.md     # Convenciones de prueba
│       └── security.md    # Requisitos de seguridad
```

Las reglas sin `frontmatter paths` se cargan al iniciar con la misma prioridad que `.claude/CLAUDE.md`.

Reglas específicas de ruta

Las reglas pueden limitarse a archivos específicos usando frontmatter YAML con el campo `paths`. Estas reglas condicionales solo se aplican cuando Claude trabaja con archivos que coinciden con los patrones especificados.

```
---
paths:
  - "src/api/**/*.ts"
---

## Reglas de desarrollo de API

- Todos los puntos finales de API deben incluir validación de entrada
- Usar el formato de respuesta de error estándar
- Incluir comentarios de documentación OpenAPI
```

Las reglas sin un campo `paths` se cargan incondicionalmente y se aplican a todos los archivos. Las reglas con alcance de ruta se activan cuando Claude lee archivos que coinciden con el patrón, no en cada uso de herramienta.

Use patrones glob en el campo `paths` para hacer coincidir archivos por extensión, directorio o cualquier combinación:

Patrón	Coincide
<code>**/*.ts</code>	Todos los archivos TypeScript en cualquier directorio
<code>src/**/*.*</code>	Todos los archivos bajo el directorio <code>src/</code>

Patrón	Coincide
<code>*.md</code>	Archivos Markdown en la raíz del proyecto
<code>src/components/*.tsx</code>	Componentes React en un directorio específico

Puede especificar múltiples patrones y usar expansión de llaves para hacer coincidir múltiples extensiones en un patrón:

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.{ts}"
  - "tests/**/*.{test.ts}"
---
```

Compartir reglas entre proyectos con enlaces simbólicos

El directorio `.claude/rules/` admite enlaces simbólicos, por lo que puede mantener un conjunto compartido de reglas y vincularlas en múltiples proyectos. Los enlaces simbólicos se resuelven y se cargan normalmente, y los enlaces simbólicos circulares se detectan y se manejan correctamente.

Este ejemplo vincula tanto un directorio compartido como un archivo individual:

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

Reglas a nivel de usuario

Las reglas personales en `~/.claude/rules/` se aplican a cada proyecto en su máquina. Úselas para preferencias que no son específicas del proyecto:

```
~/.claude/rules/
├─ preferences.md    # Sus preferencias personales de codificación
└─ workflows.md      # Sus flujos de trabajo preferidos
```

Las reglas a nivel de usuario se cargan antes que las reglas del proyecto, dando a las reglas del proyecto mayor prioridad.

Gestionar CLAUDE.md para equipos grandes

Para organizaciones que implementan Claude Code en equipos, puede centralizar instrucciones y controlar qué archivos CLAUDE.md se cargan.

Implementar CLAUDE.md en toda la organización

Las organizaciones pueden implementar un CLAUDE.md gestionado centralmente que se aplique a todos los usuarios en una máquina. Este archivo no puede ser excluido por configuraciones individuales.

Step 1: Crear el archivo en la ubicación de política gestionada

- macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md`
- Linux y WSL: `/etc/claude-code/CLAUDE.md`
- Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`

Step 2: Implementar con su sistema de gestión de configuración

Use MDM, Group Policy, Ansible o herramientas similares para distribuir el archivo en máquinas de desarrolladores. Consulte [configuración gestionada](#) para otras opciones de configuración de toda la organización.

Excluir archivos CLAUDE.md específicos

En monorepos grandes, los archivos CLAUDE.md ancestros pueden contener instrucciones que no son relevantes para su trabajo. La configuración `claudeMdExcludes` le permite omitir archivos específicos por ruta o patrón glob.

Este ejemplo excluye un CLAUDE.md de nivel superior y un directorio de reglas de una carpeta principal. Agréguelo a `.claude/settings.local.json` para que la exclusión permanezca local en su máquina:

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

Los patrones se comparan contra rutas de archivo absolutas usando sintaxis glob. Puede configurar `claudeMdExcludes` en cualquier [capa de configuración](#): usuario, proyecto, local o política gestionada. Los arreglos se fusionan entre capas.

Los archivos CLAUDE.md de política gestionada no pueden ser excluidos. Esto asegura que las instrucciones de toda la organización siempre se apliquen independientemente de la configuración individual.

Auto memory

Auto memory permite que Claude acumule conocimiento entre sesiones sin que usted escriba nada. Claude guarda notas para sí mismo mientras trabaja: comandos de compilación, información de depuración, notas de arquitectura, preferencias de estilo de código y hábitos de flujo de trabajo. Claude no guarda algo cada sesión. Decide qué vale la pena recordar basándose en si la información sería útil en una conversación futura.

Note:

Auto memory requiere Claude Code v2.1.59 o posterior. Verifique su versión con `claude --version`.

Habilitar o deshabilitar auto memory

Auto memory está habilitado de forma predeterminada. Para alternarlo, abra `/memory` en una sesión y use el alternador de auto memory, o establezca `autoMemoryEnabled` en la configuración de su proyecto:

```
{
  "autoMemoryEnabled": false
}
```

Para deshabilitar auto memory a través de variable de entorno, establezca

`CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`.

Ubicación de almacenamiento

Cada proyecto obtiene su propio directorio de memoria en `~/.claude/projects/<project>/memory/`. La ruta `<project>` se deriva del repositorio git, por lo que todos los worktrees y subdirectorios dentro del mismo repositorio comparten un directorio de auto memory. Fuera de un repositorio git, se usa la raíz del proyecto en su lugar.

Para almacenar auto memory en una ubicación diferente, establezca

`autoMemoryDirectory` en la configuración de usuario o local:

```
{
  "autoMemoryDirectory": "~/my-custom-memory-dir"
}
```

Esta configuración se acepta desde la política, local y configuración de usuario. No se acepta desde la configuración del proyecto (`.claude/settings.json`) para evitar que un proyecto compartido redirija escrituras de auto memory a ubicaciones sensibles.

El directorio contiene un punto de entrada `MEMORY.md` y archivos de tema opcionales:

```
~/ .claude/projects/<project>/memory/
├─ MEMORY.md           # Índice conciso, cargado en cada sesión
├─ debugging.md        # Notas detalladas sobre patrones de depuración
├─ api-conventions.md  # Decisiones de diseño de API
└─ ...                 # Cualquier otro archivo de tema que Claude cree
```

`MEMORY.md` actúa como un índice del directorio de memoria. Claude lee y escribe archivos en este directorio durante su sesión, usando `MEMORY.md` para realizar un seguimiento de lo que se almacena dónde.

Auto memory es local de la máquina. Todos los worktrees y subdirectorios dentro del mismo repositorio git comparten un directorio de auto memory. Los archivos no se comparten entre máquinas o entornos en la nube.

Cómo funciona

Las primeras 200 líneas de `MEMORY.md` se cargan al inicio de cada conversación. El contenido más allá de la línea 200 no se carga al inicio de la sesión. Claude mantiene `MEMORY.md` conciso moviendo notas detalladas a archivos de tema separados.

Este límite de 200 líneas se aplica solo a `MEMORY.md`. Los archivos `CLAUDE.md` se cargan completamente independientemente de la longitud, aunque los archivos más cortos producen mejor adherencia.

Los archivos de tema como `debugging.md` o `patterns.md` no se cargan al iniciar. Claude los lee bajo demanda usando sus herramientas de archivo estándar cuando necesita la información.

Claude lee y escribe archivos de memoria durante su sesión. Cuando ve “Writing memory” o “Recalled memory” en la interfaz de Claude Code, Claude está actualizando o leyendo activamente desde `~/ .claude/projects/<project>/memory/`.

Auditar y editar su memoria

Los archivos de auto memory son markdown plano que puede editar o eliminar en cualquier momento. Ejecute `/memory` para examinar y abrir archivos de memoria desde dentro de una sesión.

Ver y editar con `/memory`

El comando `/memory` enumera todos los archivos CLAUDE.md y rules cargados en su sesión actual, le permite alternar auto memory activado o desactivado, y proporciona un enlace para abrir la carpeta de auto memory. Seleccione cualquier archivo para abrirlo en su editor.

Cuando le pide a Claude que recuerde algo, como “siempre usar pnpm, no npm” o “recuerde que las pruebas de API requieren una instancia local de Redis”, Claude lo guarda en auto memory. Para agregar instrucciones a CLAUDE.md en su lugar, pídale a Claude directamente, como “agregue esto a CLAUDE.md”, o edite el archivo usted mismo a través de `/memory`.

Solucionar problemas de memoria

Estos son los problemas más comunes con CLAUDE.md y auto memory, junto con pasos para depurarlos.

Claude no está siguiendo mi CLAUDE.md

CLAUDE.md es contexto, no cumplimiento. Claude lo lee e intenta seguirlo, pero no hay garantía de cumplimiento estricto, especialmente para instrucciones vagas o conflictivas.

Para depurar:

- Ejecute `/memory` para verificar que sus archivos CLAUDE.md se están cargando. Si un archivo no aparece en la lista, Claude no puede verlo.
- Verifique que el CLAUDE.md relevante esté en una ubicación que se cargue para su sesión (consulte [Elija dónde colocar los archivos CLAUDE.md](#)).
- Haga que las instrucciones sean más específicas. “Usar indentación de 2 espacios” funciona mejor que “formatear código bien”.
- Busque instrucciones conflictivas en archivos CLAUDE.md. Si dos archivos dan orientación diferente para el mismo comportamiento, Claude puede elegir uno arbitrariamente.

Tip:

Use el `hook` `InstructionsLoaded` para registrar exactamente qué archivos de instrucciones se cargan, cuándo se cargan y por qué. Esto es útil para depurar reglas específicas de ruta o archivos cargados de forma perezosa en subdirectorios.

No sé qué guardó auto memory

Ejecute `/memory` y seleccione la carpeta de auto memory para examinar lo que Claude ha guardado. Todo es markdown plano que puede leer, editar o eliminar.

Mi CLAUDE.md es demasiado grande

Los archivos de más de 200 líneas consumen más contexto y pueden reducir la adherencia. Mueva contenido detallado a archivos separados referenciados con importaciones `@path` (consulte [Importar archivos adicionales](#)), o divida sus instrucciones en archivos `.claude/rules/`.

Las instrucciones parecen perdidas después de `/compact`

CLAUDE.md sobrevive completamente a la compactación. Después de `/compact`, Claude vuelve a leer su CLAUDE.md desde el disco e lo reinyecta fresco en la sesión. Si una instrucción desapareció después de la compactación, se dio solo en la conversación, no se escribió en CLAUDE.md. Agréguelo a CLAUDE.md para que persista entre sesiones.

Consulte [Escriba instrucciones efectivas](#) para obtener orientación sobre tamaño, estructura y especificidad.

Recursos relacionados

- [Skills](#): empaquetar flujos de trabajo repetibles que se cargan bajo demanda
- [Settings](#): configurar el comportamiento de Claude Code con archivos de configuración
- [Manage sessions](#): gestionar contexto, reanudar conversaciones y ejecutar sesiones paralelas
- [Subagent memory](#): permitir que los subagents mantengan su propia auto memory

Configuración de Claude Code

Configure Claude Code con configuraciones globales y a nivel de proyecto, y variables de entorno.

Claude Code ofrece una variedad de configuraciones para personalizar su comportamiento según sus necesidades. Puede configurar Claude Code ejecutando el comando `/config` cuando utiliza el REPL interactivo, que abre una interfaz de Configuración con pestañas donde puede ver información de estado y modificar opciones de configuración.

Alcances de configuración

Claude Code utiliza un **sistema de alcances** para determinar dónde se aplican las configuraciones y quién las comparte. Comprender los alcances le ayuda a decidir cómo configurar Claude Code para uso personal, colaboración en equipo o implementación empresarial.

Alcances disponibles

Alcance	Ubicación	A quién afecta	¿Se comparte con el equipo?
Managed	Configuraciones administradas por servidor, plist / registro, o <code>managed-settings.json</code> a nivel de sistema	Todos los usuarios en la máquina	Sí (implementado por TI)
User	Directorio <code>~/.claude/</code>	Usted, en todos los proyectos	No
Project	<code>.claude/</code> en el repositorio	Todos los colaboradores en este repositorio	Sí (confirmado en git)
Local	<code>.claude/settings.local.json</code>	Usted, solo en este repositorio	No (ignorado por git)

Cuándo usar cada alcance

El alcance **Managed** es para:

- Políticas de seguridad que deben aplicarse en toda la organización

- Requisitos de cumplimiento que no se pueden anular
- Configuraciones estandarizadas implementadas por TI/DevOps

El alcance **User** es mejor para:

- Preferencias personales que desea en todas partes (temas, configuraciones del editor)
- Herramientas y plugins que utiliza en todos los proyectos
- Claves API y autenticación (almacenadas de forma segura)

El alcance **Project** es mejor para:

- Configuraciones compartidas por el equipo (permisos, hooks, MCP servers)
- Plugins que todo el equipo debe tener
- Estandarización de herramientas entre colaboradores

El alcance **Local** es mejor para:

- Anulaciones personales para un proyecto específico
- Configuraciones de prueba antes de compartir con el equipo
- Configuraciones específicas de la máquina que no funcionarán para otros

Cómo interactúan los alcances

Cuando la misma configuración se configura en múltiples alcances, los alcances más específicos tienen prioridad:

1. **Managed** (más alto) - no puede ser anulado por nada
2. **Argumentos de línea de comandos** - anulaciones de sesión temporal
3. **Local** - anula configuraciones de proyecto y usuario
4. **Project** - anula configuraciones de usuario
5. **User** (más bajo) - se aplica cuando nada más especifica la configuración

Por ejemplo, si un permiso se permite en la configuración de usuario pero se deniega en la configuración del proyecto, la configuración del proyecto tiene prioridad y el permiso se bloquea.

Qué usa alcances

Los alcances se aplican a muchas características de Claude Code:

Característica	Ubicación de usuario	Ubicación de proyecto	Ubicación local
Settings	<code>~/.claude/settings.json</code>	<code>.claude/settings.json</code>	<code>.claude/settings.local.json</code>
Subagents	<code>~/.claude/agents/</code>	<code>.claude/agents/</code>	—
MCP servers	<code>~/.claude.json</code>	<code>.mcp.json</code>	<code>~/.claude.json</code> (por proyecto)
Plugins	<code>~/.claude/settings.json</code>	<code>.claude/settings.json</code>	<code>.claude/settings.local.json</code>
CLAUDE.md	<code>~/.claude/CLAUDE.md</code>	<code>CLAUDE.md</code> o <code>.claude/CLAUDE.md</code>	—

Archivos de configuración

El archivo `settings.json` es nuestro mecanismo oficial para configurar Claude Code a través de configuraciones jerárquicas:

- **Configuraciones de usuario** se definen en `~/.claude/settings.json` y se aplican a todos los proyectos.
- **Configuraciones de proyecto** se guardan en su directorio de proyecto:
 - `.claude/settings.json` para configuraciones que se verifican en el control de código fuente y se comparten con su equipo
 - `.claude/settings.local.json` para configuraciones que no se verifican, útiles para preferencias personales y experimentación. Claude Code configurará git para ignorar `.claude/settings.local.json` cuando se cree.
- **Configuraciones administradas:** Para organizaciones que necesitan control centralizado, Claude Code admite múltiples mecanismos de entrega para configuraciones

administradas. Todos utilizan el mismo formato JSON y no pueden ser anulados por configuraciones de usuario o proyecto:

- **Configuraciones administradas por servidor:** entregadas desde los servidores de Anthropic a través de la consola de administración de Claude.ai. Consulte [configuraciones administradas por servidor](#).
- **Políticas de MDM/nivel de SO:** entregadas a través de administración de dispositivos nativa en macOS y Windows:
 - macOS: dominio de preferencias administradas `com.anthropic.claudecode` (implementado a través de perfiles de configuración en Jamf, Kandji u otras herramientas MDM)
 - Windows: clave de registro `HKLM\SOFTWARE\Policies\ClaudeCode` con un valor `Settings` (REG_SZ o REG_EXPAND_SZ) que contiene JSON (implementado a través de Política de Grupo o Intune)
 - Windows (nivel de usuario): `HKCU\SOFTWARE\Policies\ClaudeCode` (prioridad de política más baja, solo se usa cuando no existe una fuente a nivel de administrador)
- **Basado en archivos:** `managed-settings.json` y `managed-mcp.json` implementados en directorios del sistema:
 - macOS: `/Library/Application Support/ClaudeCode/`
 - Linux y WSL: `/etc/claude-code/`
 - Windows: `C:\Program Files\ClaudeCode\`

Consulte [configuraciones administradas](#) y [Configuración de MCP administrada](#) para obtener detalles.

Note:

Las implementaciones administradas también pueden restringir **adiciones de marketplace de plugins** usando `strictKnownMarketplaces`. Para obtener más información, consulte [Restricciones de marketplace administradas](#).

- **Otra configuración** se almacena en `~/.claude.json`. Este archivo contiene sus preferencias (tema, configuraciones de notificación, modo de editor), sesión OAuth, configuraciones de [MCP server](#) para alcances de usuario y local, estado por proyecto (herramientas permitidas, configuraciones de confianza) y varios cachés. Los MCP servers con alcance de proyecto se almacenan por separado en `.mcp.json`.

Note:

Claude Code crea automáticamente copias de seguridad con marca de tiempo de archivos de configuración y retiene las cinco copias de seguridad más recientes para prevenir pérdida de datos.

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test *)",
      "Read(~/.zshrc)"
    ],
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp"
  },
  "companyAnnouncements": [
    "Welcome to Acme Corp! Review our code guidelines at docs.acme.com",
    "Reminder: Code reviews required for all PRs",
    "New security policy in effect"
  ]
}
```

La línea `$schema` en el ejemplo anterior apunta al [esquema JSON oficial](https://json.schemastore.org/claude-code-settings.json) para configuraciones de Claude Code. Agregarlo a su `settings.json` habilita autocompletado y validación en línea en VS Code, Cursor y cualquier otro editor que admita validación de esquema JSON.

Configuraciones disponibles

`settings.json` admite varias opciones:

Clave	Descripción	Ejemplo
<code>apiKeyHelper</code>	Script personalizado, a ejecutarse en <code>/bin/sh</code> , para generar un valor de autenticación. Este valor se enviará como encabezados <code>X-Api-Key</code> y <code>Authorization: Bearer</code> para solicitudes de modelo	<code>/bin/ generate_temp_api_key .sh</code>
<code>cleanupPeriodDays</code>	Las sesiones inactivas durante más tiempo que este período se eliminan al inicio. Establecer en <code>0</code> elimina inmediatamente todas las sesiones. (predeterminado: 30 días)	<code>20</code>
<code>companyAnnouncements</code>	Anuncio a mostrar a los usuarios al inicio. Si se proporcionan múltiples anuncios, se alternarán aleatoriamente.	<code>["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]</code>
<code>env</code>	Variables de entorno que se aplicarán a cada sesión	<code>{"F00": "bar"}</code>
<code>attribution</code>	Personalizar atribución para commits de git y solicitudes de extracción. Consulte Configuración de atribución	<code>{"commit": " Generated with Claude Code", "pr": ""}</code>
<code>includeCoAuthoredBy</code>	Obsoleto: Use <code>attribution</code> en su lugar. Si incluir la línea <code>co-authored-by Claude</code> en commits de git y solicitudes de extracción (predeterminado: <code>true</code>)	<code>false</code>
<code>includeGitInstructions</code>	Incluir instrucciones de flujo de trabajo de commit y PR integradas en el indicador del sistema de Claude (predeterminado: <code>true</code>). Establezca en <code>false</code> para eliminar estas instrucciones, por ejemplo cuando utiliza sus propias skills de flujo de trabajo de git. La variable de entorno <code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code> tiene prioridad sobre esta configuración cuando se establece	<code>false</code>

Clave	Descripción	Ejemplo
<code>permissions</code>	Consulte la tabla a continuación para la estructura de permisos.	
<code>hooks</code>	Configurar comandos personalizados para ejecutarse en eventos del ciclo de vida. Consulte documentación de hooks para el formato	Consulte hooks
<code>disableAllHooks</code>	Deshabilitar todos los hooks y cualquier línea de estado personalizada	<code>true</code>
<code>allowManagedHooksOnly</code>	(Solo configuraciones administradas) Prevenir la carga de hooks de usuario, proyecto y plugin. Solo permite hooks administrados y hooks SDK. Consulte Configuración de hooks	<code>true</code>
<code>allowedHttpHookUrls</code>	Lista blanca de patrones de URL que los hooks HTTP pueden dirigirse. Admite <code>*</code> como comodín. Cuando se establece, los hooks con URLs que no coinciden se bloquean. Sin definir = sin restricción, matriz vacía = bloquear todos los hooks HTTP. Las matrices se fusionan entre fuentes de configuración. Consulte Configuración de hooks	<code>["https://hooks.example.com/*"]</code>
<code>httpHookAllowedEnvVars</code>	Lista blanca de nombres de variables de entorno que los hooks HTTP pueden interpolar en encabezados. Cuando se establece, el <code>allowedEnvVars</code> efectivo de cada hook es la intersección con esta lista. Sin definir = sin restricción. Las matrices se fusionan entre fuentes de configuración. Consulte Configuración de hooks	<code>["MY_TOKEN", "HOOK_SECRET"]</code>

Clave	Descripción	Ejemplo
<code>allowManagedPermissionsRulesOnly</code>	(Solo configuraciones administradas) Prevenir que configuraciones de usuario y proyecto definan reglas de permiso <code>allow</code> , <code>ask</code> o <code>deny</code> . Solo se aplican las reglas en configuraciones administradas. Consulte Configuraciones solo administradas	<code>true</code>
<code>allowManagedMcpServersOnly</code>	(Solo configuraciones administradas) Solo se respetan <code>allowedMcpServers</code> de configuraciones administradas. <code>deniedMcpServers</code> aún se fusiona de todas las fuentes. Los usuarios aún pueden agregar MCP servers, pero solo se aplica la lista blanca definida por el administrador. Consulte Configuración de MCP administrada	<code>true</code>
<code>model</code>	Anular el modelo predeterminado a usar para Claude Code	<code>"claude-sonnet-4-6"</code>
<code>availableModels</code>	Restringir qué modelos pueden seleccionar los usuarios a través de <code>/model</code> , <code>--model</code> , herramienta Config o <code>ANTHROPIC_MODEL</code> . No afecta la opción Predeterminado. Consulte Restringir selección de modelo	<code>["sonnet", "haiku"]</code>
<code>modelOverrides</code>	Asignar IDs de modelo de Anthropic a IDs de modelo específicos del proveedor como ARNs de perfil de inferencia de Bedrock. Cada entrada del selector de modelo utiliza su valor asignado al llamar a la API del proveedor. Consulte Anular IDs de modelo por versión	<pre>{"claude-opus-4-6": "arn:aws:bedrock:..." }</pre>
<code>otelHeadersHelper</code>	Script para generar encabezados dinámicos de OpenTelemetry. Se ejecuta al inicio y periódicamente (consulte Encabezados dinámicos)	<code>/bin/ generate_otel_headers .sh</code>

Clave	Descripción	Ejemplo
<code>statusLine</code>	Configurar una línea de estado personalizada para mostrar contexto. Consulte documentación de statusLine	<pre>{"type": "command", "command": "~/ .claude/ statusline.sh"}</pre>
<code>fileSuggestion</code>	Configurar un script personalizado para autocompletado de archivos <code>@</code> . Consulte Configuración de sugerencia de archivos	<pre>{"type": "command", "command": "~/ .claude/file- suggestion.sh"}</pre>
<code>respectGitignore</code>	Controlar si el selector de archivos <code>@</code> respeta patrones <code>.gitignore</code> . Cuando es <code>true</code> (predeterminado), los archivos que coinciden con patrones <code>.gitignore</code> se excluyen de las sugerencias	<code>false</code>
<code>outputStyle</code>	Configurar un estilo de salida para ajustar el indicador del sistema. Consulte documentación de estilos de salida	<code>"Explanatory"</code>
<code>forceLoginMethod</code>	Use <code>claudeai</code> para restringir el inicio de sesión a cuentas de Claude.ai, <code>console</code> para restringir el inicio de sesión a cuentas de Claude Console (facturación de uso de API)	<code>claudeai</code>
<code>forceLoginOrgUUID</code>	Especificar el UUID de una organización para seleccionarla automáticamente durante el inicio de sesión, omitiendo el paso de selección de organización. Requiere que <code>forceLoginMethod</code> esté establecido	<code>"xxxxxxxx-xxxx-xxxx- xxxx-xxxxxxxxxxxx"</code>
<code>enableAllProjectMcpServers</code>	Aprobar automáticamente todos los MCP servers definidos en archivos <code>.mcp.json</code> del proyecto	<code>true</code>
<code>enabledMcpjsonServers</code>	Lista de MCP servers específicos de archivos <code>.mcp.json</code> para aprobar	<code>["memory", "github"]</code>

Clave	Descripción	Ejemplo
<code>disabledMcpjsonServers</code>	Lista de MCP servers específicos de archivos <code>.mcp.json</code> para rechazar	<code>["filesystem"]</code>
<code>allowedMcpServers</code>	Cuando se establece en <code>managed-settings.json</code> , lista blanca de MCP servers que los usuarios pueden configurar. Sin definir = sin restricciones, matriz vacía = bloqueo. Se aplica a todos los alcances. La lista de denegación tiene prioridad. Consulte Configuración de MCP administrada	<code>{ "serverName": "github" }</code>
<code>deniedMcpServers</code>	Cuando se establece en <code>managed-settings.json</code> , lista de denegación de MCP servers que se bloquean explícitamente. Se aplica a todos los alcances incluyendo servidores administrados. La lista de denegación tiene prioridad sobre la lista blanca. Consulte Configuración de MCP administrada	<code>{ "serverName": "filesystem" }</code>
<code>strictKnownMarketplaces</code>	Cuando se establece en <code>managed-settings.json</code> , lista blanca de marketplaces de plugins que los usuarios pueden agregar. Sin definir = sin restricciones, matriz vacía = bloqueo. Se aplica solo a adiciones de marketplace. Consulte Restricciones de marketplace administradas	<code>{ "source": "github", "repo": "acme-corp/plugins" }</code>
<code>blockedMarketplaces</code>	(Solo configuraciones administradas) Lista de bloqueo de fuentes de marketplace. Las fuentes bloqueadas se verifican antes de descargar, por lo que nunca tocan el sistema de archivos. Consulte Restricciones de marketplace administradas	<code>{ "source": "github", "repo": "untrusted/plugins" }</code>

Clave	Descripción	Ejemplo
<code>pluginTrustMessage</code>	(Solo configuraciones administradas) Mensaje personalizado agregado a la advertencia de confianza de plugin mostrada antes de la instalación. Utilice esto para agregar contexto específico de la organización, por ejemplo para confirmar que los plugins de su marketplace interno están verificados.	<code>"All plugins from our marketplace are approved by IT"</code>
<code>awsAuthRefresh</code>	Script personalizado que modifica el directorio <code>.aws</code> (consulte configuración de credenciales avanzada)	<code>aws sso login --profile myprofile</code>
<code>awsCredentialExport</code>	Script personalizado que genera JSON con credenciales de AWS (consulte configuración de credenciales avanzada)	<code>/bin/generate_aws_grant.sh</code>
<code>alwaysThinkingEnabled</code>	Habilitar pensamiento extendido de forma predeterminada para todas las sesiones. Típicamente configurado a través del comando <code>/config</code> en lugar de editar directamente	<code>true</code>
<code>plansDirectory</code>	Personalizar dónde se almacenan los archivos de plan. La ruta es relativa a la raíz del proyecto. Predeterminado: <code>~/ .claude/plans</code>	<code>"./plans"</code>
<code>showTurnDuration</code>	Mostrar mensajes de duración de turno después de respuestas (por ejemplo, "Cooked for 1m 6s"). Establezca en <code>false</code> para ocultar estos mensajes	<code>true</code>
<code>spinnerVerbs</code>	Personalizar los verbos de acción mostrados en el spinner y mensajes de duración de turno. Establezca <code>mode</code> en <code>"replace"</code> para usar solo sus verbos, o <code>"append"</code> para agregarlos a los predeterminados	<code>{"mode": "append", "verbs": ["Pondering", "Crafting"]}</code>

Clave	Descripción	Ejemplo
<code>language</code>	Configurar el idioma de respuesta preferido de Claude (por ejemplo, <code>"japanese"</code> , <code>"spanish"</code> , <code>"french"</code>). Claude responderá en este idioma de forma predeterminada	<code>"japanese"</code>
<code>autoUpdatesChannel</code>	Canal de lanzamiento a seguir para actualizaciones. Use <code>"stable"</code> para una versión que típicamente tiene aproximadamente una semana de antigüedad y omite versiones con regresiones importantes, o <code>"latest"</code> (predeterminado) para el lanzamiento más reciente	<code>"stable"</code>
<code>spinnerTipsEnabled</code>	Mostrar consejos en el spinner mientras Claude está trabajando. Establezca en <code>false</code> para deshabilitar consejos (predeterminado: <code>true</code>)	<code>false</code>
<code>spinnerTipsOverride</code>	Anular consejos del spinner con cadenas personalizadas. <code>tips</code> : matriz de cadenas de consejo. <code>excludeDefault</code> : si es <code>true</code> , mostrar solo consejos personalizados; si es <code>false</code> o está ausente, los consejos personalizados se fusionan con consejos integrados	<pre>{ "excludeDefault": true, "tips": ["Use our internal tool X"] }</pre>
<code>terminalProgressBarEnabled</code>	Habilitar la barra de progreso del terminal que muestra progreso en terminales compatibles como Windows Terminal e iTerm2 (predeterminado: <code>true</code>)	<code>false</code>
<code>prefersReducedMotion</code>	Reducir o deshabilitar animaciones de UI (spinners, shimmer, efectos flash) para accesibilidad	<code>true</code>

Clave	Descripción	Ejemplo
<code>fastModePerSessionOptIn</code>	Cuando es <code>true</code> , el modo rápido no persiste entre sesiones. Cada sesión comienza con el modo rápido desactivado, requiriendo que los usuarios lo habiliten con <code>/fast</code> . La preferencia de modo rápido del usuario aún se guarda. Consulte Requerir opt-in por sesión	<code>true</code>
<code>teammateMode</code>	Cómo se muestran los compañeros del equipo de agentes : <code>auto</code> (elige paneles divididos en tmux o iTerm2, en proceso de otro modo), <code>in-process</code> , o <code>tmux</code> . Consulte configurar equipos de agentes	<code>"in-process"</code>

Configuración de permisos

Claves	Descripción	Ejemplo
<code>allow</code>	Matriz de reglas de permiso para permitir el uso de herramientas. Consulte Sintaxis de regla de permiso a continuación para detalles de coincidencia de patrones	<code>["Bash(git diff *)"]</code>
<code>ask</code>	Matriz de reglas de permiso para pedir confirmación al usar herramientas. Consulte Sintaxis de regla de permiso a continuación	<code>["Bash(git push *)"]</code>
<code>deny</code>	Matriz de reglas de permiso para denegar el uso de herramientas. Use esto para excluir archivos sensibles del acceso de Claude Code. Consulte Sintaxis de regla de permiso y Limitaciones de permiso de Bash	<code>["WebFetch", "Bash(curl *)", "Read(../env)", "Read(../secrets/**)"]</code>
<code>additionalDirectories</code>	Directorios de trabajo adicionales a los que Claude tiene acceso	<code>["../docs/"]</code>

Claves	Descripción	Ejemplo
<code>defaultMode</code>	Modo de permiso predeterminado al abrir Claude Code	<code>"acceptEdits"</code>
<code>disableBypassPermissionsMode</code>	Establezca en <code>"disable"</code> para prevenir que el modo <code>bypassPermissions</code> sea activado. Esto deshabilita la bandera de línea de comandos <code>--dangerously-skip-permissions</code> . Consulte configuraciones administradas	<code>"disable"</code>

Sintaxis de regla de permiso

Las reglas de permiso siguen el formato `Tool` o `Tool(specifier)`. Las reglas se evalúan en orden: primero reglas de denegación, luego preguntar, luego permitir. La primera regla coincidente gana.

Ejemplos rápidos:

Regla	Efecto
<code>Bash</code>	Coincide con todos los comandos Bash
<code>Bash(npm run *)</code>	Coincide con comandos que comienzan con <code>npm run</code>
<code>Read(./env)</code>	Coincide con la lectura del archivo <code>.env</code>
<code>WebFetch(domain:example.com)</code>	Coincide con solicitudes de búsqueda a <code>example.com</code>

Para la referencia completa de sintaxis de regla, incluyendo comportamiento de comodín, patrones específicos de herramientas para Read, Edit, WebFetch, MCP y reglas de Agent, y limitaciones de seguridad de patrones de Bash, consulte [Sintaxis de regla de permiso](#).

Configuración de sandbox

Configure el comportamiento avanzado de sandboxing. El sandboxing aísla comandos bash de su sistema de archivos y red. Consulte [Sandboxing](#) para obtener detalles.

Claves	Descripción	Ejemplo
<code>enabled</code>	Habilitar sandboxing de bash (macOS, Linux y WSL2). Predeterminado: false	<code>true</code>
<code>autoAllowBashIfSandboxed</code>	Aprobar automáticamente comandos bash cuando están en sandbox. Predeterminado: true	<code>true</code>
<code>excludedCommands</code>	Comandos que deben ejecutarse fuera del sandbox	<code>["git", "docker"]</code>
<code>allowUnsandboxedCommands</code>	Permitir que los comandos se ejecuten fuera del sandbox a través del parámetro <code>dangerouslyDisableSandbox</code> . Cuando se establece en <code>false</code> , la escotilla de escape <code>dangerouslyDisableSandbox</code> se deshabilita completamente y todos los comandos deben ejecutarse en sandbox (o estar en <code>excludedCommands</code>). Útil para políticas empresariales que requieren sandboxing estricto. Predeterminado: true	<code>false</code>
<code>filesystem.allowWrite</code>	Rutas adicionales donde los comandos en sandbox pueden escribir. Las matrices se fusionan en todos los alcances de configuración: las rutas de usuario, proyecto y administradas se combinan, no se reemplazan. También se fusionan con rutas de reglas de permiso <code>Edit(...)</code> <code>allow</code> . Consulte prefijos de ruta de sandbox a continuación.	<code>["//tmp/build", "~/.kube"]</code>

Claves	Descripción	Ejemplo
<code>filesystem.denyWrite</code>	Rutas donde los comandos en sandbox no pueden escribir. Las matrices se fusionan en todos los alcances de configuración. También se fusionan con rutas de reglas de permiso <code>Edit(...)</code> deny.	<code>["//etc", "//usr/ local/bin"]</code>
<code>filesystem.denyRead</code>	Rutas donde los comandos en sandbox no pueden leer. Las matrices se fusionan en todos los alcances de configuración. También se fusionan con rutas de reglas de permiso <code>Read(...)</code> deny.	<code>["~/.aws/ credentials"]</code>
<code>network.allowUnixSockets</code>	Rutas de socket Unix accesibles en sandbox (para agentes SSH, etc.)	<code>["~/.ssh/ agent- socket"]</code>
<code>network.allowAllUnixSockets</code>	Permitir todas las conexiones de socket Unix en sandbox. Predeterminado: false	<code>true</code>
<code>network.allowLocalBinding</code>	Permitir vinculación a puertos localhost (solo macOS). Predeterminado: false	<code>true</code>
<code>network.allowedDomains</code>	Matriz de dominios para permitir tráfico de red saliente. Admite comodines (por ejemplo, <code>*.example.com</code>).	<code>["github.com", "*.npmjs.org"]</code>
<code>network.allowManagedDomainsOnly</code>	(Solo configuraciones administradas) Solo se respetan <code>allowedDomains</code> y reglas <code>allowWebFetch(domain: ...)</code> de configuraciones administradas. Los dominios de configuraciones de usuario, proyecto y local se ignoran. Los dominios no permitidos se bloquean automáticamente sin solicitar al usuario. Los dominios denegados aún se respetan de todas las fuentes. Predeterminado: false	<code>true</code>

Claves	Descripción	Ejemplo
<code>network.httpProxyPort</code>	Puerto de proxy HTTP usado si desea traer su propio proxy. Si no se especifica, Claude ejecutará su propio proxy.	<code>8080</code>
<code>network.socksProxyPort</code>	Puerto de proxy SOCKS5 usado si desea traer su propio proxy. Si no se especifica, Claude ejecutará su propio proxy.	<code>8081</code>
<code>enableWeakerNestedSandbox</code>	Habilitar sandbox más débil para entornos Docker sin privilegios (solo Linux y WSL2). Reduce la seguridad. Predeterminado: false	<code>true</code>
<code>enableWeakerNetworkIsolation</code>	(Solo macOS) Permitir acceso al servicio de confianza TLS del sistema (<code>com.apple.trustd.agent</code>) en el sandbox. Requerido para herramientas basadas en Go como <code>gh</code> , <code>gcloud</code> y <code>terraform</code> para verificar certificados TLS cuando se usa <code>httpProxyPort</code> con un proxy MITM y CA personalizada. Reduce la seguridad al abrir una posible ruta de exfiltración de datos. Predeterminado: false	<code>true</code>

Prefijos de ruta de sandbox

Las rutas en `filesystem.allowWrite`, `filesystem.denyWrite` y `filesystem.denyRead` admiten estos prefijos:

Prefijo	Significado	Ejemplo
<code>//</code>	Ruta absoluta desde la raíz del sistema de archivos	<code>//tmp/build</code> se convierte en <code>/tmp/build</code>
<code>~/</code>	Relativo al directorio de inicio	<code>~/kube</code> se convierte en <code>\$HOME/.kube</code>

Prefijo	Significado	Ejemplo
/	Relativo al directorio del archivo de configuración	/build se convierte en \$ SETTINGS_DIR/build
./ o sin prefijo	Ruta relativa (resuelta por el tiempo de ejecución del sandbox)	./output

Ejemplo de configuración:

```
{
  "sandbox": {
    "enabled": true,
    "autoAllowBashIfSandboxed": true,
    "excludedCommands": ["docker"],
    "filesystem": {
      "allowWrite": ["/tmp/build", "~/kube"],
      "denyRead": ["/.aws/credentials"]
    },
  },
  "network": {
    "allowedDomains": ["github.com", "*.npmjs.org", "registry.yarnpkg.com"],
    "allowUnixSockets": [
      "/var/run/docker.sock"
    ],
    "allowLocalBinding": true
  }
}
```

Restricciones de sistema de archivos y red pueden configurarse de dos formas que se fusionan:

- **Configuraciones `sandbox.filesystem`** (mostradas arriba): Controlan rutas en el límite del sandbox a nivel de SO. Estas restricciones se aplican a todos los comandos de subproceso (por ejemplo, `kubectl`, `terraform`, `npm`), no solo a las herramientas de archivo de Claude.
- **Reglas de permiso:** Use reglas `allow/deny` `Edit` para controlar el acceso a la herramienta de archivo de Claude, reglas `deny` `Read` para bloquear lecturas, y reglas `allow/deny` `WebFetch` para controlar dominios de red. Las rutas de estas reglas también se fusionan en la configuración del sandbox.

Configuración de atribución

Claude Code agrega atribución a commits de git y solicitudes de extracción. Estos se configuran por separado:

- Los commits usan [git trailers](#) (como `Co-Authored-By`) de forma predeterminada, que pueden personalizarse o deshabilitarse
- Las descripciones de solicitudes de extracción son texto sin formato

Claves	Descripción
<code>commit</code>	Atribución para commits de git, incluyendo cualquier trailer. La cadena vacía oculta la atribución de commit
<code>pr</code>	Atribución para descripciones de solicitudes de extracción. La cadena vacía oculta la atribución de solicitud de extracción

Atribución de commit predeterminada:

```
 Generated with [Claude Code](https://claude.com/claude-code)

Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>
```

Atribución de solicitud de extracción predeterminada:

```
 Generated with [Claude Code](https://claude.com/claude-code)
```

Ejemplo:

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

Note:

La configuración `attribution` tiene prioridad sobre la configuración `includeCoAuthoredBy` obsoleta. Para ocultar toda la atribución, establezca `commit` y `pr` en cadenas vacías.

Configuración de sugerencia de archivos

Configure un comando personalizado para autocompletado de ruta de archivo `@`. La sugerencia de archivo integrada utiliza recorrido rápido del sistema de archivos, pero los monorepos grandes pueden beneficiarse de indexación específica del proyecto como un índice de archivo precompilado o herramientas personalizadas.

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/ .claude/file-suggestion.sh"
  }
}
```

El comando se ejecuta con las mismas variables de entorno que [hooks](#), incluyendo `CLAUDE_PROJECT_DIR`. Recibe JSON a través de stdin con un campo `query`:

```
{"query": "src/comp"}
```

Genere rutas de archivo separadas por saltos de línea a stdout (actualmente limitado a 15):

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

Ejemplo:

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```


Configuración de hooks

Estas configuraciones controlan qué hooks se pueden ejecutar y a qué pueden acceder los hooks HTTP. La configuración `allowManagedHooksOnly` solo se puede configurar en [configuraciones administradas](#). Las listas blancas de URL y variable de entorno se pueden establecer en cualquier nivel de configuración y se fusionan entre fuentes.

Comportamiento cuando `allowManagedHooksOnly` es `true`:

- Se cargan hooks administrados y hooks SDK
- Se bloquean hooks de usuario, proyecto y plugin

Restringir URLs de hooks HTTP:

Limitar qué URLs pueden dirigirse los hooks HTTP. Admite `*` como comodín para coincidencia. Cuando la matriz se define, los hooks HTTP que se dirigen a URLs que no coinciden se bloquean silenciosamente.

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

Restringir variables de entorno de hooks HTTP:

Limitar qué nombres de variables de entorno los hooks HTTP pueden interpolar en valores de encabezado. El `allowedEnvVars` efectivo de cada hook es la intersección de su propia lista y esta configuración.

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

Precedencia de configuración

Las configuraciones se aplican en orden de precedencia. De mayor a menor:

1. **Configuraciones administradas** ([administradas por servidor](#), [políticas de MDM/nivel de SO](#), o [configuraciones administradas](#))
 - Políticas implementadas por TI a través de entrega de servidor, perfiles de configuración MDM, políticas de registro o archivos de configuración administrados
 - No pueden ser anuladas por ningún otro nivel, incluyendo argumentos de línea de comandos

- Dentro del nivel administrado, la precedencia es: administrado por servidor > políticas de MDM/nivel de SO > `managed-settings.json` > registro HKCU (solo Windows). Solo se usa una fuente administrada; las fuentes no se fusionan.

2. Argumentos de línea de comandos

- Anulaciones temporales para una sesión específica

3. Configuraciones de proyecto local (`.claude/settings.local.json`)

- Configuraciones personales específicas del proyecto

4. Configuraciones de proyecto compartidas (`.claude/settings.json`)

- Configuraciones de proyecto compartidas por el equipo en control de código fuente

5. Configuraciones de usuario (`~/.claude/settings.json`)

- Configuraciones globales personales

Esta jerarquía asegura que las políticas organizacionales siempre se apliquen mientras aún permite que equipos e individuos personalicen su experiencia.

Por ejemplo, si su configuración de usuario permite `Bash(npm run *)` pero la configuración compartida de un proyecto la deniega, la configuración del proyecto tiene prioridad y el comando se bloquea.

Note:

Las configuraciones de matriz se fusionan entre alcances. Cuando la misma configuración con valor de matriz (como `sandbox.filesystem.allowWrite` o `permissions.allow`) aparece en múltiples alcances, las matrices se **concatenan y se deduplicán**, no se reemplazan. Esto significa que los alcances de menor prioridad pueden agregar entradas sin anular las establecidas por alcances de mayor prioridad, y viceversa. Por ejemplo, si las configuraciones administradas establecen `allowWrite` en `["//opt/company-tools"]` y un usuario agrega `["~/.kube"]`, ambas rutas se incluyen en la configuración final.

Verificar configuraciones activas

Ejecute `/status` dentro de Claude Code para ver qué fuentes de configuración están activas y de dónde provienen. La salida muestra cada capa de configuración (administrada, usuario, proyecto) junto con su origen, como `Enterprise managed settings (remote)`, `Enterprise managed settings (plist)`, `Enterprise managed settings (HKLM)`, o `Enterprise managed settings (file)`. Si un archivo de configuración contiene errores, `/status` reporta el problema para que pueda corregirlo.

Puntos clave sobre el sistema de configuración

- **Archivos de memoria (`CLAUDE.md`)**: Contienen instrucciones y contexto que Claude carga al inicio
- **Archivos de configuración (JSON)**: Configurar permisos, variables de entorno y comportamiento de herramientas
- **Skills**: Indicaciones personalizadas que pueden invocarse con `/skill-name` o cargarse automáticamente por Claude
- **MCP servers**: Extender Claude Code con herramientas e integraciones adicionales
- **Precedencia**: Las configuraciones de nivel superior (Managed) anulan las de nivel inferior (User/Project)
- **Herencia**: Las configuraciones se fusionan, con configuraciones más específicas agregando o anulando las más amplias

Indicador del sistema

El indicador del sistema interno de Claude Code no se publica. Para agregar instrucciones personalizadas, use archivos `CLAUDE.md` o la bandera `--append-system-prompt`.

Excluyendo archivos sensibles

Para prevenir que Claude Code acceda a archivos que contienen información sensible como claves API, secretos y archivos de entorno, use la configuración `permissions.deny` en su archivo `.claude/settings.json`:

```
{
  "permissions": {
    "deny": [
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

Esto reemplaza la configuración `ignorePatterns` obsoleta. Los archivos que coinciden con estos patrones se excluyen del descubrimiento de archivos y resultados de búsqueda, y se deniegan las operaciones de lectura en estos archivos.

Configuración de subagent

Claude Code admite subagents de IA personalizados que pueden configurarse en niveles de usuario y proyecto. Estos subagents se almacenan como archivos Markdown con frontmatter YAML:

- **Subagents de usuario:** `~/.claude/agents/` - Disponibles en todos sus proyectos
- **Subagents de proyecto:** `.claude/agents/` - Específicos de su proyecto y pueden compartirse con su equipo

Los archivos de subagent definen asistentes de IA especializados con indicaciones personalizadas y permisos de herramientas. Obtenga más información sobre cómo crear y usar subagents en la [documentación de subagents](#).

Configuración de plugins

Claude Code admite un sistema de plugins que le permite extender la funcionalidad con skills, agentes, hooks y MCP servers. Los plugins se distribuyen a través de marketplaces y pueden configurarse en niveles de usuario y repositorio.

Configuración de plugins

Configuraciones relacionadas con plugins en `settings.json`:

```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

`enabledPlugins`

Controla qué plugins están habilitados. Formato: `"plugin-name@marketplace-name": true/false`

Alcances:

- **Configuraciones de usuario** (`~/.claude/settings.json`): Preferencias personales de plugins
- **Configuraciones de proyecto** (`.claude/settings.json`): Plugins específicos del proyecto compartidos con el equipo
- **Configuraciones locales** (`.claude/settings.local.json`): Anulaciones por máquina (no confirmadas)

Ejemplo:

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

extraKnownMarketplaces

Define marketplaces adicionales que deben estar disponibles para el repositorio. Típicamente se usa en configuraciones a nivel de repositorio para asegurar que los miembros del equipo tengan acceso a fuentes de plugins requeridas.

Cuando un repositorio incluye `extraKnownMarketplaces` :

1. Se solicita a los miembros del equipo instalar el marketplace cuando confían en la carpeta
2. Luego se solicita a los miembros del equipo instalar plugins de ese marketplace
3. Los usuarios pueden omitir marketplaces o plugins no deseados (almacenados en configuraciones de usuario)
4. La instalación respeta límites de confianza y requiere consentimiento explícito

Ejemplo:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

Tipos de fuente de marketplace:

- `github` : Repositorio de GitHub (usa `repo`)
- `git` : Cualquier URL de git (usa `url`)
- `directory` : Ruta del sistema de archivos local (usa `path` , solo para desarrollo)
- `hostPattern` : Patrón regex para coincidir con hosts de marketplace (usa `hostPattern`)

`strictKnownMarketplaces`

Solo configuraciones administradas: Controla qué marketplaces de plugins se permite a los usuarios agregar. Esta configuración solo se puede configurar en [configuraciones administradas](#) y proporciona a los administradores control estricto sobre fuentes de marketplace.

Ubicaciones de archivos de configuración administrada:

- **macOS:** `/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux y WSL:** `/etc/claude-code/managed-settings.json`
- **Windows:** `C:\Program Files\ClaudeCode\managed-settings.json`

Características clave:

- Solo disponible en configuraciones administradas (`managed-settings.json`)

- No puede ser anulado por configuraciones de usuario o proyecto (precedencia más alta)
- Se aplica ANTES de operaciones de red/sistema de archivos (las fuentes bloqueadas nunca se ejecutan)
- Usa coincidencia exacta para especificaciones de fuente (incluyendo `ref`, `path` para fuentes de git), excepto `hostPattern`, que usa coincidencia regex

Comportamiento de lista blanca:

- `undefined` (predeterminado): Sin restricciones - los usuarios pueden agregar cualquier marketplace
- Matriz vacía `[]`: Bloqueo completo - los usuarios no pueden agregar nuevos marketplaces
- Lista de fuentes: Los usuarios solo pueden agregar marketplaces que coincidan exactamente

Todos los tipos de fuente admitidos:

La lista blanca admite siete tipos de fuente de marketplace. La mayoría de las fuentes usan coincidencia exacta, mientras que `hostPattern` usa coincidencia regex contra el host del marketplace.

1. Repositorios de GitHub:

```
{ "source": "github", "repo": "acme-corp/approved-plugins" }
{ "source": "github", "repo": "acme-corp/security-tools", "ref": "v2.0" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main", "path": "marketplace" }
```

Campos: `repo` (requerido), `ref` (opcional: rama/etiqueta/SHA), `path` (opcional: subdirectorio)

1. Repositorios de Git:

```
{ "source": "git", "url": "https://gitlab.example.com/tools/plugins.git" }
{ "source": "git", "url": "https://bitbucket.org/acme-corp/plugins.git", "ref": "production" }
{ "source": "git", "url": "ssh://git@git.example.com/plugins.git", "ref": "v3.1", "path": "approved" }
```

Campos: **url** (requerido), **ref** (opcional: rama/etiqueta/SHA), **path** (opcional: subdirectorio)

1. Marketplaces basados en URL:

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

Campos: **url** (requerido), **headers** (opcional: encabezados HTTP para acceso autenticado)

Note:

Los marketplaces basados en URL solo descargan el archivo **marketplace.json**. No descargan archivos de plugins del servidor. Los plugins en marketplaces basados en URL deben usar fuentes externas (URLs de GitHub, npm o git) en lugar de rutas relativas. Para plugins con rutas relativas, use un marketplace basado en Git en su lugar. Consulte [Solución de problemas](#) para obtener detalles.

1. Paquetes NPM:

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

Campos: **package** (requerido, admite paquetes con alcance)

1. Rutas de archivo:

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

Campos: **path** (requerido: ruta absoluta al archivo marketplace.json)

1. Rutas de directorio:

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```


Campos: `path` (requerido: ruta absoluta al directorio que contiene `.claude-plugin/marketplace.json`)

1. Coincidencia de patrón de host:

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

Campos: `hostPattern` (requerido: patrón regex para coincidir contra el host del marketplace)

Use coincidencia de patrón de host cuando desee permitir todos los marketplaces de un host específico sin enumerar cada repositorio individualmente. Esto es útil para organizaciones con servidores internos de GitHub Enterprise o GitLab donde los desarrolladores crean sus propios marketplaces.

Extracción de host por tipo de fuente:

- `github` : siempre coincide contra `github.com`
- `git` : extrae nombre de host de la URL (admite formatos HTTPS y SSH)
- `url` : extrae nombre de host de la URL
- `npm` , `file` , `directory` : no admitido para coincidencia de patrón de host

Ejemplos de configuración:

Ejemplo: permitir solo marketplaces específicos:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    },
    {
      "source": "npm",
      "package": "@acme-corp/compliance-plugins"
    }
  ]
}
```

Ejemplo - Deshabilitar todas las adiciones de marketplace:

```
{
  "strictKnownMarketplaces": []
}
```

Ejemplo: permitir todos los marketplaces de un servidor git interno:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Requisitos de coincidencia exacta:

Las fuentes de marketplace deben coincidir **exactamente** para que se permita la adición de un usuario. Para fuentes basadas en git (`github` y `git`), esto incluye todos los campos opcionales:

- El `repo` o `url` debe coincidir exactamente
- El campo `ref` debe coincidir exactamente (o ambos estar sin definir)
- El campo `path` debe coincidir exactamente (o ambos estar sin definir)

Ejemplos de fuentes que **NO coinciden**:

```
// Estas son fuentes DIFERENTES:
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// Estas también son DIFERENTES:
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

Comparación con `extraKnownMarketplaces` :

Aspecto	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Propósito	Aplicación de política organizacional	Conveniencia del equipo
Archivo de configuración	Solo <code>managed-settings.json</code>	Cualquier archivo de configuración

Aspecto	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Comportamiento	Bloquea adiciones no permitidas	Auto-instala marketplaces faltantes
Cuándo se aplica	Antes de operaciones de red/sistema de archivos	Después de solicitud de confianza del usuario
Puede ser anulado	No (precedencia más alta)	Sí (por configuraciones de mayor precedencia)
Formato de fuente	Objeto de fuente directo	Marketplace nombrado con fuente anidada
Caso de uso	Cumplimiento, restricciones de seguridad	Incorporación, estandarización

Diferencia de formato:

`strictKnownMarketplaces` usa objetos de fuente directos:

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` requiere marketplaces nombrados:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

Notas importantes:

- Las restricciones se verifican ANTES de cualquier solicitud de red u operación del sistema de archivos

- Cuando se bloquea, los usuarios ven mensajes de error claros indicando que la fuente está bloqueada por política administrada
- La restricción se aplica solo a agregar NUEVOS marketplaces; los marketplaces instalados previamente permanecen accesibles
- Las configuraciones administradas tienen la precedencia más alta y no pueden ser anuladas

Consulte [Restricciones de marketplace administradas](#) para documentación dirigida al usuario.

Gestionar plugins

Use el comando `/plugin` para gestionar plugins interactivamente:

- Examinar plugins disponibles de marketplaces
- Instalar/desinstalar plugins
- Habilitar/deshabilitar plugins
- Ver detalles de plugins (comandos, agentes, hooks proporcionados)
- Agregar/eliminar marketplaces

Obtenga más información sobre el sistema de plugins en la [documentación de plugins](#).

Variables de entorno

Claude Code admite las siguientes variables de entorno para controlar su comportamiento:

Note:

Todas las variables de entorno también se pueden configurar en `settings.json`. Esto es útil como una forma de establecer automáticamente variables de entorno para cada sesión, o para implementar un conjunto de variables de entorno para todo su equipo u organización.

Variable	Propósito	
<code>ANTHROPIC_API_KEY</code>	Clave API enviada como encabezado <code>X-API-Key</code> , típicamente para el SDK de Claude (para uso interactivo, ejecute <code>/login</code>)	

Variable	Propósito	
<code>ANTHROPIC_AUTH_TOKEN</code>	Valor personalizado para el encabezado <code>Authorization</code> (el valor que establezca aquí será prefijado con <code>Bearer</code>)	
<code>ANTHROPIC_CUSTOM_HEADERS</code>	Encabezados personalizados para agregar a solicitudes (formato <code>Name: Value</code> , separados por saltos de línea para múltiples encabezados)	
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Consulte Configuración de modelo	
<code>ANTHROPIC_DEFAULT_OPENAI_MODEL</code>	Consulte Configuración de modelo	
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Consulte Configuración de modelo	
<code>ANTHROPIC_FOUNDRY_API_KEY</code>	Clave API para autenticación de Microsoft Foundry (consulte Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_BASE_URL</code>	URL base completa para el recurso de Foundry (por ejemplo, <code>https://my-resource.services.ai.azure.com/anthropic</code>). Alternativa a <code>ANTHROPIC_FOUNDRY_RESOURCE</code> (consulte Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_RESOURCE</code>	Nombre del recurso de Foundry (por ejemplo, <code>my-resource</code>). Requerido si <code>ANTHROPIC_FOUNDRY_BASE_URL</code> no está establecido (consulte Microsoft Foundry)	
<code>ANTHROPIC_MODEL</code>	Nombre de la configuración de modelo a usar (consulte Configuración de modelo)	
<code>ANTHROPIC_SMALL_FAST_MODEL</code>	[OBSOLETO] Nombre de modelo de clase Haiku para tareas de fondo	

Variable	Propósito	
<code>ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION</code>	Anular región de AWS para el modelo de clase Haiku cuando se usa Bedrock	
<code>AWS_BEARER_TOKEN_BEDROCK</code>	Clave API de Bedrock para autenticación (consulte Claves API de Bedrock)	
<code>BASH_DEFAULT_TIMEOUT_MS</code>	Tiempo de espera predeterminado para comandos bash de larga duración	
<code>BASH_MAX_OUTPUT_LENGTH</code>	Número máximo de caracteres en salidas de bash antes de que se truncen en el medio	
<code>BASH_MAX_TIMEOUT_MS</code>	Tiempo de espera máximo que el modelo puede establecer para comandos bash de larga duración	
<code>CLAUDE_AUTOCOMPACT_PROMPT_OVERRIDE</code>	Establecer el porcentaje de capacidad de ventana de contexto (1-100) en el que se activa la compactación automática. De forma predeterminada, la compactación automática se activa aproximadamente al 95% de capacidad. Use valores más bajos como <code>50</code> para compactar antes. Los valores por encima del umbral predeterminado no tienen efecto. Se aplica a conversaciones principales y subagentes. Este porcentaje se alinea con el campo <code>context_window.used_percentage</code> disponible en línea de estado	
<code>CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR</code>	Volver al directorio de trabajo original después de cada comando Bash	

Variable	Propósito	
<code>CLAUDE_CODE_ACCOUNT_UUID</code>	UUID de cuenta para el usuario autenticado. Utilizado por llamadores de SDK para proporcionar información de cuenta de forma síncrona, evitando una condición de carrera donde eventos de telemetría tempranos carecen de metadatos de cuenta. Requiere que <code>CLAUDE_CODE_USER_EMAIL</code> y <code>CLAUDE_CODE_ORGANIZATION_UUID</code> también estén establecidos	
<code>CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD</code>	Establecer en <code>1</code> para cargar archivos CLAUDE.md de directorios especificados con <code>--add-dir</code> . De forma predeterminada, los directorios adicionales no cargan archivos de memoria	<code>1</code>
<code>CLAUDE_CODE_API_KEY_HELPER_TTL_MS</code>	Intervalo en milisegundos en el que las credenciales deben actualizarse (cuando se usa <code>apiKeyHelper</code>)	
<code>CLAUDE_CODE_CLIENT_CERT</code>	Ruta al archivo de certificado de cliente para autenticación mTLS	
<code>CLAUDE_CODE_CLIENT_KEY</code>	Ruta al archivo de clave privada de cliente para autenticación mTLS	
<code>CLAUDE_CODE_CLIENT_KEY_PASSPHRASE</code>	Frase de contraseña para <code>CLAUDE_CODE_CLIENT_KEY</code> cifrado (opcional)	
<code>CLAUDE_CODE_DISABLE_1M_CONTEXT</code>	Establecer en <code>1</code> para deshabilitar soporte de ventana de contexto de 1M . Cuando se establece, las variantes de modelo de 1M no están disponibles en el selector de modelo. Útil para entornos empresariales con requisitos de cumplimiento	

Variable	Propósito	
<code>CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING</code>	Establecer en <code>1</code> para deshabilitar razonamiento adaptativo para Opus 4.6 y Sonnet 4.6. Cuando se deshabilita, estos modelos vuelven al presupuesto de pensamiento fijo controlado por <code>MAX_THINKING_TOKENS</code>	
<code>CLAUDE_CODE_DISABLE_AUTO_MEMORY</code>	Establecer en <code>1</code> para deshabilitar memoria automática . Establecer en <code>0</code> para forzar memoria automática durante el despliegue gradual. Cuando se deshabilita, Claude no crea ni carga archivos de memoria automática	
<code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code>	Establecer en <code>1</code> para eliminar instrucciones de flujo de trabajo de commit y PR integradas del indicador del sistema de Claude. Útil cuando se usan sus propias skills de flujo de trabajo de git. Tiene prioridad sobre la configuración <code>includeGitInstructions</code> cuando se establece	
<code>CLAUDE_CODE_DISABLE_BACKGROUND_TASKS</code>	Establecer en <code>1</code> para deshabilitar toda la funcionalidad de tareas de fondo, incluyendo el parámetro <code>run_in_background</code> en herramientas Bash y subagent, auto-backgrounding y el atajo Ctrl+B	
<code>CLAUDE_CODE_DISABLE_CRON</code>	Establecer en <code>1</code> para deshabilitar tareas programadas . La skill <code>/loop</code> y las herramientas cron se vuelven no disponibles y cualquier tarea ya programada deja de ejecutarse, incluyendo tareas que ya se están ejecutando a mitad de sesión	

Variable	Propósito	
<code>CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS</code>	Establecer en <code>1</code> para deshabilitar encabezados <code>anthropic-beta</code> específicos de la API de Anthropic. Use esto si experimenta problemas como “Unexpected value(s) for the <code>anthropic-beta</code> header” cuando se usa una puerta de enlace LLM con proveedores de terceros	
<code>CLAUDE_CODE_DISABLE_FAST_MODE</code>	Establecer en <code>1</code> para deshabilitar modo rápido	
<code>CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY</code>	Establecer en <code>1</code> para deshabilitar las encuestas de calidad de sesión “¿Cómo está Claude?”. También se deshabilita cuando se usan proveedores de terceros o cuando la telemetría está deshabilitada. Consulte Encuestas de calidad de sesión	
<code>CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC</code>	Equivalente a establecer <code>DISABLE_AUTOUPDATER</code> , <code>DISABLE_BUG_COMMAND</code> , <code>DISABLE_ERROR_REPORTING</code> y <code>DISABLE_TELEMETRY</code>	
<code>CLAUDE_CODE_DISABLE_TERMINAL_TITLE</code>	Establecer en <code>1</code> para deshabilitar actualizaciones automáticas del título del terminal basadas en contexto de conversación	
<code>CLAUDE_CODE_EFFORT_LEVEL</code>	Establecer el nivel de esfuerzo para modelos admitidos. Valores: <code>low</code> , <code>medium</code> , <code>high</code> . El esfuerzo más bajo es más rápido y barato, el esfuerzo más alto proporciona razonamiento más profundo. Admitido en Opus 4.6 y Sonnet 4.6. Consulte Ajustar nivel de esfuerzo	

Variable	Propósito	
<code>CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION</code>	Establecer en <code>false</code> para deshabilitar sugerencias de indicación (el toggle “Prompt suggestions” en <code>/config</code>). Estas son las predicciones atenuadas que aparecen en su entrada de indicación después de que Claude responde. Consulte Sugerencias de indicación	
<code>CLAUDE_CODE_ENABLE_TASKS</code>	Establecer en <code>false</code> para revertir temporalmente a la lista TODO anterior en lugar del sistema de seguimiento de tareas. Predeterminado: <code>true</code> . Consulte Lista de tareas	
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	Establecer en <code>1</code> para habilitar la recopilación de datos de OpenTelemetry para métricas y registro. Requerido antes de configurar exportadores de OTel. Consulte Monitoreo	
<code>CLAUDE_CODE_EXIT_AFTER_STOP_DELAY</code>	Tiempo en milisegundos a esperar después de que el bucle de consulta se vuelva inactivo antes de salir automáticamente. Útil para flujos de trabajo automatizados y scripts que usan modo SDK	
<code>CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS</code>	Establecer en <code>1</code> para habilitar equipos de agentes . Los equipos de agentes son experimentales y están deshabilitados de forma predeterminada	
<code>CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS</code>	Anular el límite de token predeterminado para lecturas de archivo. Útil cuando necesita leer archivos más grandes en su totalidad	
<code>CLAUDE_CODE_HIDE_ACCOUNT_INFO</code>	Establecer en <code>1</code> para ocultar su dirección de correo electrónico y nombre de organización de la UI de Claude Code. Útil cuando se transmite o se graban sesiones	

Variable	Propósito	
<code>CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL</code>	Omitir auto-instalación de extensiones IDE	
<code>CLAUDE_CODE_MAX_OUTPUT_TOKENS</code>	Establecer el número máximo de tokens de salida para la mayoría de solicitudes. Predeterminado: 32,000. Máximo: 64,000. Aumentar este valor reduce la ventana de contexto efectiva disponible antes de que se active compactación automática .	
<code>CLAUDE_CODE_ORGANIZATION_UUID</code>	UUID de organización para el usuario autenticado. Utilizado por llamadores de SDK para proporcionar información de cuenta de forma síncrona. Requiere que <code>CLAUDE_CODE_ACCOUNT_UUID</code> y <code>CLAUDE_CODE_USER_EMAIL</code> también estén establecidos	
<code>CLAUDE_CODE_OTEL_HELPERS_HELPER_DEBOUNCE_MS</code>	Intervalo para actualizar encabezados dinámicos de OpenTelemetry en milisegundos (predeterminado: 1740000 / 29 minutos). Consulte Encabezados dinámicos	
<code>CLAUDE_CODE_PLAN_MODE_REQUIRED</code>	Auto-establecido en <code>true</code> en compañeros de equipo de agentes que requieren aprobación de plan. Solo lectura: establecido por Claude Code al generar compañeros. Consulte requerir aprobación de plan	
<code>CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS</code>	Tiempo de espera en milisegundos para operaciones de git al instalar o actualizar plugins (predeterminado: 120000). Aumente este valor para repositorios grandes o conexiones de red lentas. Consulte Las operaciones de Git agotan el tiempo	

Variable	Propósito	
<code>CLAUDE_CODE_PROXY_RESOLVES_HOSTS</code>	Establecer en <code>true</code> para permitir que el proxy realice resolución de DNS en lugar de la persona que llama. Opt-in para entornos donde el proxy debe manejar la resolución de nombres de host	
<code>CLAUDE_CODE_SHELL</code>	Anular detección automática de shell. Útil cuando su shell de inicio difiere de su shell de trabajo preferido (por ejemplo, <code>bash</code> vs <code>zsh</code>)	
<code>CLAUDE_CODE_SHELL_PREFIX</code>	Prefijo de comando para envolver todos los comandos bash (por ejemplo, para registro o auditoría). Ejemplo: <code>/path/to/logger.sh</code> ejecutará <code>/path/to/logger.sh <command></code>	
<code>CLAUDE_CODE_SIMPLE</code>	Establecer en <code>1</code> para ejecutar con un indicador del sistema mínimo y solo las herramientas Bash, lectura de archivo y edición de archivo. Deshabilita herramientas MCP, adjuntos, hooks y archivos CLAUDE.md	
<code>CLAUDE_CODE_SKIP_BEDROCK_AUTH</code>	Omitir autenticación de AWS para Bedrock (por ejemplo, cuando se usa una puerta de enlace LLM)	
<code>CLAUDE_CODE_SKIP_FOUNDRY_AUTH</code>	Omitir autenticación de Azure para Microsoft Foundry (por ejemplo, cuando se usa una puerta de enlace LLM)	
<code>CLAUDE_CODE_SKIP_VERTEX_AUTH</code>	Omitir autenticación de Google para Vertex (por ejemplo, cuando se usa una puerta de enlace LLM)	
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	Consulte Configuración de modelo	

Variable	Propósito	
<code>CLAUDE_CODE_TASK_LIST_ID</code>	Compartir una lista de tareas entre sesiones. Establezca el mismo ID en múltiples instancias de Claude Code para coordinar en una lista de tareas compartida. Consulte Lista de tareas	
<code>CLAUDE_CODE_TEAM_NAME</code>	Nombre del equipo de agentes al que pertenece este compañero. Se establece automáticamente en miembros de equipo de agentes	
<code>CLAUDE_CODE_TMPDIR</code>	Anular el directorio temporal usado para archivos temporales internos. Claude Code agrega <code>/claude/</code> a esta ruta. Predeterminado: <code>/tmp</code> en Unix/macOS, <code>os.tmpdir()</code> en Windows	
<code>CLAUDE_CODE_USER_EMAIL</code>	Dirección de correo electrónico para el usuario autenticado. Utilizado por llamadores de SDK para proporcionar información de cuenta de forma síncrona. Requiere que <code>CLAUDE_CODE_ACCOUNT_UUID</code> y <code>CLAUDE_CODE_ORGANIZATION_UUID</code> también estén establecidos	
<code>CLAUDE_CODE_USE_BEDROCK</code>	Usar Bedrock	
<code>CLAUDE_CODE_USE_FOUNDRY</code>	Usar Microsoft Foundry	
<code>CLAUDE_CODE_USE_VERTEX</code>	Usar Vertex	
<code>CLAUDE_CONFIG_DIR</code>	Personalizar dónde Claude Code almacena sus archivos de configuración y datos	
<code>DISABLE_AUTOUPDATER</code>	Establecer en <code>1</code> para deshabilitar actualizaciones automáticas.	

Variable	Propósito	
<code>DISABLE_BUG_COMMAND</code>	Establecer en <code>1</code> para deshabilitar el comando <code>/bug</code>	
<code>DISABLE_COST_WARNING</code>	Establecer en <code>1</code> para deshabilitar mensajes de advertencia de costo	
<code>DISABLE_ERROR_REPORTING</code>	Establecer en <code>1</code> para optar por no participar en informes de errores de Sentry	
<code>DISABLE_INSTALLATION_CHECKS</code>	Establecer en <code>1</code> para deshabilitar advertencias de instalación. Use solo cuando administre manualmente la ubicación de instalación, ya que esto puede enmascarar problemas con instalaciones estándar	
<code>DISABLE_NON_ESSENTIAL_MODEL_CALLS</code>	Establecer en <code>1</code> para deshabilitar llamadas de modelo para rutas no críticas como texto de sabor	
<code>DISABLE_PROMPT_CACHING</code>	Establecer en <code>1</code> para deshabilitar prompt caching para todos los modelos (tiene prioridad sobre configuraciones por modelo)	
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Establecer en <code>1</code> para deshabilitar prompt caching para modelos Haiku	
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Establecer en <code>1</code> para deshabilitar prompt caching para modelos Opus	
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Establecer en <code>1</code> para deshabilitar prompt caching para modelos Sonnet	
<code>DISABLE_TELEMETRY</code>	Establecer en <code>1</code> para optar por no participar en telemetría de Statsig (tenga en cuenta que los eventos de Statsig no incluyen datos de usuario como código, rutas de archivo o comandos bash)	

Variable	Propósito	
<code>ENABLE_CLAUDEAI_MCP_SERVERS</code>	Establecer en <code>false</code> para deshabilitar MCP servers de claude.ai en Claude Code. Habilitado de forma predeterminada para usuarios conectados	
<code>ENABLE_TOOL_SEARCH</code>	Controla búsqueda de herramientas MCP . Valores: <code>auto</code> (predeterminado, habilita al 10% de contexto), <code>auto:N</code> (umbral personalizado, por ejemplo, <code>auto:5</code> para 5%), <code>true</code> (siempre activado), <code>false</code> (deshabilitado)	
<code>FORCE_AUTOUPDATE_PLUGINS</code>	Establecer en <code>true</code> para forzar auto-actualizaciones de plugins incluso cuando el auto-actualizador principal está deshabilitado a través de <code>DISABLE_AUTOUPDATER</code>	
<code>HTTP_PROXY</code>	Especificar servidor proxy HTTP para conexiones de red	
<code>HTTPS_PROXY</code>	Especificar servidor proxy HTTPS para conexiones de red	
<code>IS_DEMO</code>	Establecer en <code>true</code> para habilitar modo de demostración: oculta correo electrónico y organización de la UI, omite incorporación y oculta comandos internos. Útil para transmitir o grabar sesiones	
<code>MAX_MCP_OUTPUT_TOKENS</code>	Número máximo de tokens permitidos en respuestas de herramientas MCP. Claude Code muestra una advertencia cuando la salida excede 10,000 tokens (predeterminado: 25000)	

Variable	Propósito	
<code>MAX_THINKING_TOKENS</code>	Anular el presupuesto de token de pensamiento extendido . El pensamiento está habilitado al presupuesto máximo (31,999 tokens) de forma predeterminada. Use esto para limitar el presupuesto (por ejemplo, <code>MAX_THINKING_TOKENS=10000</code>) o deshabilitar el pensamiento completamente (<code>MAX_THINKING_TOKENS=0</code>). Para Opus 4.6, la profundidad de pensamiento se controla por nivel de esfuerzo en su lugar, y esta variable se ignora a menos que se establezca en <code>0</code> para deshabilitar el pensamiento.	
<code>MCP_CLIENT_SECRET</code>	Secreto de cliente OAuth para MCP servers que requieren credenciales preconfiguradas . Evita el indicador interactivo al agregar un servidor con <code>--client-secret</code>	
<code>MCP_OAUTH_CALLBACK_PORT</code>	Puerto fijo para la devolución de llamada de redirección de OAuth, como alternativa a <code>--callback-port</code> al agregar un MCP server con credenciales preconfiguradas	
<code>MCP_TIMEOUT</code>	Tiempo de espera en milisegundos para inicio de MCP server	
<code>MCP_TOOL_TIMEOUT</code>	Tiempo de espera en milisegundos para ejecución de herramienta MCP	
<code>NO_PROXY</code>	Lista de dominios e IPs a los que se emitirán solicitudes directamente, omitiendo proxy	

Variable	Propósito	
<code>SLASH_COMMAND_TOOL_CHARACTER_BUDGET</code>	Anular el presupuesto de caracteres para metadatos de skill mostrados a la herramienta Skill . El presupuesto se escala dinámicamente al 2% de la ventana de contexto, con un respaldo de 16,000 caracteres. Nombre heredado mantenido para compatibilidad hacia atrás	
<code>USE_BUILTIN_RIPGREP</code>	Establecer en <code>0</code> para usar <code>rg</code> instalado en el sistema en lugar de <code>rg</code> incluido con Claude Code	
<code>VERTEX_REGION_CLAUDE_3_5_HAIKU</code>	Anular región para Claude 3.5 Haiku cuando se usa Vertex AI	
<code>VERTEX_REGION_CLAUDE_3_7_SONNET</code>	Anular región para Claude 3.7 Sonnet cuando se usa Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_OPUS</code>	Anular región para Claude 4.0 Opus cuando se usa Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_SONNET</code>	Anular región para Claude 4.0 Sonnet cuando se usa Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_1_OPUS</code>	Anular región para Claude 4.1 Opus cuando se usa Vertex AI	

Herramientas disponibles para Claude

Claude Code tiene acceso a un conjunto de herramientas poderosas que le ayudan a entender y modificar su base de código:

Herramienta	Descripción	Permiso requerido
Agent	Genera un subagent con su propia ventana de contexto para manejar una tarea	No
AskUserQuestion	Hace preguntas de opción múltiple para recopilar requisitos o aclarar ambigüedad	No

Herramienta	Descripción	Permiso requerido
Bash	Ejecuta comandos de shell en su entorno. Consulte Comportamiento de herramienta Bash	Sí
CronCreate	Programa una indicación recurrente o de una sola vez dentro de la sesión actual (desaparece cuando Claude sale). Consulte tareas programadas	No
CronDelete	Cancela una tarea programada por ID	No
CronList	Lista todas las tareas programadas en la sesión	No
Edit	Realiza ediciones dirigidas a archivos específicos	Sí
EnterPlanMode	Cambia a modo de plan para diseñar un enfoque antes de codificar	No
EnterWorktree	Crea un git worktree aislado y cambia a él	No
ExitPlanMode	Presenta un plan para aprobación y sale del modo de plan	Sí
ExitWorktree	Sale de una sesión de worktree y vuelve al directorio original	No
Glob	Encuentra archivos basados en coincidencia de patrones	No
Grep	Busca patrones en contenidos de archivos	No
ListMcpResourcesTool	Lista recursos expuestos por MCP servers conectados	No

Herramienta	Descripción	Permiso requerido
LSP	Inteligencia de código a través de servidores de lenguaje. Reporta errores de tipo y advertencias automáticamente después de ediciones de archivo. También admite operaciones de navegación: saltar a definiciones, encontrar referencias, obtener información de tipo, listar símbolos, encontrar implementaciones, rastrear jerarquías de llamadas. Requiere un plugin de inteligencia de código y su binario de servidor de lenguaje	No
NotebookEdit	Modifica celdas de cuaderno Jupyter	Sí
Read	Lee el contenido de archivos	No
ReadMcpResourceTool	Lee un recurso MCP específico por URI	No
Skill	Ejecuta una skill dentro de la conversación principal	Sí
TaskCreate	Crea una nueva tarea en la lista de tareas	No
TaskGet	Recupera detalles completos para una tarea específica	No
TaskList	Lista todas las tareas con su estado actual	No
TaskOutput	Recupera salida de una tarea de fondo	No
TaskStop	Mata una tarea de fondo en ejecución por ID	No
TaskUpdate	Actualiza estado de tarea, dependencias, detalles o elimina tareas	No

Herramienta	Descripción	Permiso requerido
TodoWrite	Gestiona la lista de verificación de tareas de sesión. Disponible en modo no interactivo y el SDK de Agent ; las sesiones interactivas usan TaskCreate, TaskGet, TaskList y TaskUpdate en su lugar	No
ToolSearch	Busca y carga herramientas diferidas cuando búsqueda de herramientas está habilitada	No
WebFetch	Obtiene contenido de una URL especificada	Sí
WebSearch	Realiza búsquedas web	Sí
Write	Crea o sobrescribe archivos	Sí

Las reglas de permiso se pueden configurar usando `/allowed-tools` o en [configuración de permisos](#). También consulte [Reglas de permiso específicas de herramientas](#).

Comportamiento de herramienta Bash

La herramienta Bash ejecuta comandos de shell con el siguiente comportamiento de persistencia:

- **El directorio de trabajo persiste:** Cuando Claude cambia el directorio de trabajo (por ejemplo, `cd /path/to/dir`), los comandos Bash posteriores se ejecutarán en ese directorio. Puede usar `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` para volver al directorio del proyecto después de cada comando.
- **Las variables de entorno NO persisten:** Las variables de entorno establecidas en un comando Bash (por ejemplo, `export MY_VAR=value`) **no** están disponibles en comandos Bash posteriores. Cada comando Bash se ejecuta en un entorno de shell fresco.

Para hacer que las variables de entorno estén disponibles en comandos Bash, tiene **tres opciones**:

Opción 1: Activar entorno antes de iniciar Claude Code (enfoque más simple)

Active su entorno virtual en su terminal antes de lanzar Claude Code:

```
conda activate myenv
## o: source /path/to/venv/bin/activate
claude
```

Esto funciona para entornos de shell pero las variables de entorno establecidas dentro de los comandos Bash de Claude no persistirán entre comandos.

Opción 2: Establecer `CLAUDE_ENV_FILE` antes de iniciar Claude Code (configuración de entorno persistente)

Exporte la ruta a un script de shell que contiene su configuración de entorno:

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

Donde `/path/to/env-setup.sh` contiene:

```
conda activate myenv
## o: source /path/to/venv/bin/activate
## o: export MY_VAR=value
```

Claude Code obtendrá este archivo antes de cada comando Bash, haciendo que el entorno sea persistente en todos los comandos.

Opción 3: Usar un hook `SessionStart` (configuración específica del proyecto)

Configure en `.claude/settings.json`:

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"${CLAUDE_ENV_FILE}\""
      }]
    }]
  }
}
```

El hook escribe en `$CLAUDE_ENV_FILE`, que luego se obtiene antes de cada comando Bash. Esto es ideal para configuraciones de proyecto compartidas por el equipo.

Consulte [hooks SessionStart](#) para obtener más detalles sobre la Opción 3.

Extender herramientas con hooks

Puede ejecutar comandos personalizados antes o después de que cualquier herramienta se ejecute usando [hooks de Claude Code](#).

Por ejemplo, podría ejecutar automáticamente un formateador de Python después de que Claude modifique archivos de Python, o prevenir modificaciones a archivos de configuración de producción bloqueando operaciones de Write a ciertas rutas.

Ver también

- [Permisos](#): sistema de permisos, sintaxis de reglas, patrones específicos de herramientas y políticas administradas
- [Autenticación](#): configurar acceso de usuario a Claude Code
- [Solución de problemas](#): soluciones para problemas de configuración comunes

Configuración del modelo

Aprenda sobre la configuración del modelo Claude Code, incluidos los alias de modelo como `opuslan`

Modelos disponibles

Para la configuración de `model` en Claude Code, puede configurar:

- Un **alias de modelo**
- Un **nombre de modelo**
 - API de Anthropic: Un [nombre de modelo](#) completo
 - Bedrock: un ARN de perfil de inferencia
 - Foundry: un nombre de implementación
 - Vertex: un nombre de versión

Alias de modelo

Los alias de modelo proporcionan una forma conveniente de seleccionar configuraciones de modelo sin necesidad de recordar números de versión exactos:

Alias de modelo	Comportamiento
<code>default</code>	Configuración de modelo recomendada, según el tipo de cuenta
<code>sonnet</code>	Utiliza el último modelo Sonnet (actualmente Sonnet 4.6) para tareas de codificación diaria
<code>opus</code>	Utiliza el último modelo Opus (actualmente Opus 4.6) para tareas de razonamiento complejo
<code>haiku</code>	Utiliza el modelo Haiku rápido y eficiente para tareas simples
<code>sonnet[1m]</code>	Utiliza Sonnet con una ventana de contexto de 1 millón de tokens para sesiones largas
<code>opuslan</code>	Modo especial que utiliza <code>opus</code> durante el modo de plan, luego cambia a <code>sonnet</code> para la ejecución

Los alias siempre apuntan a la versión más reciente. Para fijar una versión específica, utilice el nombre de modelo completo (por ejemplo, `claude-opus-4-6`) o establezca la variable de entorno correspondiente como `ANTHROPIC_DEFAULT_OPUS_MODEL`.

Configurar su modelo

Puede configurar su modelo de varias formas, enumeradas en orden de prioridad:

1. **Durante la sesión** - Utilice `/model <alias|name>` para cambiar modelos durante la sesión
2. **Al iniciar** - Inicie con `claude --model <alias|name>`
3. **Variable de entorno** - Establezca `ANTHROPIC_MODEL=<alias|name>`
4. **Configuración** - Configure permanentemente en su archivo de configuración utilizando el campo `model`.

Ejemplo de uso:

```
## Iniciar con Opus
claude --model opus

## Cambiar a Sonnet durante la sesión
/model sonnet
```

Archivo de configuración de ejemplo:

```
{
  "permissions": {
    ...
  },
  "model": "opus"
}
```

Restringir la selección de modelo

Los administradores empresariales pueden utilizar `availableModels` en [configuración administrada o de política](#) para restringir qué modelos pueden seleccionar los usuarios.

Cuando se establece `availableModels`, los usuarios no pueden cambiar a modelos que no estén en la lista a través de `/model`, la bandera `--model`, la herramienta Config o la variable de entorno `ANTHROPIC_MODEL`.

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

Comportamiento del modelo predeterminado

La opción Predeterminado en el selector de modelo no se ve afectada por `availableModels`. Siempre permanece disponible y representa el valor predeterminado de tiempo de ejecución del sistema [basado en el nivel de suscripción del usuario](#).

Incluso con `availableModels: []`, los usuarios aún pueden usar Claude Code con el modelo Predeterminado para su nivel.

Controlar el modelo en el que se ejecutan los usuarios

Para controlar completamente la experiencia del modelo, utilice `availableModels` junto con la configuración `model`:

- **availableModels:** restringe a qué pueden cambiar los usuarios
- **model:** establece la anulación de modelo explícita, tomando precedencia sobre el Predeterminado

Este ejemplo garantiza que todos los usuarios ejecuten Sonnet 4.6 y solo puedan elegir entre Sonnet y Haiku:

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

Comportamiento de fusión

Cuando `availableModels` se establece en múltiples niveles, como configuración de usuario y configuración de proyecto, los arrays se fusionan y se deduplican. Para aplicar una lista de permitidos estricta, establezca `availableModels` en configuración administrada o de política que tenga la máxima prioridad.

Comportamiento especial del modelo

Configuración del modelo `default`

El comportamiento de `default` depende del tipo de cuenta:

- **Max y Team Premium:** por defecto Opus 4.6
- **Pro y Team Standard:** por defecto Sonnet 4.6
- **Enterprise:** Opus 4.6 está disponible pero no es el predeterminado

Claude Code puede retroceder automáticamente a Sonnet si alcanza un umbral de uso con Opus.

Configuración del modelo `opusplan`

El alias de modelo `opusplan` proporciona un enfoque híbrido automatizado:

- **En modo de plan** - Utiliza `opus` para razonamiento complejo y decisiones de arquitectura
- **En modo de ejecución** - Cambia automáticamente a `sonnet` para generación de código e implementación

Esto le da lo mejor de ambos mundos: el razonamiento superior de Opus para la planificación y la eficiencia de Sonnet para la ejecución.

Ajustar el nivel de esfuerzo

[Los niveles de esfuerzo](#) controlan el razonamiento adaptativo, que asigna dinámicamente el pensamiento en función de la complejidad de la tarea. El esfuerzo menor es más rápido y económico para tareas directas, mientras que el esfuerzo mayor proporciona un razonamiento más profundo para problemas complejos.

Hay tres niveles disponibles: **low**, **medium** y **high**. Opus 4.6 tiene un valor predeterminado de esfuerzo medio para suscriptores de Max y Team.

Configurar esfuerzo:

- **En `/model`** : utilice las teclas de flecha izquierda/derecha para ajustar el control deslizante de esfuerzo al seleccionar un modelo
- **Variable de entorno:** establezca `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high`
- **Configuración:** establezca `effortLevel` en su archivo de configuración

El esfuerzo es compatible con Opus 4.6 y Sonnet 4.6. El control deslizante de esfuerzo aparece en `/model` cuando se selecciona un modelo compatible. El nivel de esfuerzo actual también se muestra junto al logotipo y el indicador (por ejemplo, “with low effort”), para que pueda confirmar qué configuración está activa sin abrir `/model`.

Para desactivar el razonamiento adaptativo en Opus 4.6 y Sonnet 4.6 y revertir al presupuesto de pensamiento fijo anterior, establezca

`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Cuando está desactivado, estos modelos utilizan el presupuesto fijo controlado por `MAX_THINKING_TOKENS`. Consulte [variables de entorno](#).

Contexto extendido

Opus 4.6 y Sonnet 4.6 admiten una [ventana de contexto de 1 millón de tokens](#) para sesiones largas con bases de código grandes.

Note:

La ventana de contexto de 1M está actualmente en beta. Las características, precios y disponibilidad pueden cambiar.

El contexto extendido está disponible para:

- **Usuarios de API y pago por uso:** acceso completo a contexto de 1M
- **Suscriptores de Pro, Max, Teams y Enterprise:** disponible con [uso adicional](#) habilitado

Para desactivar completamente el contexto de 1M, establezca

`CLAUDE_CODE_DISABLE_1M_CONTEXT=1`. Esto elimina variantes de modelo de 1M del selector de modelo. Consulte [variables de entorno](#).

Seleccionar un modelo de 1M no cambia inmediatamente la facturación. Su sesión utiliza tasas estándar hasta que excede 200K tokens de contexto. Más allá de 200K tokens, las solicitudes se cobran a [precios de contexto largo](#) con [límites de velocidad](#) dedicados. Para suscriptores, los tokens más allá de 200K se facturan como uso adicional en lugar de a través de la suscripción.

Si su cuenta admite contexto de 1M, la opción aparece en el selector de modelo (`/model`) en las últimas versiones de Claude Code. Si no la ve, intente reiniciar su sesión.

También puede utilizar el sufijo `[1m]` con alias de modelo o nombres de modelo completos:

```
## Utilizar el alias sonnet[1m]
/model sonnet[1m]

## O añadir [1m] a un nombre de modelo completo
/model claude-sonnet-4-6[1m]
```

Verificar su modelo actual

Puede ver qué modelo está utilizando actualmente de varias formas:

- 1. En [línea de estado](#) (si está configurada)
- 2. En `/status`, que también muestra la información de su cuenta.

Variables de entorno

Puede utilizar las siguientes variables de entorno, que deben ser **nombres de modelo** completos (o equivalentes para su proveedor de API), para controlar los nombres de modelo a los que se asignan los alias.

Variable de entorno	Descripción
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	El modelo a utilizar para <code>opus</code> , o para <code>opusplan</code> cuando Plan Mode está activo.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	El modelo a utilizar para <code>sonnet</code> , o para <code>opusplan</code> cuando Plan Mode no está activo.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	El modelo a utilizar para <code>haiku</code> , o funcionalidad de fondo
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	El modelo a utilizar para subagents

Nota: `ANTHROPIC_SMALL_FAST_MODEL` está deprecado en favor de `ANTHROPIC_DEFAULT_HAIKU_MODEL`.

Fijar modelos para implementaciones de terceros

Al implementar Claude Code a través de [Bedrock](#), [Vertex AI](#), o [Foundry](#), fije las versiones de modelo antes de implementar para los usuarios.

Sin fijar, Claude Code utiliza alias de modelo (`sonnet`, `opus`, `haiku`) que se resuelven a la versión más reciente. Cuando Anthropic lanza un nuevo modelo, los usuarios cuyas cuentas no tienen la nueva versión habilitada se romperán silenciosamente.

Warning:

Establezca las tres variables de entorno de modelo en IDs de versión específicos como parte de su configuración inicial. Omitir este paso significa que una actualización de Claude Code puede romper sus usuarios sin ninguna acción de su parte.

Utilice las siguientes variables de entorno con IDs de modelo específicos de versión para su proveedor:

Pro- vee- dor	Ejemplo
Bedrock	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'</code>
Vertex AI	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>
Foundry	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>

Aplice el mismo patrón para `ANTHROPIC_DEFAULT_SONNET_MODEL` y `ANTHROPIC_DEFAULT_HAIKU_MODEL`. Para IDs de modelo actuales y heredados en todos los proveedores, consulte [Descripción general de modelos](#). Para actualizar usuarios a una nueva versión de modelo, actualice estas variables de entorno e implemente nuevamente.

Note:

La lista de permitidos `settings.availableModels` aún se aplica cuando se utilizan proveedores de terceros. El filtrado coincide con el alias de modelo (`opus` , `sonnet` , `haiku`), no con el ID de modelo específico del proveedor.

Anular IDs de modelo por versión

Las variables de entorno a nivel de familia anteriores configuran un ID de modelo por alias de familia. Si necesita asignar varias versiones dentro de la misma familia a IDs de proveedor distintos, utilice la configuración `modelOverrides` en su lugar.

`modelOverrides` asigna IDs de modelo individuales de Anthropic a las cadenas específicas del proveedor que Claude Code envía a la API de su proveedor. Cuando un usuario selecciona un modelo asignado en el selector `/model`, Claude Code utiliza su valor configurado en lugar del valor predeterminado integrado.

Esto permite a los administradores empresariales enrutar cada versión de modelo a un ARN de perfil de inferencia de Bedrock específico, un nombre de versión de Vertex AI o un nombre de implementación de Foundry para gobernanza, asignación de costos o enrutamiento regional.

Establezca `modelOverrides` en su [archivo de configuración](#):

```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/opus-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-sonnet-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/sonnet-prod"
  }
}
```

Las claves deben ser IDs de modelo de Anthropic como se enumeran en la [Descripción general de modelos](#). Para IDs de modelo con fecha, incluya el sufijo de fecha exactamente como aparece allí. Las claves desconocidas se ignoran.

Las anulaciones reemplazan los IDs de modelo integrados que respaldan cada entrada en el selector `/model`. En Bedrock, las anulaciones tienen precedencia sobre cualquier perfil de inferencia que Claude Code descubra automáticamente al iniciar. Los valores que proporciona directamente a través de `ANTHROPIC_MODEL`, `--model`, o las variables de entorno `ANTHROPIC_DEFAULT_*_MODEL` se pasan al proveedor tal como están y no se transforman por `modelOverrides`.

`modelOverrides` funciona junto con `availableModels`. La lista de permitidos se evalúa contra el ID de modelo de Anthropic, no el valor de anulación, por lo que una entrada como `"opus"` en `availableModels` continúa coincidiendo incluso cuando las versiones de Opus se asignan a ARNs.

Configuración de almacenamiento en caché de indicaciones

Claude Code utiliza automáticamente [almacenamiento en caché de indicaciones](#) para optimizar el rendimiento y reducir costos. Puede desactivar el almacenamiento en caché de indicaciones globalmente o para niveles de modelo específicos:

Variable de entorno	Descripción
<code>DISABLE_PROMPT_CACHING</code>	Establezca en 1 para desactivar el almacenamiento en caché de indicaciones para todos los modelos (tiene precedencia sobre configuraciones por modelo)
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Establezca en 1 para desactivar el almacenamiento en caché de indicaciones solo para modelos Haiku
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Establezca en 1 para desactivar el almacenamiento en caché de indicaciones solo para modelos Sonnet
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Establezca en 1 para desactivar el almacenamiento en caché de indicaciones solo para modelos Opus

Estas variables de entorno le dan control granular sobre el comportamiento del almacenamiento en caché de indicaciones. La configuración global `DISABLE_PROMPT_CACHING` tiene precedencia sobre las configuraciones específicas del modelo, permitiéndole desactivar rápidamente todo el almacenamiento en caché cuando sea necesario. Las configuraciones por modelo son útiles para control selectivo, como cuando se depuran modelos específicos o se trabaja con proveedores de nube que pueden tener diferentes implementaciones de almacenamiento en caché.

Estilos de salida

Adapte Claude Code para usos más allá de la ingeniería de software

Los estilos de salida le permiten usar Claude Code como cualquier tipo de agente mientras se mantienen sus capacidades principales, como ejecutar scripts locales, leer/escribir archivos y realizar un seguimiento de TODOs.

Estilos de salida integrados

El estilo de salida **Default** de Claude Code es el mensaje del sistema existente, diseñado para ayudarlo a completar tareas de ingeniería de software de manera eficiente.

Hay dos estilos de salida integrados adicionales enfocados en enseñarle la base de código y cómo funciona Claude:

- **Explanatory:** Proporciona “Insights” educativos entre ayudarlo a completar tareas de ingeniería de software. Le ayuda a entender las opciones de implementación y los patrones de la base de código.
- **Learning:** Modo colaborativo de aprendizaje práctico donde Claude no solo comparará “Insights” mientras codifica, sino que también le pedirá que contribuya con pequeñas piezas de código estratégicas. Claude Code agregará marcadores `TODO(human)` en su código para que usted implemente.

Cómo funcionan los estilos de salida

Los estilos de salida modifican directamente el mensaje del sistema de Claude Code.

- Todos los estilos de salida excluyen instrucciones para una salida eficiente (como responder de manera concisa).
- Los estilos de salida personalizados excluyen instrucciones para codificación (como verificar código con pruebas), a menos que `keep-coding-instructions` sea verdadero.
- Todos los estilos de salida tienen sus propias instrucciones personalizadas agregadas al final del mensaje del sistema.
- Todos los estilos de salida activan recordatorios para que Claude se adhiera a las instrucciones del estilo de salida durante la conversación.

Cambiar su estilo de salida

Ejecute `/config` y seleccione **Output style** para elegir un estilo de un menú. Su selección se guarda en `.claude/settings.local.json` en el [nivel de proyecto local](#).

Para establecer un estilo sin el menú, edite el campo `outputStyle` directamente en un archivo de configuración:

```
{
  "outputStyle": "Explanatory"
}
```

Debido a que el estilo de salida se establece en el mensaje del sistema al inicio de la sesión, los cambios surten efecto la próxima vez que inicie una nueva sesión. Esto mantiene el mensaje del sistema estable durante una conversación para que el almacenamiento en caché de prompts pueda reducir la latencia y el costo.

Crear un estilo de salida personalizado

Los estilos de salida personalizados son archivos Markdown con frontmatter y el texto que se agregará al mensaje del sistema:

```
---
name: My Custom Style
description:
  A brief description of what this style does, to be displayed to the user
---

## Custom Style Instructions

You are an interactive CLI tool that helps users with software engineering
tasks. [Your custom instructions here...]

### Specific Behaviors

[Define how the assistant should behave in this style...]
```

Puede guardar estos archivos en el nivel de usuario (`~/claude/output-styles`) o en el nivel de proyecto (`.claude/output-styles`).

Frontmatter

Los archivos de estilo de salida admiten frontmatter para especificar metadatos:

Frontmatter	Propósito	Predeterminado
<code>name</code>	Nombre del estilo de salida, si no es el nombre del archivo	Se hereda del nombre del archivo
<code>description</code>	Descripción del estilo de salida, mostrada en el selector <code>/config</code>	Ninguno
<code>keep-coding-instructions</code>	Si se deben mantener las partes del mensaje del sistema de Claude Code relacionadas con la codificación.	false

Comparaciones con características relacionadas

Estilos de salida vs. CLAUDE.md vs. `--append-system-prompt`

Los estilos de salida “apagan” completamente las partes del mensaje del sistema predeterminado de Claude Code específicas de la ingeniería de software. Ni CLAUDE.md ni `--append-system-prompt` editan el mensaje del sistema predeterminado de Claude Code. CLAUDE.md agrega el contenido como un mensaje de usuario *después* del mensaje del sistema predeterminado de Claude Code. `--append-system-prompt` agrega el contenido al mensaje del sistema.

Estilos de salida vs. [Agents](#)

Los estilos de salida afectan directamente el bucle del agente principal y solo afectan el mensaje del sistema. Los agentes se invocan para manejar tareas específicas y pueden incluir configuraciones adicionales como el modelo a usar, las herramientas disponibles y algo de contexto sobre cuándo usar el agente.

Estilos de salida vs. [Skills](#)

Los estilos de salida modifican cómo responde Claude (formato, tono, estructura) y siempre están activos una vez seleccionados. Skills son prompts específicos de tareas que invoca con `/skill-name` o que Claude carga automáticamente cuando es relevante. Use estilos de salida para preferencias de formato consistentes; use skills para flujos de trabajo y tareas reutilizables.

Acelera las respuestas con el modo rápido

Obtén respuestas más rápidas de Opus 4.6 en Claude Code al activar el modo rápido.

Note:

El modo rápido está en [vista previa de investigación](#). La función, los precios y la disponibilidad pueden cambiar según los comentarios.

El modo rápido es una configuración de alta velocidad para Claude Opus 4.6, haciendo que el modelo sea 2.5 veces más rápido a un costo más alto por token. Actívalo con `/fast` cuando necesites velocidad para trabajo interactivo como iteración rápida o depuración en vivo, y desactívalo cuando el costo sea más importante que la latencia.

El modo rápido no es un modelo diferente. Utiliza el mismo Opus 4.6 con una configuración de API diferente que prioriza la velocidad sobre la eficiencia de costos. Obtienes la misma calidad y capacidades, solo respuestas más rápidas.

Note:

El modo rápido requiere Claude Code v2.1.36 o posterior. Verifica tu versión con `claude --version`.

Lo que debes saber:

- Usa `/fast` para activar o desactivar el modo rápido en Claude Code CLI. También disponible a través de `/fast` en la extensión Claude Code VS Code.
- Los precios del modo rápido para Opus 4.6 comienzan en \$30/150 MTok. El modo rápido está disponible con un descuento del 50% para todos los planes hasta las 11:59 p.m. PT del 16 de febrero.
- Disponible para todos los usuarios de Claude Code en planes de suscripción (Pro/Max/Team/Enterprise) y Claude Console.
- Para los usuarios de Claude Code en planes de suscripción (Pro/Max/Team/Enterprise), el modo rápido está disponible solo a través de uso adicional y no está incluido en los límites de velocidad de la suscripción.

Esta página cubre cómo [activar el modo rápido](#), su [compensación de costos](#), [cuándo usarlo](#), [requisitos](#), [opción de participación por sesión](#), y [comportamiento de límite de velocidad](#).

Activar el modo rápido

Activa el modo rápido de cualquiera de estas formas:

- Escribe `/fast` y presiona Tab para activar o desactivar
- Establece `"fastMode": true` en tu [archivo de configuración de usuario](#)

De forma predeterminada, el modo rápido persiste entre sesiones. Los administradores pueden configurar el modo rápido para que se reinicie cada sesión. Consulta [opción de participación por sesión](#) para obtener más detalles.

Para la mejor eficiencia de costos, habilita el modo rápido al inicio de una sesión en lugar de cambiar a mitad de la conversación. Consulta [entender la compensación de costos](#) para obtener más detalles.

Cuando habilitas el modo rápido:

- Si estás en un modelo diferente, Claude Code cambia automáticamente a Opus 4.6
- Verás un mensaje de confirmación: “Fast mode ON”
- Un pequeño icono  aparece junto al prompt mientras el modo rápido está activo
- Ejecuta `/fast` nuevamente en cualquier momento para verificar si el modo rápido está activado o desactivado

Cuando desactivas el modo rápido con `/fast` nuevamente, permaneces en Opus 4.6. El modelo no revierte a tu modelo anterior. Para cambiar a un modelo diferente, usa `/model`.

Entender la compensación de costos

El modo rápido tiene precios por token más altos que el Opus 4.6 estándar:

Modo	Entrada (MTok)	Salida (MTok)
Modo rápido en Opus 4.6 (<200K)	\$30	\$150
Modo rápido en Opus 4.6 (>200K)	\$60	\$225

El modo rápido es compatible con la ventana de contexto extendida de 1M tokens.

Cuando cambias al modo rápido a mitad de la conversación, pagas el precio completo del token de entrada sin caché del modo rápido para todo el contexto de la conversación. Esto cuesta más que si hubieras habilitado el modo rápido desde el inicio.

Decidir cuándo usar el modo rápido

El modo rápido es mejor para trabajo interactivo donde la latencia de respuesta es más importante que el costo:

- Iteración rápida en cambios de código
- Sesiones de depuración en vivo
- Trabajo sensible al tiempo con plazos ajustados

El modo estándar es mejor para:

- Tareas autónomas largas donde la velocidad importa menos
- Procesamiento por lotes o canalizaciones CI/CD
- Cargas de trabajo sensibles al costo

Modo rápido versus nivel de esfuerzo

El modo rápido y el nivel de esfuerzo afectan la velocidad de respuesta, pero de manera diferente:

Configuración	Efecto
Modo rápido	Misma calidad de modelo, latencia más baja, costo más alto
Nivel de esfuerzo más bajo	Menos tiempo de pensamiento, respuestas más rápidas, calidad potencialmente más baja en tareas complejas

Puedes combinar ambos: usa el modo rápido con un [nivel de esfuerzo](#) más bajo para máxima velocidad en tareas sencillas.

Requisitos

El modo rápido requiere todos los siguientes:

- **No disponible en proveedores de nube de terceros:** el modo rápido no está disponible en Amazon Bedrock, Google Vertex AI o Microsoft Azure Foundry. El modo rápido está disponible a través de la API de Anthropic Console y para planes de suscripción de Claude usando uso adicional.
- **Uso adicional habilitado:** tu cuenta debe tener el uso adicional habilitado, lo que permite facturación más allá del uso incluido en tu plan. Para cuentas individuales,

habilita esto en tu [configuración de facturación de Console](#). Para Teams y Enterprise, un administrador debe habilitar el uso adicional para la organización.

Note:

El uso del modo rápido se factura directamente al uso adicional, incluso si tienes uso restante en tu plan. Esto significa que los tokens del modo rápido no cuentan contra el uso incluido de tu plan y se cobran a la tarifa del modo rápido desde el primer token.

- **Habilitación del administrador para Teams y Enterprise:** el modo rápido está deshabilitado de forma predeterminada para organizaciones Teams y Enterprise. Un administrador debe [habilitar explícitamente el modo rápido](#) antes de que los usuarios puedan acceder a él.

Note:

Si tu administrador no ha habilitado el modo rápido para tu organización, el comando `/fast` mostrará “Fast mode has been disabled by your organization.”

Habilitar el modo rápido para tu organización

Los administradores pueden habilitar el modo rápido en:

- **Console** (clientes de API): [Preferencias de Claude Code](#)
- **Claude AI** (Teams y Enterprise): [Admin Settings > Claude Code](#)

Otra opción para desactivar completamente el modo rápido es establecer

`CLAUDE_CODE_DISABLE_FAST_MODE=1`. Consulta [Variables de entorno](#).

Opción de participación por sesión

De forma predeterminada, el modo rápido persiste entre sesiones: si un usuario habilita el modo rápido, permanece activado en futuras sesiones. Los administradores en planes [Teams](#) o [Enterprise](#) pueden evitar esto estableciendo `fastModePerSessionOptIn` en `true` en [configuración administrada](#) o [configuración administrada por servidor](#). Esto hace que cada sesión comience con el modo rápido desactivado, requiriendo que los usuarios lo habiliten explícitamente con `/fast`.

```
{  
  "fastModePerSessionOptIn": true  
}
```

Esto es útil para controlar costos en organizaciones donde los usuarios ejecutan múltiples sesiones concurrentes. Los usuarios aún pueden habilitar el modo rápido con `/fast` cuando necesiten velocidad, pero se reinicia al inicio de cada nueva sesión. La preferencia del modo rápido del usuario aún se guarda, por lo que eliminar esta configuración restaura el comportamiento persistente predeterminado.

Manejar límites de velocidad

El modo rápido tiene límites de velocidad separados del Opus 4.6 estándar. Cuando alcanzas el límite de velocidad del modo rápido o se agotan tus créditos de uso adicional:

1. El modo rápido automáticamente vuelve al Opus 4.6 estándar
2. El icono  se vuelve gris para indicar enfriamiento
3. Continúas trabajando a velocidad y precios estándar
4. Cuando expira el enfriamiento, el modo rápido se vuelve a habilitar automáticamente

Para desactivar el modo rápido manualmente en lugar de esperar el enfriamiento, ejecuta `/fast` nuevamente.

Vista previa de investigación

El modo rápido es una función de vista previa de investigación. Esto significa:

- La función puede cambiar según los comentarios
- La disponibilidad y los precios están sujetos a cambios
- La configuración de API subyacente puede evolucionar

Reporta problemas o comentarios a través de tus canales de soporte habituales de Anthropic.

Ver también

- [Configuración de modelo](#): cambiar modelos y ajustar niveles de esfuerzo
- [Gestionar costos de manera efectiva](#): rastrear el uso de tokens y reducir costos
- [Configuración de línea de estado](#): mostrar información de modelo y contexto

Gestionar costos de manera efectiva

Realice un seguimiento del uso de tokens, establezca límites de gasto del equipo y reduzca los costos de Claude Code con la gestión del contexto, la selección de modelos, la configuración del pensamiento extendido y los hooks de preprocesamiento.

Claude Code consume tokens en cada interacción. Los costos varían según el tamaño de la base de código, la complejidad de la consulta y la duración de la conversación. El costo promedio es de \$6 por desarrollador por día, con costos diarios que se mantienen por debajo de \$12 para el 90% de los usuarios.

Para el uso en equipo, Claude Code cobra por consumo de tokens de API. En promedio, Claude Code cuesta aproximadamente \$100-200 por desarrollador por mes con Sonnet 4.6, aunque hay una gran variación dependiendo de cuántas instancias ejecutan los usuarios y si la están utilizando en automatización.

Esta página cubre cómo [realizar un seguimiento de sus costos](#), [gestionar costos para equipos](#) y [reducir el uso de tokens](#).

Realice un seguimiento de sus costos

Uso del comando `/cost`

Note:

El comando `/cost` muestra el uso de tokens de API y está destinado a usuarios de API. Los suscriptores de Claude Max y Pro tienen el uso incluido en su suscripción, por lo que los datos de `/cost` no son relevantes para fines de facturación. Los suscriptores pueden usar `/stats` para ver patrones de uso.

El comando `/cost` proporciona estadísticas detalladas de uso de tokens para su sesión actual:

Total cost: \$0.55
Total duration (API): 6m 19.7s
Total duration (wall): 6h 33m 10.2s
Total code changes: 0 lines added, 0 lines removed

Gestión de costos para equipos

Cuando utiliza Claude API, puede [establecer límites de gasto del espacio de trabajo](#) en el gasto total del espacio de trabajo de Claude Code. Los administradores pueden [ver informes de costos y uso](#) en la Consola.

Note:

Cuando autentica por primera vez Claude Code con su cuenta de Claude Console, se crea automáticamente un espacio de trabajo llamado “Claude Code” para usted. Este espacio de trabajo proporciona seguimiento y gestión centralizada de costos para todo el uso de Claude Code en su organización. No puede crear claves de API para este espacio de trabajo; es exclusivamente para autenticación y uso de Claude Code.

En Bedrock, Vertex y Foundry, Claude Code no envía métricas desde su nube. Para obtener métricas de costos, varias grandes empresas informaron usar [LiteLLM](#), que es una herramienta de código abierto que ayuda a las empresas a [realizar un seguimiento del gasto por clave](#). Este proyecto no está afiliado con Anthropic y no ha sido auditado por seguridad.

Recomendaciones de límite de velocidad

Al configurar Claude Code para equipos, considere estas recomendaciones de Tokens Por Minuto (TPM) y Solicitudes Por Minuto (RPM) por usuario según el tamaño de su organización:

Tamaño del equipo	TPM por usuario	RPM por usuario
1-5 usuarios	200k-300k	5-7
5-20 usuarios	100k-150k	2.5-3.5
20-50 usuarios	50k-75k	1.25-1.75
50-100 usuarios	25k-35k	0.62-0.87
100-500 usuarios	15k-20k	0.37-0.47

Tamaño del equipo	TPM por usuario	RPM por usuario
500+ usuarios	10k-15k	0.25-0.35

Por ejemplo, si tiene 200 usuarios, podría solicitar 20k TPM para cada usuario, o 4 millones de TPM totales ($200 \times 20,000 = 4$ millones).

El TPM por usuario disminuye a medida que crece el tamaño del equipo porque menos usuarios tienden a usar Claude Code simultáneamente en organizaciones más grandes. Estos límites de velocidad se aplican a nivel de organización, no por usuario individual, lo que significa que los usuarios individuales pueden consumir temporalmente más que su parte calculada cuando otros no están usando activamente el servicio.

Note:

Si anticipa escenarios con uso concurrente inusualmente alto (como sesiones de capacitación en vivo con grupos grandes), es posible que necesite asignaciones de TPM más altas por usuario.

Costos de tokens del equipo de agentes

[Los equipos de agentes](#) generan múltiples instancias de Claude Code, cada una con su propia ventana de contexto. El uso de tokens se escala con el número de compañeros de equipo activos y cuánto tiempo se ejecuta cada uno.

Para mantener los costos del equipo de agentes manejables:

- Use Sonnet para compañeros de equipo. Equilibra capacidad y costo para tareas de coordinación.
- Mantenga los equipos pequeños. Cada compañero de equipo ejecuta su propia ventana de contexto, por lo que el uso de tokens es aproximadamente proporcional al tamaño del equipo.
- Mantenga los prompts de generación enfocados. Los compañeros de equipo cargan CLAUDE.md, servidores MCP y skills automáticamente, pero todo en el prompt de generación se suma a su contexto desde el principio.
- Limpie los equipos cuando el trabajo esté hecho. Los compañeros de equipo activos continúan consumiendo tokens incluso si están inactivos.
- Los equipos de agentes están deshabilitados por defecto. Establezca `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` en su [settings.json](#) o entorno para habilitarlos. Consulte [habilitar equipos de agentes](#).

Reducir el uso de tokens

Los costos de tokens se escalan con el tamaño del contexto: cuanto más contexto procesa Claude, más tokens utiliza. Claude Code optimiza automáticamente los costos a través del almacenamiento en caché de prompts (que reduce costos para contenido repetido como prompts del sistema) y auto-compactación (que resume el historial de conversación cuando se acerca a los límites del contexto).

Las siguientes estrategias lo ayudan a mantener el contexto pequeño y reducir los costos por mensaje.

Gestione el contexto de manera proactiva

Use `/cost` para verificar su uso actual de tokens, o [configure su línea de estado](#) para mostrarla continuamente.

- **Limpie entre tareas:** Use `/clear` para comenzar de nuevo cuando cambie a trabajo no relacionado. El contexto obsoleto desperdicia tokens en cada mensaje posterior. Use `/rename` antes de limpiar para que pueda encontrar fácilmente la sesión más tarde, luego `/resume` para volver a ella.
- **Agregue instrucciones de compactación personalizadas:** `/compact Focus on code samples and API usage` le dice a Claude qué preservar durante la summarización.

También puede personalizar el comportamiento de compactación en su CLAUDE.md:

```
## Compact instructions
```

```
When you are using compact, please focus on test output and code changes
```

Elija el modelo correcto

Sonnet maneja bien la mayoría de tareas de codificación y cuesta menos que Opus. Reserve Opus para decisiones arquitectónicas complejas o razonamiento de múltiples pasos.

Use `/model` para cambiar modelos a mitad de sesión, o establezca un valor predeterminado en `/config`. Para tareas simples de subagent, especifique `model: haiku` en su [configuración de subagent](#).

Reduzca la sobrecarga del servidor MCP

Cada servidor MCP agrega definiciones de herramientas a su contexto, incluso cuando está inactivo. Ejecute `/context` para ver qué está consumiendo espacio.

- **Prefiera herramientas CLI cuando estén disponibles:** Herramientas como `gh`, `aws`, `gcloud` y `sentry-cli` son más eficientes en contexto que los servidores MCP porque no agregan definiciones de herramientas persistentes. Claude puede ejecutar comandos CLI directamente sin la sobrecarga.
- **Deshabilite servidores no utilizados:** Ejecute `/mcp` para ver servidores configurados y deshabilite cualquiera que no esté usando activamente.
- **La búsqueda de herramientas es automática:** Cuando las descripciones de herramientas MCP exceden el 10% de su ventana de contexto, Claude Code automáticamente las difiere y carga herramientas bajo demanda a través de [búsqueda de herramientas](#). Dado que las herramientas diferidas solo entran en contexto cuando se usan realmente, un umbral más bajo significa menos definiciones de herramientas inactivas consumiendo espacio. Establezca un umbral más bajo con `ENABLE_TOOL_SEARCH=auto:<N>` (por ejemplo, `auto:5` se activa cuando las herramientas exceden el 5% de su ventana de contexto).

Instale plugins de inteligencia de código para lenguajes tipados

[Los plugins de inteligencia de código](#) le dan a Claude navegación de símbolos precisa en lugar de búsqueda basada en texto, reduciendo lecturas de archivos innecesarias al explorar código desconocido. Una única llamada “ir a definición” reemplaza lo que de otro modo sería un grep seguido de lectura de múltiples archivos candidatos. Los servidores de lenguaje instalados también reportan errores de tipo automáticamente después de ediciones, por lo que Claude detecta errores sin ejecutar un compilador.

Descargue el procesamiento en hooks y skills

Los [hooks](#) personalizados pueden preprocesar datos antes de que Claude los vea. En lugar de que Claude lea un archivo de registro de 10,000 líneas para encontrar errores, un hook puede buscar `ERROR` y devolver solo las líneas coincidentes, reduciendo el contexto de decenas de miles de tokens a cientos.

Una [skill](#) puede darle a Claude conocimiento de dominio para que no tenga que explorar. Por ejemplo, una skill “codebase-overview” podría describir la arquitectura de su proyecto, directorios clave y convenciones de nomenclatura. Cuando Claude invoca la skill, obtiene este contexto inmediatamente en lugar de gastar tokens leyendo múltiples archivos para entender la estructura.

Por ejemplo, este hook `PreToolUse` filtra la salida de prueba para mostrar solo fallos:

settings.json

Agregue esto a su [settings.json](#) para ejecutar el hook antes de cada comando Bash:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

El hook llama a este script, que verifica si el comando es un ejecutor de pruebas y lo modifica para mostrar solo fallos:

```
#!/bin/bash
input=$(cat)
cmd=$(echo "$input" | jq -r '.tool_input.command')

## If running tests, filter to show only failures
if [[ "$cmd" =~ ^(npm test|pytest|go test) ]]; then
  filtered_cmd="$cmd 2>&1 | grep -A 5 -E '(FAIL|ERROR|error:)' | head -100"
  echo "{\"hookSpecificOutput\":{\"hookEventName\":\"PreToolUse\", \"permissionDecision\":\"allow\", \"updatedInput\":{\"command\":\"$filtered_cmd\"}}}"
else
  echo "{}"
fi
```

Mueva instrucciones de **CLAUDE.md** a **skills**

Su archivo [CLAUDE.md](#) se carga en contexto al inicio de la sesión. Si contiene instrucciones detalladas para flujos de trabajo específicos (como revisiones de PR o migraciones de bases de datos), esos tokens están presentes incluso cuando está haciendo trabajo no relacionado. [Skills](#) se cargan bajo demanda solo cuando se invocan, por lo que mover instrucciones especializadas a skills mantiene su contexto base más pequeño. Apunte a mantener CLAUDE.md bajo aproximadamente ~500 líneas incluyendo solo lo esencial.

Ajuste el pensamiento extendido

El pensamiento extendido está habilitado por defecto con un presupuesto de 31,999 tokens porque mejora significativamente el rendimiento en tareas complejas de planificación y razonamiento. Sin embargo, los tokens de pensamiento se facturan como tokens de salida, por lo que para tareas más simples donde el razonamiento profundo no es necesario, puede reducir costos bajando el [nivel de esfuerzo](#) en `/model` para Opus 4.6, deshabilitando el pensamiento en `/config`, o bajando el presupuesto (por ejemplo, `MAX_THINKING_TOKENS=8000`).

Delegue operaciones detalladas a subagents

Ejecutar pruebas, obtener documentación o procesar archivos de registro puede consumir contexto significativo. Delegue estos a [subagents](#) para que la salida detallada permanezca en el contexto del subagent mientras solo un resumen regresa a su conversación principal.

Gestione los costos del equipo de agentes

Los equipos de agentes usan aproximadamente 7 veces más tokens que sesiones estándar cuando los compañeros de equipo se ejecutan en plan mode, porque cada compañero de equipo mantiene su propia ventana de contexto y se ejecuta como una instancia separada de Claude. Mantenga las tareas del equipo pequeñas y autónomas para limitar el uso de tokens por compañero de equipo. Consulte [equipos de agentes](#) para obtener detalles.

Escriba prompts específicos

Solicitudes vagas como “mejorar esta base de código” desencadenan escaneo amplio. Solicitudes específicas como “agregar validación de entrada a la función de inicio de sesión en auth.ts” permiten que Claude trabaje eficientemente con lecturas de archivos mínimas.

Trabaje eficientemente en tareas complejas

Para trabajo más largo o más complejo, estos hábitos ayudan a evitar tokens desperdiciados por tomar el camino equivocado:

- **Use plan mode para tareas complejas:** Presione Shift+Tab para entrar en [plan mode](#) antes de la implementación. Claude explora la base de código y propone un enfoque para su aprobación, previniendo re-trabajo costoso cuando la dirección inicial es incorrecta.
- **Corrija el curso temprano:** Si Claude comienza a ir en la dirección equivocada, presione Escape para detener inmediatamente. Use `/rewind` o presione Escape dos veces para restaurar la conversación y el código a un checkpoint anterior.
- **Proporcione objetivos de verificación:** Incluya casos de prueba, pegue capturas de pantalla o defina la salida esperada en su prompt. Cuando Claude puede verificar su propio trabajo, detecta problemas antes de que necesite solicitar correcciones.
- **Pruebe incrementalmente:** Escriba un archivo, pruébelo, luego continúe. Esto detecta problemas temprano cuando son baratos de arreglar.

Uso de tokens en segundo plano

Claude Code usa tokens para algunas funcionalidades en segundo plano incluso cuando está inactivo:

- **Summarización de conversación:** Trabajos en segundo plano que resumen conversaciones anteriores para la característica `claude --resume`
- **Procesamiento de comandos:** Algunos comandos como `/cost` pueden generar solicitudes para verificar el estado

Estos procesos en segundo plano consumen una pequeña cantidad de tokens (típicamente menos de \$0.04 por sesión) incluso sin interacción activa.

Comprensión de cambios en el comportamiento de Claude Code

Claude Code recibe actualizaciones regularmente que pueden cambiar cómo funcionan las características, incluido el reporte de costos. Ejecute `claude --version` para verificar su versión actual. Para preguntas específicas de facturación, contacte al soporte de Anthropic a través de su [cuenta de Consola](#). Para implementaciones de equipo, comience con un pequeño grupo piloto para establecer patrones de uso antes de un despliegue más amplio.

Part 5: Extensibility

Referencia de hooks

Referencia para eventos de hooks de Claude Code, esquema de configuración, formatos de entrada/salida JSON, códigos de salida, hooks asincronos, hooks HTTP, hooks de prompt y hooks de herramientas MCP.

Tip:

Para una guía de inicio rápido con ejemplos, consulte [Automatizar flujos de trabajo con hooks](#).

Los hooks son comandos de shell definidos por el usuario, puntos finales HTTP o prompts de LLM que se ejecutan automáticamente en puntos específicos del ciclo de vida de Claude Code. Utilice esta referencia para buscar esquemas de eventos, opciones de configuración, formatos de entrada/salida JSON y características avanzadas como hooks asincronos, hooks HTTP y hooks de herramientas MCP. Si está configurando hooks por primera vez, comience con la [guía](#) en su lugar.

Ciclo de vida del hook

Los hooks se activan en puntos específicos durante una sesión de Claude Code. Cuando se activa un evento y un matcher coincide, Claude Code pasa contexto JSON sobre el evento a su controlador de hook. Para hooks de comando, la entrada llega en stdin. Para hooks HTTP, llega como el cuerpo de la solicitud POST. Su controlador puede entonces inspeccionar la entrada, tomar medidas y opcionalmente devolver una decisión. Algunos eventos se activan una vez por sesión, mientras que otros se activan repetidamente dentro del bucle agéntico:

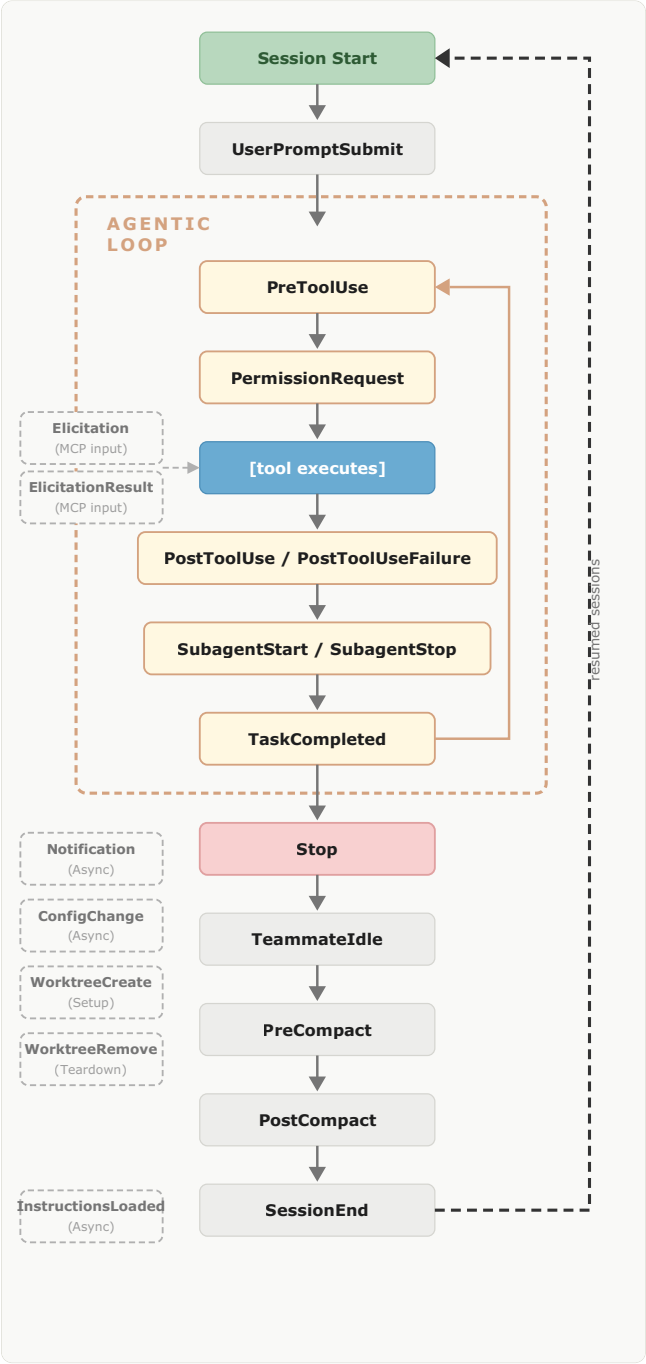


Diagrama del ciclo de vida del hook que muestra la secuencia de hooks desde SessionStart a través del bucle agéntico hasta SessionEnd, con WorktreeCreate, WorktreeRemove e InstructionsLoaded como eventos asíncronos independientes

La tabla a continuación resume cuándo se activa cada evento. La sección [Hook events](#) documenta el esquema de entrada completo y las opciones de control de decisión para cada uno.

Event	When it fires
SessionStart	When a session begins or resumes
UserPromptSubmit	When you submit a prompt, before Claude processes it
PreToolUse	Before a tool call executes. Can block it
PermissionRequest	When a permission dialog appears
PostToolUse	After a tool call succeeds
PostToolUseFailure	After a tool call fails
Notification	When Claude Code sends a notification
SubagentStart	When a subagent is spawned
SubagentStop	When a subagent finishes
Stop	When Claude finishes responding
TeammateIdle	When an agent team teammate is about to go idle
TaskCompleted	When a task is being marked as completed
InstructionsLoaded	When a CLAUDE.md or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
ConfigChange	When a configuration file changes during a session
WorktreeCreate	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
WorktreeRemove	When a worktree is being removed, either at session exit or when a subagent finishes
PreCompact	Before context compaction

Event	When it fires
PostCompact	After context compaction completes
Elicitation	When an MCP server requests user input during a tool call
ElicitationResult	After a user responds to an MCP elicitation, before the response is sent back to the server
SessionEnd	When a session terminates

Cómo se resuelve un hook

Para ver cómo encajan estas piezas, considere este hook `PreToolUse` que bloquea comandos de shell destructivos. El hook ejecuta `block-rm.sh` antes de cada llamada a la herramienta Bash:

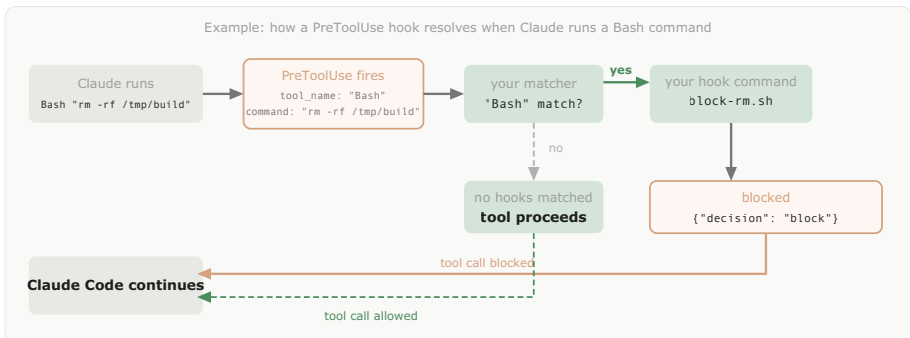
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

El script lee la entrada JSON desde stdin, extrae el comando y devuelve una `permissionDecision` de `"deny"` si contiene `rm -rf`:

```
#!/bin/bash
## .claude/hooks/block-rm.sh
COMMAND=$(jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q 'rm -rf'; then
  jq -n '{
    hookSpecificOutput: {
      hookEventName: "PreToolUse",
      permissionDecision: "deny",
      permissionDecisionReason: "Destructive command blocked by hook"
    }
  }'
else
  exit 0 # allow the command
fi
```

Ahora suponga que Claude Code decide ejecutar `Bash "rm -rf /tmp/build"`. Esto es lo que sucede:



Flujo de resolución de hook: se activa el evento PreToolUse, el matcher verifica la coincidencia de Bash, se ejecuta el controlador de hook, el resultado se devuelve a Claude Code

Step 1: Se activa el evento

El evento `PreToolUse` se activa. Claude Code envía la entrada de la herramienta como JSON en stdin al hook:

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: El matcher verifica

El matcher `"Bash"` coincide con el nombre de la herramienta, por lo que se ejecuta `block-rm.sh`. Si omite el matcher o usa `"*"`, el hook se ejecuta en cada ocurrencia del evento. Los hooks solo se omiten cuando se define un matcher y no coincide.

Step 3: Se ejecuta el controlador de hook

El script extrae `"rm -rf /tmp/build"` de la entrada y encuentra `rm -rf`, por lo que imprime una decisión en stdout:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

Si el comando hubiera sido seguro (como `npm test`), el script habría alcanzado `exit 0` en su lugar, lo que le dice a Claude Code que permita la llamada a la herramienta sin más acciones.

Step 4: Claude Code actúa sobre el resultado

Claude Code lee la decisión JSON, bloquea la llamada a la herramienta y muestra a Claude la razón.

La sección [Configuration](#) a continuación documenta el esquema completo, y cada sección [hook event](#) documenta qué entrada recibe su comando y qué salida puede devolver.

Configuración

Los hooks se definen en archivos de configuración JSON. La configuración tiene tres niveles de anidamiento:

1. Elija un [hook event](#) al que responder, como `PreToolUse` o `Stop`
2. Agregue un [matcher group](#) para filtrar cuándo se activa, como “solo para la herramienta Bash”
3. Defina uno o más [hook handlers](#) para ejecutar cuando coincida

Consulte [Cómo se resuelve un hook](#) arriba para un recorrido completo con un ejemplo anotado.

Note:

Esta página utiliza términos específicos para cada nivel: **hook event** para el punto del ciclo de vida, **matcher group** para el filtro y **hook handler** para el comando de shell, punto final HTTP, prompt o agente que se ejecuta. “Hook” por sí solo se refiere a la característica general.

Ubicaciones de hooks

Dónde define un hook determina su alcance:

Ubicación	Alcance	Compatible
<code>~/.claude/settings.json</code>	Todos sus proyectos	No, local en su máquina
<code>.claude/settings.json</code>	Proyecto único	Sí, puede ser confirmado en el repositorio
<code>.claude/settings.local.json</code>	Proyecto único	No, ignorado por git
Configuración de política administrada	Toda la organización	Sí, controlado por administrador
<code>Plugin hooks/hooks.json</code>	Cuando el plugin está habilitado	Sí, incluido con el plugin
<code>Skill</code> o <code>agent</code> frontmatter	Mientras el componente está activo	Sí, definido en el archivo del componente

Para obtener detalles sobre la resolución de archivos de configuración, consulte [settings](#). Los administradores empresariales pueden usar `allowManagedHooksOnly` para bloquear hooks de usuario, proyecto y plugin. Consulte [Hook configuration](#).

Patrones de matcher

El campo `matcher` es una cadena regex que filtra cuándo se activan los hooks. Use `"*"`, `""` u omita `matcher` completamente para coincidir con todas las ocurrencias. Cada tipo de evento coincide en un campo diferente:

Evento	En qué filtra el matcher	Valores de matcher de ejemplo
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	nombre de la herramienta	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	cómo comenzó la sesión	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	por qué terminó la sesión	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_disabled</code> , <code>other</code>
<code>Notification</code>	tipo de notificación	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>
<code>SubagentStart</code>	tipo de agente	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> o nombres de agentes personalizados
<code>PreCompact</code>	qué desencadenó la compactación	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	tipo de agente	los mismos valores que <code>SubagentStart</code>
<code>ConfigChange</code>	fuelle de configuración	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>
<code>UserPromptSubmit</code> , <code>Stop</code> , <code>TeammateIdle</code> , <code>TaskCompleted</code> , <code>WorktreeCreate</code> , <code>WorktreeRemove</code> , <code>InstructionsLoaded</code>	sin soporte de matcher	siempre se activa en cada ocurrencia

El matcher es un regex, por lo que `Edit|Write` coincide con cualquiera de las herramientas y `Notebook.*` coincide con cualquier herramienta que comience con Notebook. El matcher se ejecuta contra un campo de la [entrada JSON](#) que Claude Code envía a su hook en stdin. Para eventos de herramientas, ese campo es `tool_name`. Cada sección [hook event](#) enumera el conjunto completo de valores de matcher y el esquema de entrada para ese evento.

Este ejemplo ejecuta un script de linting solo cuando Claude escribe o edita un archivo:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove` e `InstructionsLoaded` no admiten matchers y siempre se activan en cada ocurrencia. Si agrega un campo `matcher` a estos eventos, se ignora silenciosamente.

Coincidir herramientas MCP

Las herramientas del servidor [MCP](#) aparecen como herramientas normales en eventos de herramientas (`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`), por lo que puede hacerlas coincidir de la misma manera que cualquier otro nombre de herramienta.

Las herramientas MCP siguen el patrón de nomenclatura `mcp__<server>__<tool>`, por ejemplo:

- `mcp__memory__create_entities`: herramienta crear entidades del servidor Memory
- `mcp__filesystem__read_file`: herramienta leer archivo del servidor Filesystem

- `mcp_github__search_repositories` : herramienta de búsqueda del servidor GitHub

Use patrones regex para dirigirse a herramientas MCP específicas o grupos de herramientas:

- `mcp__memory__.*` coincide con todas las herramientas del servidor `memory`
- `mcp__.*__write.*` coincide con cualquier herramienta que contenga “write” de cualquier servidor

Este ejemplo registra todas las operaciones del servidor de memoria y valida operaciones de escritura de cualquier servidor MCP:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Memory operation initiated' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}
```

Campos del controlador de hook

Cada objeto en el array `hooks` interno es un controlador de hook: el comando de shell, punto final HTTP, prompt de LLM o agente que se ejecuta cuando el matcher coincide. Hay cuatro tipos:

- **Command hooks** (`type: "command"`): ejecuta un comando de shell. Su script recibe la [entrada JSON](#) del evento en stdin y comunica los resultados a través de códigos de salida y stdout.
- **HTTP hooks** (`type: "http"`): envía la entrada JSON del evento como una solicitud HTTP POST a una URL. El punto final comunica los resultados a través del cuerpo de la respuesta usando el mismo [formato de salida JSON](#) que los hooks de comando.
- **Prompt hooks** (`type: "prompt"`): envía un prompt a un modelo Claude para evaluación de un solo turno. El modelo devuelve una decisión sí/no como JSON. Consulte [Prompt-based hooks](#).
- **Agent hooks** (`type: "agent"`): genera un subagente que puede usar herramientas como Read, Grep y Glob para verificar condiciones antes de devolver una decisión. Consulte [Agent-based hooks](#).

Campos comunes

Estos campos se aplican a todos los tipos de hooks:

Campo	Requerido	Descripción
<code>type</code>	sí	<code>"command"</code> , <code>"http"</code> , <code>"prompt"</code> o <code>"agent"</code>
<code>timeout</code>	no	Segundos antes de cancelar. Valores predeterminados: 600 para comando, 30 para prompt, 60 para agente
<code>statusMessage</code>	no	Mensaje de spinner personalizado mostrado mientras se ejecuta el hook
<code>once</code>	no	Si es <code>true</code> , se ejecuta solo una vez por sesión y luego se elimina. Solo skills, no agentes. Consulte Hooks in skills and agents

Campos de comando hook

Además de los [campos comunes](#), los hooks de comando aceptan estos campos:

Campo	Requerido	Descripción
<code>command</code>	sí	Comando de shell a ejecutar
<code>async</code>	no	Si es <code>true</code> , se ejecuta en segundo plano sin bloquear. Consulte Run hooks in the background

Campos de hook HTTP

Además de los [campos comunes](#), los hooks HTTP aceptan estos campos:

Campo	Requerido	Descripción
<code>url</code>	sí	URL a la que enviar la solicitud POST
<code>headers</code>	no	Encabezados HTTP adicionales como pares clave-valor. Los valores admiten interpolación de variables de entorno usando la sintaxis <code>\$VAR_NAME</code> o <code>\${VAR_NAME}</code> . Solo se resuelven las variables enumeradas en <code>allowedEnvVars</code>
<code>allowedEnvVars</code>	no	Lista de nombres de variables de entorno que pueden interpolarse en valores de encabezado. Las referencias a variables no enumeradas se reemplazan con cadenas vacías. Requerido para que funcione cualquier interpolación de variable de entorno

Claude Code envía la [entrada JSON](#) del hook como el cuerpo de la solicitud POST con `Content-Type: application/json`. El cuerpo de la respuesta usa el mismo [formato de salida JSON](#) que los hooks de comando.

El manejo de errores difiere de los hooks de comando: las respuestas que no son 2xx, los fallos de conexión y los tiempos de espera agotados producen errores que no bloquean y permiten que la ejecución continúe. Para bloquear una llamada a herramienta o denegar un permiso, devuelva una respuesta 2xx con un cuerpo JSON que contenga `decision: "block"` o un `hookSpecificOutput` con `permissionDecision: "deny"`.

Este ejemplo envía eventos `PreToolUse` a un servicio de validación local, autenticándose con un token de la variable de entorno `MY_TOKEN`:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/pre-tool-use",
            "timeout": 30,
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

Note:

Los hooks HTTP deben configurarse editando JSON de configuración directamente. El menú interactivo `/hooks` solo admite agregar hooks de comando.

Campos de hook de prompt y agente

Además de los [campos comunes](#), los hooks de prompt y agente aceptan estos campos:

Campo	Requerido	Descripción
<code>prompt</code>	sí	Texto del prompt a enviar al modelo. Use <code>\$ARGUMENTS</code> como marcador de posición para la entrada JSON del hook
<code>model</code>	no	Modelo a usar para evaluación. Por defecto es un modelo rápido

Todos los hooks coincidentes se ejecutan en paralelo, y los controladores idénticos se deduplicarán automáticamente. Los hooks de comando se deduplicarán por cadena de comando, y los hooks HTTP se deduplicarán por URL. Los controladores se ejecutan en el directorio actual con el entorno de Claude Code. La variable de entorno `$CLAUDE_CODE_REMOTE` se establece en `"true"` en entornos web remotos y no se establece en la CLI local.

Referenciar scripts por ruta

Use variables de entorno para referenciar scripts de hook relativos a la raíz del proyecto o plugin, independientemente del directorio de trabajo cuando se ejecuta el hook:

- `$CLAUDE_PROJECT_DIR` : la raíz del proyecto. Envuelva entre comillas para manejar rutas con espacios.
- `${CLAUDE_PLUGIN_ROOT}` : el directorio raíz del plugin, para scripts incluidos con un [plugin](#).

Scripts de proyecto

Este ejemplo usa `$CLAUDE_PROJECT_DIR` para ejecutar un verificador de estilo desde el directorio `.claude/hooks/` del proyecto después de cualquier llamada a herramienta `Write` o `Edit` :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}

```

Scripts de plugin

Defina hooks de plugin en `hooks/hooks.json` con un campo `description` opcional de nivel superior. Cuando se habilita un plugin, sus hooks se fusionan con sus hooks de usuario y proyecto.

Este ejemplo ejecuta un script de formato incluido con el plugin:

```

{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

Consulte la [referencia de componentes de plugin](#) para obtener detalles sobre cómo crear hooks de plugin.

Hooks en skills y agentes

Además de archivos de configuración y plugins, los hooks pueden definirse directamente en [skills](#) y [subagentes](#) usando frontmatter. Estos hooks están limitados al ciclo de vida del componente y solo se ejecutan cuando ese componente está activo.

Se admiten todos los eventos de hook. Para subagentes, los hooks **Stop** se convierten automáticamente a **SubagentStop** ya que ese es el evento que se activa cuando un subagente se completa.

Los hooks usan el mismo formato de configuración que los hooks basados en configuración pero están limitados a la vida útil del componente y se limpian cuando finaliza.

Esta skill define un hook **PreToolUse** que ejecuta un script de validación de seguridad antes de cada comando **Bash** :


```

---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---

```

Los agentes usan el mismo formato en su frontmatter YAML.

El menú `/hooks`

Escriba `/hooks` en Claude Code para abrir el administrador de hooks interactivo, donde puede ver, agregar y eliminar hooks sin editar archivos de configuración directamente. Para un recorrido paso a paso, consulte [Set up your first hook](#) en la guía.

Cada hook en el menú está etiquetado con un prefijo de corchete que indica su fuente:

- `[User]` : de `~/.claude/settings.json`
- `[Project]` : de `.claude/settings.json`
- `[Local]` : de `.claude/settings.local.json`
- `[Plugin]` : de `hooks/hooks.json` de un plugin, solo lectura

Deshabilitar o eliminar hooks

Para eliminar un hook, elimine su entrada del archivo de configuración JSON o use el menú `/hooks` y seleccione el hook para eliminarlo.

Para deshabilitar temporalmente todos los hooks sin eliminarlos, establezca `"disableAllHooks": true` en su archivo de configuración o use el botón de alternancia en el menú `/hooks`. No hay forma de deshabilitar un hook individual mientras se mantiene en la configuración.

La configuración `disableAllHooks` respeta la jerarquía de configuración administrada. Si un administrador ha configurado hooks a través de configuración de política administrada, `disableAllHooks` establecido en configuración de usuario, proyecto o local no puede deshabilitar esos hooks administrados. Solo `disableAllHooks` establecido en el nivel de configuración administrada puede deshabilitar hooks administrados.

Las ediciones directas de hooks en archivos de configuración no tienen efecto inmediato. Claude Code captura una instantánea de hooks al inicio y la usa durante toda la sesión. Esto evita que modificaciones de hooks maliciosas o accidentales tengan efecto a mitad de sesión sin su revisión. Si los hooks se modifican externamente, Claude Code le advierte y requiere revisión en el menú `/hooks` antes de que los cambios se apliquen.

Entrada y salida de hook

Los hooks de comando reciben datos JSON a través de stdin y comunican resultados a través de códigos de salida, stdout y stderr. Los hooks HTTP reciben el mismo JSON que el cuerpo de la solicitud POST y comunican resultados a través del cuerpo de la respuesta HTTP. Esta sección cubre campos y comportamiento comunes a todos los eventos. Cada sección de evento bajo [Hook events](#) incluye su esquema de entrada específico y opciones de control de decisión.

Campos de entrada comunes

Todos los eventos de hook reciben estos campos como JSON, además de campos específicos del evento documentados en cada sección [hook event](#). Para hooks de comando, este JSON llega a través de stdin. Para hooks HTTP, llega como el cuerpo de la solicitud POST.

Campo	Descripción
<code>session_id</code>	Identificador de sesión actual
<code>transcript_path</code>	Ruta al JSON de conversación
<code>cwd</code>	Directorio de trabajo actual cuando se invoca el hook
<code>permission_mode</code>	Modo de permiso actual: <code>"default"</code> , <code>"plan"</code> , <code>"acceptEdits"</code> , <code>"dontAsk"</code> o <code>"bypassPermissions"</code>
<code>hook_event_name</code>	Nombre del evento que se activó

Cuando se ejecuta con `--agent` o dentro de un subagente, se incluyen dos campos adicionales:

Campo	Descripción
<code>agent_id</code>	Identificador único para el subagente. Presente solo cuando el hook se activa dentro de una llamada de subagente. Úselo para distinguir llamadas de hook de subagente de llamadas de hilo principal.

Campo	Descripción
<code>agent_type</code>	Nombre del agente (por ejemplo, <code>"Explore"</code> o <code>"security-reviewer"</code>). Presente cuando la sesión usa <code>--agent</code> o el hook se activa dentro de un subagente. Para subagentes, el tipo del subagente tiene precedencia sobre el valor <code>--agent</code> de la sesión.

Por ejemplo, un hook `PreToolUse` para un comando Bash recibe esto en stdin:

```
{
  "session_id": "abc123",
  "transcript_path": "/home/user/.claude/projects/.../transcript.jsonl",
  "cwd": "/home/user/my-project",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test"
  }
}
```

Los campos `tool_name` y `tool_input` son específicos del evento. Cada sección [hook event](#) documenta los campos adicionales para ese evento.

Salida de código de salida

El código de salida de su comando de hook le dice a Claude Code si la acción debe proceder, ser bloqueada o ser ignorada.

Exit 0 significa éxito. Claude Code analiza stdout para [campos de salida JSON](#). La salida JSON solo se procesa en exit 0. Para la mayoría de eventos, stdout solo se muestra en modo detallado (`Ctrl+0`). Las excepciones son `UserPromptSubmit` y `SessionStart` , donde stdout se agrega como contexto que Claude puede ver y actuar.

Exit 2 significa un error de bloqueo. Claude Code ignora stdout y cualquier JSON en él. En su lugar, el texto de stderr se devuelve a Claude como un mensaje de error. El efecto depende del evento: `PreToolUse` bloquea la llamada a herramienta, `UserPromptSubmit` rechaza el prompt, y así sucesivamente. Consulte [exit code 2 behavior](#) para la lista completa.

Cualquier otro código de salida es un error que no bloquea. stderr se muestra en modo detallado (`Ctrl+0`) y la ejecución continúa.

Por ejemplo, un script de comando de hook que bloquea comandos Bash peligrosos:

```
#!/bin/bash
## Lee la entrada JSON desde stdin, verifica el comando
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" = rm* ]]; then
    echo "Blocked: rm commands are not allowed" >&2
    exit 2 # Blocking error: tool call is prevented
fi

exit 0 # Success: tool call proceeds
```

Comportamiento del código de salida 2 por evento

El código de salida 2 es la forma en que un hook señala “detente, no hagas esto”. El efecto depende del evento, porque algunos eventos representan acciones que pueden bloquearse (como una llamada a herramienta que aún no ha sucedido) y otros representan cosas que ya sucedieron o no pueden prevenirse.

Evento de hook	¿Puede bloquear?	Qué sucede en exit 2
PreToolUse	Sí	Bloquea la llamada a herramienta
PermissionRequest	Sí	Deniega el permiso
UserPromptSubmit	Sí	Bloquea el procesamiento del prompt y borra el prompt
Stop	Sí	Evita que Claude se detenga, continúa la conversación
SubagentStop	Sí	Evita que el subagente se detenga
TeammateIdle	Sí	Evita que el compañero se quede inactivo (el compañero continúa trabajando)

Evento de hook	¿Puede bloquear?	Qué sucede en exit 2
<code>TaskCompleted</code>	Sí	Evita que la tarea se marque como completada
<code>ConfigChange</code>	Sí	Bloquea que el cambio de configuración tenga efecto (excepto <code>policy_settings</code>)
<code>PostToolUse</code>	No	Muestra stderr a Claude (la herramienta ya se ejecutó)
<code>PostToolUseFailure</code>	No	Muestra stderr a Claude (la herramienta ya falló)
<code>Notification</code>	No	Muestra stderr solo al usuario
<code>SubagentStart</code>	No	Muestra stderr solo al usuario
<code>SessionStart</code>	No	Muestra stderr solo al usuario
<code>SessionEnd</code>	No	Muestra stderr solo al usuario
<code>PreCompact</code>	No	Muestra stderr solo al usuario
<code>WorktreeCreate</code>	Sí	Cualquier código de salida distinto de cero causa que la creación de worktree falle
<code>WorktreeRemove</code>	No	Los fallos se registran solo en modo de depuración
<code>InstructionsLoaded</code>	No	El código de salida se ignora

Manejo de respuesta HTTP

Los hooks HTTP usan códigos de estado HTTP y cuerpos de respuesta en lugar de códigos de salida y stdout:

- **2xx con un cuerpo vacío:** éxito, equivalente a código de salida o sin salida
- **2xx con un cuerpo de texto plano:** éxito, el texto se agrega como contexto
- **2xx con un cuerpo JSON:** éxito, analizado usando el mismo esquema [JSON output](#) que los hooks de comando
- **Estado que no es 2xx:** error que no bloquea, la ejecución continúa
- **Fallo de conexión o tiempo de espera agotado:** error que no bloquea, la ejecución continúa

A diferencia de los hooks de comando, los hooks HTTP no pueden señalar un error de bloqueo solo a través de códigos de estado. Para bloquear una llamada a herramienta o denegar un permiso, devuelva una respuesta 2xx con un cuerpo JSON que contenga los campos de decisión apropiados.

Salida JSON

Los códigos de salida le permiten permitir o bloquear, pero la salida JSON le da un control más granular. En lugar de salir con código 2 para bloquear, salga o e imprima un objeto JSON en stdout. Claude Code lee campos específicos de ese JSON para controlar el comportamiento, incluido [decision control](#) para bloquear, permitir o escalar al usuario.

Note:

Debe elegir un enfoque por hook, no ambos: use códigos de salida solos para señalización, o salga o e imprima JSON para control estructurado. Claude Code solo procesa JSON en exit o. Si sale 2, cualquier JSON se ignora.

El stdout de su hook debe contener solo el objeto JSON. Si su perfil de shell imprime texto al inicio, puede interferir con el análisis JSON. Consulte [JSON validation failed](#) en la guía de solución de problemas.

El objeto JSON admite tres tipos de campos:

- **Campos universales** como `continue` funcionan en todos los eventos. Estos se enumeran en la tabla a continuación.
- **decision y reason de nivel superior** son utilizados por algunos eventos para bloquear o proporcionar retroalimentación.

- `hookSpecificOutput` es un objeto anidado para eventos que necesitan control más rico. Requiere un campo `hookEventName` establecido en el nombre del evento.

Campo	Predeterminado	Descripción
<code>continue</code>	<code>true</code>	Si es <code>false</code> , Claude detiene el procesamiento completamente después de que se ejecuta el hook. Tiene precedencia sobre cualquier campo de decisión específico del evento
<code>stopReason</code>	ninguno	Mensaje mostrado al usuario cuando <code>continue</code> es <code>false</code> . No se muestra a Claude
<code>suppressOutput</code>	<code>false</code>	Si es <code>true</code> , oculta stdout de la salida del modo detallado
<code>systemMessage</code>	ninguno	Mensaje de advertencia mostrado al usuario

Para detener Claude completamente independientemente del tipo de evento:

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

Control de decisión

No todos los eventos admiten bloqueo o control de comportamiento a través de JSON. Los eventos que lo hacen cada uno usan un conjunto diferente de campos para expresar esa decisión. Use esta tabla como referencia rápida antes de escribir un hook:

Eventos	Patrón de decisión	Campos clave
UserPromptSubmit, PostToolUse, PostToolUseFailure, Stop, SubagentStop, ConfigChange	<code>decision</code> de nivel superior	<code>decision: "block", reason</code>

Eventos	Patrón de decisión	Campos clave
TeammateIdle, TaskCompleted	Código de salida o <code>continue: false</code>	El código de salida 2 bloquea la acción con retroalimentación de stderr. JSON <code>{"continue": false, "stopReason": "...}"</code> también detiene al compañero completamente, coincidiendo con el comportamiento del hook <code>Stop</code>
PreToolUse	<code>hookSpecificOutput</code>	<code>permissionDecision</code> (allow/deny/ask), <code>permissionDecisionReason</code>
PermissionRequest	<code>hookSpecificOutput</code>	<code>decision.behavior</code> (allow/deny)
WorktreeCreate	ruta stdout	El hook imprime la ruta absoluta del worktree creado. La salida que no es cero falla la creación
WorktreeRemove, Notification, SessionEnd, PreCompact, InstructionsLoaded	Ninguno	Sin control de decisión. Se usa para efectos secundarios como registro o limpieza

Aquí hay ejemplos de cada patrón en acción:

Decisión de nivel superior

Utilizado por `UserPromptSubmit`, `PostToolUse`, `PostToolUseFailure`, `Stop`, `SubagentStop` y `ConfigChange`. El único valor es `"block"`. Para permitir que la acción continúe, omita `decision` de su JSON, o salga o sin ningún JSON en absoluto:

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

Usa `hookSpecificOutput` para control más rico: permitir, denegar o escalar al usuario. También puede modificar la entrada de la herramienta antes de que se ejecute o inyectar contexto adicional para Claude. Consulte [PreToolUse decision control](#) para el conjunto completo de opciones.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

Usa `hookSpecificOutput` para permitir o denegar una solicitud de permiso en nombre del usuario. Al permitir, también puede modificar la entrada de la herramienta o aplicar reglas de permiso para que el usuario no sea solicitado nuevamente. Consulte [Permission-Request decision control](#) para el conjunto completo de opciones.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Para ejemplos extendidos incluyendo validación de comandos Bash, filtrado de prompts y scripts de aprobación automática, consulte [What you can automate](#) en la guía y la [Bash command validator reference implementation](#).

Eventos de hook

Cada evento corresponde a un punto en el ciclo de vida de Claude Code donde los hooks pueden ejecutarse. Las secciones a continuación están ordenadas para coincidir con el ciclo de vida: desde la configuración de sesión a través del bucle agéntico hasta el final de la sesión. Cada sección describe cuándo se activa el evento, qué matchers admite, la entrada JSON que recibe y cómo controlar el comportamiento a través de la salida.

SessionStart

Se ejecuta cuando Claude Code inicia una nueva sesión o reanuda una sesión existente. Útil para cargar contexto de desarrollo como problemas existentes o cambios recientes en su base de código, o configurar variables de entorno. Para contexto estático que no requiere un script, use [CLAUDE.md](#) en su lugar.

SessionStart se ejecuta en cada sesión, así que mantenga estos hooks rápidos. Solo se admiten hooks `type: "command"`.

El valor del matcher corresponde a cómo se inició la sesión:

Matcher	Cuándo se activa
<code>startup</code>	Nueva sesión
<code>resume</code>	<code>--resume</code> , <code>--continue</code> o <code>/resume</code>
<code>clear</code>	<code>/clear</code>
<code>compact</code>	Compactación automática o manual

Entrada de SessionStart

Además de los [campos de entrada comunes](#), los hooks SessionStart reciben `source`, `model` y opcionalmente `agent_type`. El campo `source` indica cómo comenzó la sesión: `"startup"` para nuevas sesiones, `"resume"` para sesiones reanudadas, `"clear"` después de `/clear` o `"compact"` después de compactación. El campo `model` contiene el identificador del modelo. Si inicia Claude Code con `claude --agent <name>`, un campo `agent_type` contiene el nombre del agente.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "SessionStart",
  "source": "startup",
  "model": "claude-sonnet-4-6"
}
```

Control de decisión de SessionStart

Cualquier texto que su script de hook imprima en stdout se agrega como contexto para Claude. Además de los [campos de salida JSON](#) disponibles para todos los hooks, puede devolver estos campos específicos del evento:

Campo	Descripción
<code>additionalContext</code>	Cadena agregada al contexto de Claude. Los valores de múltiples hooks se concatenan

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

Persistir variables de entorno

Los hooks SessionStart tienen acceso a la variable de entorno `CLAUDE_ENV_FILE`, que proporciona una ruta de archivo donde puede persistir variables de entorno para comandos Bash posteriores.

Para establecer variables de entorno individuales, escriba declaraciones `export` en `CLAUDE_ENV_FILE`. Use `append (>>)` para preservar variables establecidas por otros hooks:

```
#!/bin/bash

if [ -n "$CLAUDE_ENV_FILE" ]; then
  echo 'export NODE_ENV=production' >> "$CLAUDE_ENV_FILE"
  echo 'export DEBUG_LOG=true' >> "$CLAUDE_ENV_FILE"
  echo 'export PATH="$PATH:./node_modules/.bin"' >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

Para capturar todos los cambios de entorno de comandos de configuración, compare las variables exportadas antes y después:

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## Run your setup commands that modify the environment
source ~/.nvm/nvm.sh
nvm use 20

if [ -n "$CLAUDE_ENV_FILE" ]; then
  ENV_AFTER=$(export -p | sort)
  comm -13 <(echo "$ENV_BEFORE") <(echo "$ENV_AFTER") >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

Cualquier variable escrita en este archivo estará disponible en todos los comandos Bash posteriores que Claude Code ejecute durante la sesión.

Note:

`CLAUDE_ENV_FILE` está disponible para hooks `SessionStart`. Otros tipos de hooks no tienen acceso a esta variable.

InstructionsLoaded

Se activa cuando se carga un archivo `CLAUDE.md` o `.claude/rules/*.md` en contexto. Este evento se activa al inicio de la sesión para archivos cargados con entusiasmo y nuevamente más tarde cuando se cargan archivos de forma perezosa, por ejemplo cuando Claude accede a un subdirectorio que contiene un `CLAUDE.md` anidado o cuando reglas condicionales con frontmatter `paths:` coinciden. El hook no admite bloqueo o control de decisión. Se ejecuta de forma asíncronica con fines de observabilidad.

InstructionsLoaded no admite matchers y se activa en cada ocurrencia de carga.

Entrada de InstructionsLoaded

Además de los [campos de entrada comunes](#), los hooks InstructionsLoaded reciben estos campos:

Campo	Descripción
<code>file_path</code>	Ruta absoluta al archivo de instrucciones que se cargó
<code>memory_type</code>	Alcance del archivo: <code>"User"</code> , <code>"Project"</code> , <code>"Local"</code> o <code>"Managed"</code>
<code>load_reason</code>	Por qué se cargó el archivo: <code>"session_start"</code> , <code>"nested_traversal"</code> , <code>"path_glob_match"</code> o <code>"include"</code>
<code>globs</code>	Patrones de glob de ruta del frontmatter <code>paths:</code> del archivo, si los hay. Presente solo para cargas <code>path_glob_match</code>
<code>trigger_file_path</code>	Ruta al archivo cuyo acceso desencadenó esta carga, para cargas perezosas
<code>parent_file_path</code>	Ruta al archivo de instrucciones padre que incluyó este, para cargas <code>include</code>

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "permission_mode": "default",
  "hook_event_name": "InstructionsLoaded",
  "file_path": "/Users/my-project/CLAUDE.md",
  "memory_type": "Project",
  "load_reason": "session_start"
}
```

Control de decisión de InstructionsLoaded

Los hooks InstructionsLoaded no tienen control de decisión. No pueden bloquear o modificar la carga de instrucciones. Use este evento para registro de auditoría, seguimiento de cumplimiento u observabilidad.

UserPromptSubmit

Se ejecuta cuando el usuario envía un prompt, antes de que Claude lo procese. Esto le permite agregar contexto adicional basado en el prompt/conversación, validar prompts o bloquear ciertos tipos de prompts.

Entrada de UserPromptSubmit

Además de los [campos de entrada comunes](#), los hooks UserPromptSubmit reciben el campo `prompt` que contiene el texto que el usuario envió.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "UserPromptSubmit",
  "prompt": "Write a function to calculate the factorial of a number"
}
```

Control de decisión de UserPromptSubmit

Los hooks `UserPromptSubmit` pueden controlar si se procesa un prompt de usuario y agregar contexto. Todos los [campos de salida JSON](#) están disponibles.

Hay dos formas de agregar contexto a la conversación en código de salida o:

- **Stdout de texto plano:** cualquier texto que no sea JSON escrito en stdout se agrega como contexto
- **JSON con `additionalContext`:** use el formato JSON a continuación para más control. El campo `additionalContext` se agrega como contexto

El stdout plano se muestra como salida de hook en la transcripción. El campo `additionalContext` se agrega de forma más discreta.

Para bloquear un prompt, devuelva un objeto JSON con `decision` establecido en `"block"`:

Campo	Descripción
<code>decision</code>	<code>"block"</code> evita que el prompt se procese y lo borra del contexto. Omita para permitir que el prompt continúe
<code>reason</code>	Se muestra al usuario cuando <code>decision</code> es <code>"block"</code> . No se agrega al contexto
<code>additionalContext</code>	Cadena agregada al contexto de Claude

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

Note:

El formato JSON no es necesario para casos simples. Para agregar contexto, puede imprimir texto plano en stdout con código de salida o. Use JSON cuando necesite bloquear prompts o desee un control más estructurado.

PreToolUse

Se ejecuta después de que Claude crea parámetros de herramienta y antes de procesar la llamada a herramienta. Coincide en el nombre de la herramienta: `Bash` , `Edit` , `Write` , `Read` , `Glob` , `Grep` , `Agent` , `WebFetch` , `WebSearch` y cualquier [nombre de herramienta MCP](#).

Use [PreToolUse decision control](#) para permitir, denegar o pedir permiso para usar la herramienta.

Entrada de PreToolUse

Además de los [campos de entrada comunes](#), los hooks PreToolUse reciben `tool_name` , `tool_input` y `tool_use_id` . Los campos `tool_input` dependen de la herramienta:

Bash

Ejecuta comandos de shell.

Campo	Tipo	Ejemplo	Descripción
<code>command</code>	string	<code>"npm test"</code>	El comando de shell a ejecutar
<code>description</code>	string	<code>"Run test suite"</code>	Descripción opcional de lo que hace el comando
<code>timeout</code>	number	<code>120000</code>	Tiempo de espera opcional en milisegundos
<code>run_in_background</code>	boolean	<code>false</code>	Si se ejecuta el comando en segundo plano

Write

Crea o sobrescribe un archivo.

Campo	Tipo	Ejemplo	Descripción
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Ruta absoluta al archivo a escribir
<code>content</code>	string	<code>"file content"</code>	Contenido a escribir en el archivo

Edit

Reemplaza una cadena en un archivo existente.

Campo	Tipo	Ejemplo	Descripción
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Ruta absoluta al archivo a editar
<code>old_string</code>	string	<code>"original text"</code>	Texto a encontrar y reemplazar
<code>new_string</code>	string	<code>"replacement text"</code>	Texto de reemplazo
<code>replace_all</code>	boolean	<code>false</code>	Si se reemplazan todas las ocurrencias

Read

Lee contenidos de archivo.

Campo	Tipo	Ejemplo	Descripción
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Ruta absoluta al archivo a leer
<code>offset</code>	number	<code>10</code>	Número de línea opcional para comenzar a leer desde
<code>limit</code>	number	<code>50</code>	Número opcional de líneas a leer

Glob

Encuentra archivos que coincidan con un patrón glob.

Campo	Tipo	Ejemplo	Descripción
<code>pattern</code>	string	<code>"**/*.ts"</code>	Patrón glob para coincidir archivos contra

Campo	Tipo	Ejemplo	Descripción
<code>path</code>	string	<code>"/path/to/dir"</code>	Directorio opcional para buscar. Por defecto es el directorio de trabajo actual

Grep

Busca contenidos de archivo con expresiones regulares.

Campo	Tipo	Ejemplo	Descripción
<code>pattern</code>	string	<code>"TODO.*fix"</code>	Patrón de expresión regular a buscar
<code>path</code>	string	<code>"/path/to/dir"</code>	Archivo o directorio opcional para buscar
<code>glob</code>	string	<code>"*.ts"</code>	Patrón glob opcional para filtrar archivos
<code>output_mode</code>	string	<code>"content"</code>	<code>"content"</code> , <code>"files_with_matches"</code> o <code>"count"</code> . Por defecto es <code>"files_with_matches"</code>
<code>-i</code>	boolean	<code>true</code>	Búsqueda insensible a mayúsculas y minúsculas
<code>multiline</code>	boolean	<code>false</code>	Habilitar coincidencia multilínea

WebFetch

Obtiene y procesa contenido web.

Campo	Tipo	Ejemplo	Descripción
<code>url</code>	string	<code>"https://example.com/api"</code>	URL para obtener contenido de

Campo	Tipo	Ejemplo	Descripción
<code>prompt</code>	string	<code>"Extract the API endpoints"</code>	Prompt a ejecutar en el contenido obtenido

WebSearch

Busca en la web.

Campo	Tipo	Ejemplo	Descripción
<code>query</code>	string	<code>"react hooks best practices"</code>	Consulta de búsqueda
<code>allowed_domains</code>	array	<code>["docs.example.com"]</code>	Opcional: incluir solo resultados de estos dominios
<code>blocked_domains</code>	array	<code>["spam.example.com"]</code>	Opcional: excluir resultados de estos dominios

Agent

Genera un [subagente](#).

Campo	Tipo	Ejemplo	Descripción
<code>prompt</code>	string	<code>"Find all API endpoints"</code>	La tarea para que el agente realice
<code>description</code>	string	<code>"Find API endpoints"</code>	Descripción breve de la tarea
<code>subagent_type</code>	string	<code>"Explore"</code>	Tipo de agente especializado a usar
<code>model</code>	string	<code>"sonnet"</code>	Alias de modelo opcional para anular el predeterminado

Control de decisión de PreToolUse

Los hooks `PreToolUse` pueden controlar si una llamada a herramienta continúa. A diferencia de otros hooks que usan un campo `decision` de nivel superior, `PreToolUse` devuelve su decisión dentro de un objeto `hookSpecificOutput`. Esto le da control más rico: tres resultados (permitir, denegar o preguntar) más la capacidad de modificar la entrada de la herramienta antes de la ejecución.

Campo	Descripción
<code>permissionDecision</code>	"allow" omite el sistema de permisos, "deny" evita la llamada a herramienta, "ask" solicita al usuario que confirme
<code>permissionDecisionReason</code>	Para "allow" y "ask", se muestra al usuario pero no a Claude. Para "deny", se muestra a Claude
<code>updatedInput</code>	Modifica los parámetros de entrada de la herramienta antes de la ejecución. Combine con "allow" para aprobación automática, o "ask" para mostrar la entrada modificada al usuario
<code>additionalContext</code>	Cadena agregada al contexto de Claude antes de que se ejecute la herramienta

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

Note:

`PreToolUse` anteriormente usaba campos `decision` y `reason` de nivel superior, pero estos están deprecados para este evento. Use `hookSpecificOutput.permissionDecision` y `hookSpecificOutput.permissionDecisionReason` en su lugar. Los valores deprecados

"approve" y "block" se asignan a "allow" y "deny" respectivamente. Otros eventos como PostToolUse y Stop continúan usando `decision` y `reason` de nivel superior como su formato actual.

PermissionRequest

Se ejecuta cuando se muestra un diálogo de permiso al usuario. Use [PermissionRequest decision control](#) para permitir o denegar en nombre del usuario.

Coincide en el nombre de la herramienta, los mismos valores que PreToolUse.

Entrada de PermissionRequest

Los hooks PermissionRequest reciben campos `tool_name` y `tool_input` como los hooks PreToolUse, pero sin `tool_use_id`. Un array `permission_suggestions` opcional contiene las opciones “siempre permitir” que el usuario normalmente vería en el diálogo de permiso. La diferencia es cuándo se activa el hook: los hooks PermissionRequest se ejecutan cuando un diálogo de permiso está a punto de mostrarse al usuario, mientras que los hooks PreToolUse se ejecutan antes de la ejecución de la herramienta independientemente del estado de permiso.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PermissionRequest",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf node_modules",
    "description": "Remove node_modules directory"
  },
  "permission_suggestions": [
    { "type": "toolAlwaysAllow", "tool": "Bash" }
  ]
}
```

Control de decisión de PermissionRequest

Los hooks `PermissionRequest` pueden permitir o denegar solicitudes de permiso. Además de los [campos de salida JSON](#) disponibles para todos los hooks, su script de hook puede devolver un objeto `decision` con estos campos específicos del evento:

Campo	Descripción
<code>behavior</code>	<code>"allow"</code> otorga el permiso, <code>"deny"</code> lo deniega
<code>updatedInput</code>	Solo para <code>"allow"</code> : modifica los parámetros de entrada de la herramienta antes de la ejecución
<code>updatedPermissions</code>	Solo para <code>"allow"</code> : aplica actualizaciones de reglas de permiso, equivalente a que el usuario seleccione una opción “siempre permitir”
<code>message</code>	Solo para <code>"deny"</code> : le dice a Claude por qué se denegó el permiso
<code>interrupt</code>	Solo para <code>"deny"</code> : si es <code>true</code> , detiene a Claude

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

PostToolUse

Se ejecuta inmediatamente después de que una herramienta se completa exitosamente. Coincide en el nombre de la herramienta, los mismos valores que PreToolUse.

Entrada de PostToolUse

Los hooks `PostToolUse` se activan después de que una herramienta ya se ha ejecutado exitosamente. La entrada incluye tanto `tool_input` , los argumentos enviados a la herramienta, como `tool_response` , el resultado que devolvió. El esquema exacto para ambos depende de la herramienta.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "PostToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  },
  "tool_response": {
    "filePath": "/path/to/file.txt",
    "success": true
  },
  "tool_use_id": "toolu_01ABC123..."
}
```

Control de decisión de PostToolUse

Los hooks `PostToolUse` pueden proporcionar retroalimentación a Claude después de la ejecución de la herramienta. Además de los [campos de salida JSON](#) disponibles para todos los hooks, su script de hook puede devolver estos campos específicos del evento:

Campo	Descripción
<code>decision</code>	<code>"block"</code> solicita a Claude con la <code>reason</code> . Omita para permitir que la acción continúe
<code>reason</code>	Explicación mostrada a Claude cuando <code>decision</code> es <code>"block"</code>
<code>additionalContext</code>	Contexto adicional para que Claude considere
<code>updatedMCPToolOutput</code>	Solo para herramientas MCP : reemplaza la salida de la herramienta con el valor proporcionado

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

Se ejecuta cuando falla la ejecución de una herramienta. Este evento se activa para llamadas a herramientas que lanzan errores o devuelven resultados de fallo. Úselo para registrar fallos, enviar alertas o proporcionar retroalimentación correctiva a Claude.

Coincide en el nombre de la herramienta, los mismos valores que PreToolUse.

Entrada de PostToolUseFailure

Los hooks PostToolUseFailure reciben los mismos campos `tool_name` y `tool_input` que PostToolUse, junto con información de error como campos de nivel superior:

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUseFailure",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test",
    "description": "Run test suite"
  },
  "tool_use_id": "toolu_01ABC123...",
  "error": "Command exited with non-zero status code 1",
  "is_interrupt": false
}
```


Campo	Descripción
<code>error</code>	Cadena que describe qué salió mal
<code>is_interrupt</code>	Booleano opcional que indica si el fallo fue causado por interrupción del usuario

Control de decisión de `PostToolUseFailure`

Los hooks `PostToolUseFailure` pueden proporcionar contexto a Claude después de un fallo de herramienta. Además de los [campos de salida JSON](#) disponibles para todos los hooks, su script de hook puede devolver estos campos específicos del evento:

Campo	Descripción
<code>additionalContext</code>	Contexto adicional para que Claude considere junto con el error

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

Se ejecuta cuando Claude Code envía notificaciones. Coincide en el tipo de notificación: `permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`. Omita el matcher para ejecutar hooks para todos los tipos de notificación.

Use matchers separados para ejecutar diferentes controladores dependiendo del tipo de notificación. Esta configuración desencadena un script de alerta específico de permiso cuando Claude necesita aprobación de permiso y una notificación diferente cuando Claude ha estado inactivo:

```

{
  "hooks": {
    "Notification": [
      {
        "matcher": "permission_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/permission-alert.sh"
          }
        ]
      },
      {
        "matcher": "idle_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/idle-notification.sh"
          }
        ]
      }
    ]
  }
}

```

Entrada de Notification

Además de los [campos de entrada comunes](#), los hooks Notification reciben `message` con el texto de notificación, un `title` opcional y `notification_type` que indica qué tipo se activó.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "Notification",
  "message": "Claude needs your permission to use Bash",
  "title": "Permission needed",
  "notification_type": "permission_prompt"
}
```

Los hooks Notification no pueden bloquear o modificar notificaciones. Además de los [campos de salida JSON](#) disponibles para todos los hooks, puede devolver `additionalContext` para agregar contexto a la conversación:

Campo	Descripción
<code>additionalContext</code>	Cadena agregada al contexto de Claude

SubagentStart

Se ejecuta cuando se genera un subagente de Claude Code a través de la herramienta Agent. Admite matchers para filtrar por nombre de tipo de agente (agentes integrados como `Bash`, `Explore`, `Plan` o nombres de agentes personalizados de `.claude/agents/`).

Entrada de SubagentStart

Además de los [campos de entrada comunes](#), los hooks SubagentStart reciben `agent_id` con el identificador único para el subagente y `agent_type` con el nombre del agente (agentes integrados como `"Bash"`, `"Explore"`, `"Plan"` o nombres de agentes personalizados).

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "SubagentStart",
  "agent_id": "agent-abc123",
  "agent_type": "Explore"
}
```

Los hooks SubagentStart no pueden bloquear la creación de subagentes, pero pueden inyectar contexto en el subagente. Además de los [campos de salida JSON](#) disponibles para todos los hooks, puede devolver:

Campo	Descripción
<code>additionalContext</code>	Cadena agregada al contexto del subagente

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

Se ejecuta cuando un subagente de Claude Code ha terminado de responder. Coincide en el tipo de agente, los mismos valores que SubagentStart.

Entrada de SubagentStop

Además de los [campos de entrada comunes](#), los hooks SubagentStop reciben `stop_hook_active`, `agent_id`, `agent_type`, `agent_transcript_path` y `last_assistant_message`. El campo `agent_type` es el valor utilizado para filtrado de matcher. El `transcript_path` es la transcripción de la sesión principal, mientras que `agent_transcript_path` es la propia transcripción del subagente almacenada en una car-

peta `subagents/` anidada. El campo `last_assistant_message` contiene el contenido de texto de la respuesta final del subagente, por lo que los hooks pueden acceder a él sin analizar el archivo de transcripción.

```
{
  "session_id": "abc123",
  "transcript_path": "~/claude/projects/.../abc123.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStop",
  "stop_hook_active": false,
  "agent_id": "def456",
  "agent_type": "Explore",
  "agent_transcript_path": "~/claude/projects/.../abc123/subagents/agent-
def456.jsonl",
  "last_assistant_message": "Analysis complete. Found 3 potential issues..."
}
```

Los hooks `SubagentStop` usan el mismo formato de control de decisión que los [hooks Stop](#).

Stop

Se ejecuta cuando el agente principal de Claude Code ha terminado de responder. No se ejecuta si la detención ocurrió debido a una interrupción del usuario.

Entrada de Stop

Además de los [campos de entrada comunes](#), los hooks `Stop` reciben `stop_hook_active` y `last_assistant_message`. El campo `stop_hook_active` es `true` cuando Claude Code ya está continuando como resultado de un hook de parada. Verifique este valor o procese la transcripción para evitar que Claude Code se ejecute indefinidamente. El campo `last_assistant_message` contiene el contenido de texto de la respuesta final de Claude, por lo que los hooks pueden acceder a él sin analizar el archivo de transcripción.

```
{
  "session_id": "abc123",
  "transcript_path": "~/claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "Stop",
  "stop_hook_active": true,
  "last_assistant_message": "I've completed the refactoring. Here's a summary..."
}
```

Control de decisión de Stop

Los hooks `Stop` y `SubagentStop` pueden controlar si Claude continúa. Además de los [campos de salida JSON](#) disponibles para todos los hooks, su script de hook puede devolver estos campos específicos del evento:

Campo	Descripción
<code>decision</code>	<code>"block"</code> evita que Claude se detenga. Omita para permitir que Claude se detenga
<code>reason</code>	Requerido cuando <code>decision</code> es <code>"block"</code> . Le dice a Claude por qué debe continuar

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

Se ejecuta cuando un compañero de [equipo de agentes](#) está a punto de quedarse inactivo después de terminar su turno. Úselo para aplicar puertas de calidad antes de que un compañero deje de trabajar, como requerir que pasen verificaciones de lint o verificar que existan archivos de salida.

Cuando un hook `TeammateIdle` sale con código 2, el compañero recibe el mensaje de stderr como retroalimentación y continúa trabajando en lugar de quedarse inactivo. Para detener al compañero completamente en lugar de re-ejecutarlo, devuelva JSON con `{"continue": false, "stopReason": "..."}.` Los hooks `TeammateIdle` no admiten matchers y se activan en cada ocurrencia.

Entrada de TeammateIdle

Además de los [campos de entrada comunes](#), los hooks `TeammateIdle` reciben `teammate_name` y `team_name`.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TeammateIdle",
  "teammate_name": "researcher",
  "team_name": "my-project"
}
```

Campo	Descripción
<code>teammate_name</code>	Nombre del compañero que está a punto de quedarse inactivo
<code>team_name</code>	Nombre del equipo

Control de decisión de TeammateIdle

Los hooks `TeammateIdle` admiten dos formas de controlar el comportamiento del compañero:

- **Código de salida 2:** el compañero recibe el mensaje de stderr como retroalimentación y continúa trabajando en lugar de quedarse inactivo.
- **JSON** `{"continue": false, "stopReason": "..."}:` detiene al compañero completamente, coincidiendo con el comportamiento del hook `Stop`. El `stopReason` se muestra al usuario.

Este ejemplo verifica que exista un artefacto de compilación antes de permitir que un compañero se quede inactivo:

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
    echo "Build artifact missing. Run the build before stopping." >&2
    exit 2
fi

exit 0
```

TaskCompleted

Se ejecuta cuando una tarea está siendo marcada como completada. Esto se activa en dos situaciones: cuando cualquier agente marca explícitamente una tarea como completada a través de la herramienta TaskUpdate, o cuando un compañero de [equipo de agentes](#) termina su turno con tareas en progreso. Úselo para aplicar criterios de finalización como pasar pruebas o verificaciones de lint antes de que una tarea pueda cerrarse.

Cuando un hook `TaskCompleted` sale con código 2, la tarea no se marca como completada y el mensaje de stderr se devuelve al modelo como retroalimentación. Para detener al compañero completamente en lugar de re-ejecutarlo, devuelva JSON con `{"continue": false, "stopReason": "..."} . Los hooks TaskCompleted no admiten matchers y se activan en cada ocurrencia.`

Entrada de TaskCompleted

Además de los [campos de entrada comunes](#), los hooks TaskCompleted reciben `task_id`, `task_subject` y opcionalmente `task_description`, `teammate_name` y `team_name`.


```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "TaskCompleted",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "my-project"
}
```

Campo	Descripción
<code>task_id</code>	Identificador de la tarea que se está completando
<code>task_subject</code>	Título de la tarea
<code>task_description</code>	Descripción detallada de la tarea. Puede estar ausente
<code>teammate_name</code>	Nombre del compañero que completa la tarea. Puede estar ausente
<code>team_name</code>	Nombre del equipo. Puede estar ausente

Control de decisión de TaskCompleted

Los hooks TaskCompleted admiten dos formas de controlar la finalización de tareas:

- **Código de salida 2:** la tarea no se marca como completada y el mensaje de stderr se devuelve al modelo como retroalimentación.
- **JSON** `{"continue": false, "stopReason": "..."}:` detiene al compañero completamente, coincidiendo con el comportamiento del hook `Stop`. El `stopReason` se muestra al usuario.

Este ejemplo ejecuta pruebas y bloquea la finalización de tareas si fallan:

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## Run the test suite
if ! npm test 2>&1; then
    echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
    exit 2
fi

exit 0
```

ConfigChange

Se ejecuta cuando un archivo de configuración cambia durante una sesión. Úselo para auditar cambios de configuración, aplicar políticas de seguridad o bloquear modificaciones no autorizadas a archivos de configuración.

Los hooks ConfigChange se activan para cambios en archivos de configuración, configuración de política administrada y archivos de skill. El campo `source` en la entrada le dice qué tipo de configuración cambió, y el campo `file_path` opcional proporciona la ruta al archivo cambiado.

El matcher filtra en la fuente de configuración:

Matcher	Cuándo se activa
<code>user_settings</code>	<code>~/.claude/settings.json</code> cambia
<code>project_settings</code>	<code>.claude/settings.json</code> cambia
<code>local_settings</code>	<code>.claude/settings.local.json</code> cambia
<code>policy_settings</code>	La configuración de política administrada cambia
<code>skills</code>	Un archivo de skill en <code>.claude/skills/</code> cambia

Este ejemplo registra todos los cambios de configuración para auditoría de seguridad:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

Entrada de ConfigChange

Además de los [campos de entrada comunes](#), los hooks ConfigChange reciben `source` y opcionalmente `file_path`. El campo `source` indica qué tipo de configuración cambió, y `file_path` proporciona la ruta al archivo específico que se modificó.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "ConfigChange",
  "source": "project_settings",
  "file_path": "/Users/.../my-project/.claude/settings.json"
}
```

Control de decisión de ConfigChange

Los hooks ConfigChange pueden bloquear cambios de configuración para que no tengan efecto. Use código de salida 2 o un JSON `decision` para evitar el cambio. Cuando se bloquea, la nueva configuración no se aplica a la sesión en ejecución.

Campo	Descripción
<code>decision</code>	<code>"block"</code> evita que el cambio de configuración se aplique. Omita para permitir el cambio
<code>reason</code>	Explicación mostrada al usuario cuando <code>decision</code> es <code>"block"</code>

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

Los cambios de `policy_settings` no pueden bloquearse. Los hooks aún se activan para fuentes de `policy_settings` , por lo que puede usarlos para registro de auditoría, pero cualquier decisión de bloqueo se ignora. Esto asegura que la configuración administrada por la empresa siempre tenga efecto.

WorktreeCreate

Cuando ejecuta `claude --worktree` o un [subagente usa `isolation: "worktree"`](#) , Claude Code crea una copia de trabajo aislada usando `git worktree` . Si configura un hook `WorktreeCreate`, reemplaza el comportamiento predeterminado de git, permitiéndole usar un sistema de control de versiones diferente como SVN, Perforce o Mercurial.

El hook debe imprimir la ruta absoluta al directorio de worktree creado en stdout. Claude Code usa esta ruta como el directorio de trabajo para la sesión aislada.

Este ejemplo crea una copia de trabajo SVN e imprime la ruta para que Claude Code la use. Reemplace la URL del repositorio con la suya:

```
{
  "hooks": {
    "WorktreeCreate": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'NAME=$(jq -r .name); DIR=\"$HOME/.claude/
worktrees/$NAME\"; svn checkout https://svn.example.com/repo/trunk
\\$DIR\" >&2 && echo \\$DIR\"'"
          }
        ]
      }
    ]
  }
}
```

El hook lee el **name** del worktree de la entrada JSON en stdin, verifica una copia fresca en un nuevo directorio e imprime la ruta del directorio. El **echo** en la última línea es lo que Claude Code lee como la ruta del worktree. Redirija cualquier otra salida a stderr para que no interfiera con la ruta.

Entrada de WorktreeCreate

Además de los [campos de entrada comunes](#), los hooks WorktreeCreate reciben el campo **name**. Este es un identificador slug para el nuevo worktree, especificado por el usuario o generado automáticamente (por ejemplo, **bold-oak-a3f2**).

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeCreate",
  "name": "feature-auth"
}
```

Salida de WorktreeCreate

El hook debe imprimir la ruta absoluta al directorio de worktree creado en stdout. Si el hook falla o no produce salida, la creación de worktree falla con un error.

Los hooks WorktreeCreate no usan el modelo de decisión de permitir/bloquear estándar. En su lugar, el éxito o fallo del hook determina el resultado. Solo se admiten hooks `type: "command"`.

WorktreeRemove

La contraparte de limpieza de [WorktreeCreate](#). Este hook se activa cuando se está eliminando un worktree, ya sea cuando sale de una sesión `--worktree` y elige eliminarlo, o cuando un subagente con `isolation: "worktree"` finaliza. Para worktrees basados en git, Claude maneja la limpieza automáticamente con `git worktree remove`. Si configuró un hook WorktreeCreate para un sistema de control de versiones que no es git, emparéjelo con un hook WorktreeRemove para manejar la limpieza. Sin uno, el directorio de worktree se deja en el disco.

Claude Code pasa la ruta que WorktreeCreate imprimió en stdout como `worktree_path` en la entrada del hook. Este ejemplo lee esa ruta y elimina el directorio:

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

Entrada de WorktreeRemove

Además de los [campos de entrada comunes](#), los hooks WorktreeRemove reciben el campo `worktree_path`, que es la ruta absoluta al worktree que se está eliminando.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "hook_event_name": "WorktreeRemove",
  "worktree_path": "/Users/.../my-project/.claude/worktrees/feature-auth"
}
```

Los hooks WorktreeRemove no tienen control de decisión. No pueden bloquear la eliminación de worktree pero pueden realizar tareas de limpieza como eliminar estado de control de versiones o archivar cambios. Los fallos de hook se registran solo en modo de depuración. Solo se admiten hooks `type: "command"`.

PreCompact

Se ejecuta antes de que Claude Code esté a punto de ejecutar una operación de compactación.

El valor del matcher indica si la compactación fue desencadenada manualmente o automáticamente:

Ma- tcher	Cuándo se activa
manual	/compact
auto	Compactación automática cuando la ventana de contexto está llena

Entrada de PreCompact

Además de los [campos de entrada comunes](#), los hooks PreCompact reciben `trigger` y `custom_instructions`. Para `manual`, `custom_instructions` contiene lo que el usuario pasa a `/compact`. Para `auto`, `custom_instructions` está vacío.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "PreCompact",
  "trigger": "manual",
  "custom_instructions": ""
}
```

SessionEnd

Se ejecuta cuando termina una sesión de Claude Code. Útil para tareas de limpieza, registro de estadísticas de sesión o guardado del estado de sesión. Admite matchers para filtrar por razón de salida.

El campo `reason` en la entrada del hook indica por qué terminó la sesión:

Razón	Descripción
<code>clear</code>	Sesión borrada con comando <code>/clear</code>
<code>logout</code>	El usuario cerró sesión
<code>prompt_input_exit</code>	El usuario salió mientras la entrada del prompt era visible
<code>bypass_permissions_disabled</code>	El modo de permisos de omisión fue deshabilitado
<code>other</code>	Otras razones de salida

Entrada de SessionEnd

Además de los [campos de entrada comunes](#), los hooks SessionEnd reciben un campo `reason` que indica por qué terminó la sesión. Consulte la [tabla de razones](#) anterior para todos los valores.


```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SessionEnd",
  "reason": "other"
}
```

Los hooks `SessionEnd` no tienen control de decisión. No pueden bloquear la terminación de sesión pero pueden realizar tareas de limpieza.

Hooks basados en prompt

Además de hooks de comando y HTTP, Claude Code admite hooks basados en prompt (`type: "prompt"`) que usan un LLM para evaluar si permitir o bloquear una acción, y hooks de agente (`type: "agent"`) que generan un verificador agentico con acceso a herramientas. No todos los eventos admiten todos los tipos de hooks.

Eventos que admiten los cuatro tipos de hooks (`command` , `http` , `prompt` y `agent`):

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

Eventos que solo admiten hooks `type: "command"` :

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`

- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

Cómo funcionan los hooks basados en prompt

En lugar de ejecutar un comando Bash, los hooks basados en prompt:

1. Envían la entrada del hook y su prompt a un modelo Claude, Haiku por defecto
2. El LLM responde con JSON estructurado que contiene una decisión
3. Claude Code procesa la decisión automáticamente

Configuración de hook de prompt

Establezca `type` en `"prompt"` y proporcione una cadena `prompt` en lugar de un `command`. Use el marcador de posición `$ARGUMENTS` para inyectar datos de entrada JSON del hook en su texto de prompt. Claude Code envía el prompt combinado e entrada a un modelo Claude rápido, que devuelve una decisión JSON.

Este hook `Stop` le pide al LLM que evalúe si Claude debe detenerse antes de permitir que finalice:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```

Campo	Requerido	Descripción
type	sí	Debe ser "prompt"
prompt	sí	El texto del prompt a enviar al LLM. Use \$ARGUMENTS como marcador de posición para la entrada JSON del hook. Si \$ARGUMENTS no está presente, la entrada JSON se agrega al prompt
model	no	Modelo a usar para evaluación. Por defecto es un modelo rápido
timeout	no	Tiempo de espera en segundos. Predeterminado: 30

Esquema de respuesta

El LLM debe responder con JSON que contenga:

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

Campo	Descripción
ok	true permite la acción, false la previene
reason	Requerido cuando ok es false . Explicación mostrada a Claude

Ejemplo: Hook Stop de múltiples criterios

Este hook Stop usa un prompt detallado para verificar tres condiciones antes de permitir que Claude se detenga. Si "ok" es false , Claude continúa trabajando con la razón proporcionada como su siguiente instrucción. Los hooks SubagentStop usan el mismo formato para evaluar si un subagente debe detenerse:

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "You are evaluating whether Claude should stop working.
Context: $ARGUMENTS\n\nAnalyze the conversation and determine if:\n1. All user-
requested tasks are complete\n2. Any errors need to be addressed\n3. Follow-up
work is needed\n\nRespond with JSON: {\"ok\": true} to allow stopping, or
{\"ok\": false, \"reason\": \"your explanation\"} to continue working.",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

Hooks basados en agente

Los hooks basados en agente (`type: "agent"`) son como hooks basados en prompt pero con acceso a herramientas de múltiples turnos. En lugar de una única llamada LLM, un hook de agente genera un subagente que puede leer archivos, buscar código e inspeccionar la base de código para verificar condiciones. Los hooks de agente admiten los mismos eventos que los hooks basados en prompt.

Cómo funcionan los hooks de agente

Cuando se activa un hook de agente:

1. Claude Code genera un subagente con su prompt y la entrada JSON del hook
2. El subagente puede usar herramientas como Read, Grep y Glob para investigar
3. Después de hasta 50 turnos, el subagente devuelve una decisión estructurada `{ "ok": true/false }`
4. Claude Code procesa la decisión de la misma manera que un hook de prompt

Los hooks de agente son útiles cuando la verificación requiere inspeccionar archivos reales o salida de prueba, no solo evaluar los datos de entrada del hook solos.

Configuración de hook de agente

Establezca `type` en `"agent"` y proporcione una cadena `prompt`. Los campos de configuración son los mismos que los [hooks de prompt](#), con un tiempo de espera predeterminado más largo:

Campo	Requerido	Descripción
<code>type</code>	sí	Debe ser <code>"agent"</code>
<code>prompt</code>	sí	Prompt que describe qué verificar. Use <code>\$ARGUMENTS</code> como marcador de posición para la entrada JSON del hook
<code>model</code>	no	Modelo a usar. Por defecto es un modelo rápido
<code>timeout</code>	no	Tiempo de espera en segundos. Predeterminado: 60

El esquema de respuesta es el mismo que los hooks de prompt: `{ "ok": true }` para permitir o `{ "ok": false, "reason": "..." }` para bloquear.

Este hook `Stop` verifica que todas las pruebas unitarias pasen antes de permitir que Claude finalice:

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Ejecutar hooks en segundo plano

Por defecto, los hooks bloquean la ejecución de Claude hasta que se completen. Para tareas de larga duración como implementaciones, suites de prueba o llamadas a API externas, establezca `"async": true` para ejecutar el hook en segundo plano mientras Claude continúa trabajando. Los hooks asincronos no pueden bloquear o controlar el comportamiento de Claude: campos de respuesta como `decision`, `permissionDecision` y `continue` no tienen efecto, porque la acción que habrían controlado ya se ha completado.

Configurar un hook asincrónico

Agregue `"async": true` a la configuración de un hook de comando para ejecutarlo en segundo plano sin bloquear a Claude. Este campo solo está disponible en hooks `type: "command"`.

Este hook ejecuta un script de prueba después de cada llamada a herramienta `Write`. Claude continúa trabajando inmediatamente mientras `run-tests.sh` se ejecuta durante hasta 120 segundos. Cuando el script finaliza, su salida se entrega en el siguiente turno de conversación:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

El campo `timeout` establece el tiempo máximo en segundos para el proceso de fondo. Si no se especifica, los hooks asincronos usan el mismo predeterminado de 10 minutos que los hooks sincronos.

Cómo se ejecutan los hooks asincronos

Cuando se activa un hook asincrónico, Claude Code inicia el proceso del hook e inmediatamente continúa sin esperar a que finalice. El hook recibe la misma entrada JSON a través de stdin que un hook sincrónico.

Después de que el proceso de fondo sale, si el hook produjo una respuesta JSON con un campo `systemMessage` o `additionalContext`, ese contenido se entrega a Claude como contexto en el siguiente turno de conversación.

Ejemplo: ejecutar pruebas después de cambios de archivo

Este hook inicia una suite de prueba en segundo plano cada vez que Claude escribe un archivo, luego reporta los resultados a Claude cuando las pruebas finalizan. Guarde este script en `.claude/hooks/run-tests-async.sh` en su proyecto y hágalo ejecutable con `chmod +x`:

```
#!/bin/bash
## run-tests-async.sh

## Read hook input from stdin
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## Only run tests for source files
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
    exit 0
fi

## Run tests and report results via systemMessage
RESULT=$(npm test 2>&1)
EXIT_CODE=$?

if [ $EXIT_CODE -eq 0 ]; then
    echo "{\"systemMessage\": \"Tests passed after editing $FILE_PATH\"}"
else
    echo "{\"systemMessage\": \"Tests failed after editing $FILE_PATH: $RESULT\"}"
fi
```

Luego agregue esta configuración a `.claude/settings.json` en la raíz de su proyecto. La bandera `async: true` permite que Claude continúe trabajando mientras se ejecutan las pruebas:


```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}

```

Limitaciones

Los hooks asincronos tienen varias restricciones en comparación con los hooks sincronos:

- Solo los hooks `type: "command"` admiten `async`. Los hooks basados en prompt no pueden ejecutarse de forma asincrónica.
- Los hooks asincronos no pueden bloquear llamadas a herramientas o devolver decisiones. Para cuando se completa el hook, la acción desencadenante ya ha procedido.
- La salida del hook se entrega en el siguiente turno de conversación. Si la sesión está inactiva, la respuesta espera hasta la siguiente interacción del usuario.
- Cada ejecución crea un proceso de fondo separado. No hay deduplicación en múltiples activaciones del mismo hook asincrónico.

Consideraciones de seguridad

Descargo de responsabilidad

Los hooks de comando se ejecutan con los permisos completos de su usuario del sistema.

Warning:

Los hooks de comando ejecutan comandos de shell con sus permisos de usuario completos. Pueden modificar, eliminar o acceder a cualquier archivo al que su cuenta de usuario pueda acceder. Revise y pruebe todos los comandos de hook antes de agregarlos a su configuración.

Mejores prácticas de seguridad

Tenga en cuenta estas prácticas al escribir hooks:

- **Validar y desinfectar entradas:** nunca confíe en datos de entrada ciegamente
- **Siempre entrecomillar variables de shell:** use `"$VAR"` no `$VAR`
- **Bloquear traversal de ruta:** verifique `..` en rutas de archivo
- **Usar rutas absolutas:** especifique rutas completas para scripts, usando `"$CLAUDE_PROJECT_DIR"` para la raíz del proyecto
- **Omitir archivos sensibles:** evite `.env`, `.git/`, claves, etc.

Depurar hooks

Ejecute `claude --debug` para ver detalles de ejecución de hooks, incluyendo qué hooks coincidieron, sus códigos de salida y salida. Alterne el modo detallado con `Ctrl+O` para ver el progreso del hook en la transcripción.

```
[DEBUG] Executing hooks for PostToolUse:Write
[DEBUG] Getting matching hook commands for PostToolUse with query: Write
[DEBUG] Found 1 hook matchers in settings
[DEBUG] Matched 1 hooks for query "Write"
[DEBUG] Found 1 hook commands to execute
[DEBUG] Executing hook command: <Your command> with timeout 600000ms
[DEBUG] Hook command completed with status 0: <Your stdout>
```

Para solucionar problemas comunes como hooks que no se activan, bucles infinitos de Stop hook o errores de configuración, consulte [Limitations and troubleshooting](#) en la guía.

Automatizar flujos de trabajo con hooks

Ejecuta comandos de shell automáticamente cuando Claude Code edita archivos, finaliza tareas o necesita entrada. Formatea código, envía notificaciones, valida comandos y aplica reglas del proyecto.

Los hooks son comandos de shell definidos por el usuario que se ejecutan en puntos específicos del ciclo de vida de Claude Code. Proporcionan control determinista sobre el comportamiento de Claude Code, asegurando que ciertas acciones siempre ocurran en lugar de depender de que el LLM elija ejecutarlas. Usa hooks para aplicar reglas del proyecto, automatizar tareas repetitivas e integrar Claude Code con tus herramientas existentes.

Para decisiones que requieren criterio en lugar de reglas deterministas, también puedes usar [hooks basados en prompts](#) o [hooks basados en agentes](#) que utilizan un modelo Claude para evaluar condiciones.

Para otras formas de extender Claude Code, consulta [skills](#) para dar a Claude instrucciones adicionales y comandos ejecutables, [subagents](#) para ejecutar tareas en contextos aislados, y [plugins](#) para empaquetar extensiones para compartir entre proyectos.

Tip:

Esta guía cubre casos de uso comunes y cómo comenzar. Para esquemas de eventos completos, formatos de entrada/salida JSON y características avanzadas como hooks asíncronos y hooks de herramientas MCP, consulta la [referencia de Hooks](#).

Configura tu primer hook

La forma más rápida de crear un hook es a través del menú interactivo `/hooks` en Claude Code. Este tutorial crea un hook de notificación de escritorio, para que recibas una alerta cada vez que Claude esté esperando tu entrada en lugar de ver la terminal.

Step 1: Abre el menú de hooks

Escribe `/hooks` en la CLI de Claude Code. Verás una lista de todos los eventos de hook disponibles, más una opción para desactivar todos los hooks. Cada evento corresponde a un punto en el ciclo de vida de Claude donde puedes ejecutar código personalizado. Selecciona `Notification` para crear un hook que se active cuando Claude necesite tu atención.

Step 2: Configura el matcher

El menú muestra una lista de matchers, que filtran cuándo se activa el hook. Establece el matcher en `*` para activarse en todos los tipos de notificación. Puedes estrecharlo más tarde cambiando el matcher a un valor específico como `permission_prompt` o `idle_prompt`.

Step 3: Añade tu comando

Selecciona `+ Add new hook...`. El menú te solicita un comando de shell para ejecutar cuando se active el evento. Los hooks ejecutan cualquier comando de shell que proporciones, por lo que puedes usar la herramienta de notificación integrada de tu plataforma. Copia el comando para tu SO:

macOS

Usa `osascript` para activar una notificación nativa de macOS a través de AppleScript:

```
osascript -e 'display notification "Claude Code needs your attention" with title "Claude Code"'
```

Linux

Usa `notify-send`, que viene preinstalado en la mayoría de escritorios Linux con un demonio de notificación:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

Usa PowerShell para mostrar un cuadro de mensaje nativo a través de Windows Forms de .NET:

```
powershell.exe -Command "[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')"
```

Step 4: Elige una ubicación de almacenamiento

El menú te pregunta dónde guardar la configuración del hook. Selecciona `User settings` para almacenarlo en `~/.claude/settings.json`, que aplica el hook a todos tus proyectos. También podrías elegir `Project settings` para limitarlo al proyecto actual. Consulta [Configura la ubicación del hook](#) para todos los ámbitos disponibles.

Step 5: Prueba el hook

Presiona **Esc** para volver a la CLI. Pídele a Claude que haga algo que requiera permiso, luego cambia de la terminal. Deberías recibir una notificación de escritorio.

Qué puedes automatizar

Los hooks te permiten ejecutar código en puntos clave del ciclo de vida de Claude Code: formatear archivos después de ediciones, bloquear comandos antes de que se ejecuten, enviar notificaciones cuando Claude necesita entrada, inyectar contexto al inicio de la sesión, y más. Para la lista completa de eventos de hook, consulta la [referencia de Hooks](#).

Cada ejemplo incluye un bloque de configuración listo para usar que añades a un [archivo de configuración](#). Los patrones más comunes:

- [Recibe notificaciones cuando Claude necesita entrada](#)
- [Formatea automáticamente el código después de ediciones](#)
- [Bloquea ediciones a archivos protegidos](#)
- [Reinyecta contexto después de compactación](#)
- [Audita cambios de configuración](#)

Recibe notificaciones cuando Claude necesita entrada

Obtén una notificación de escritorio cada vez que Claude termine de trabajar y necesite tu entrada, para que puedas cambiar a otras tareas sin verificar la terminal.

Este hook usa el evento **Notification**, que se activa cuando Claude está esperando entrada o permiso. Cada pestaña a continuación usa el comando de notificación nativo de la plataforma. Añade esto a `~/.claude/settings.json`, o usa el [tutorial interactivo](#) anterior para configurarlo con `/hooks`:

macOS

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude Code needs your attention\" with title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

Linux

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "notify-send 'Claude Code' 'Claude Code needs your attention'"
          }
        ]
      }
    ]
  }
}
```

Windows (PowerShell)

```

{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe -Command \"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')\"";
          }
        ]
      }
    ]
  }
}

```

Formatea automáticamente el código después de ediciones

Ejecuta automáticamente [Prettier](#) en cada archivo que Claude edita, para que el formato se mantenga consistente sin intervención manual.

Este hook usa el evento `PostToolUse` con un matcher `Edit|Write`, por lo que se ejecuta solo después de herramientas de edición de archivos. El comando extrae la ruta del archivo editado con `jq` y la pasa a Prettier. Añade esto a `.claude/settings.json` en la raíz de tu proyecto:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}

```

Note:

Los ejemplos de Bash en esta página usan `jq` para análisis JSON. Instálalo con `brew install jq` (macOS), `apt-get install jq` (Debian/Ubuntu), o consulta [descargas de jq](#).

Bloquea ediciones a archivos protegidos

Evita que Claude modifique archivos sensibles como `.env`, `package-lock.json`, o cualquier cosa en `.git/`. Claude recibe retroalimentación explicando por qué se bloqueó la edición, para que pueda ajustar su enfoque.

Este ejemplo usa un archivo de script separado que el hook llama. El script verifica la ruta del archivo de destino contra una lista de patrones protegidos y sale con código 2 para bloquear la edición.

Step 1: Crea el script del hook

Guarda esto en `.claude/hooks/protect-files.sh`:


```
#!/bin/bash
## protect-files.sh

INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

PROTECTED_PATTERNS=( ".env" "package-lock.json" ".git/" )

for pattern in "${PROTECTED_PATTERNS[@]}; do
    if [[ "$FILE_PATH" == *"$pattern"* ]]; then
        echo "Blocked: $FILE_PATH matches protected pattern '$pattern'" >&2
        exit 2
    fi
done

exit 0
```

Step 2: Haz el script ejecutable (macOS/Linux)

Los scripts de hook deben ser ejecutables para que Claude Code los ejecute:

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: Registra el hook

Añade un hook `PreToolUse` a `.claude/settings.json` que ejecute el script antes de cualquier llamada a herramienta `Edit` o `Write`:

```

{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}

```

Reinyecta contexto después de compactación

Cuando la ventana de contexto de Claude se llena, la compactación resume la conversación para liberar espacio. Esto puede perder detalles importantes. Usa un hook `SessionStart` con un matcher `compact` para reinyectar contexto crítico después de cada compactación.

Cualquier texto que tu comando escriba en stdout se añade al contexto de Claude. Este ejemplo recuerda a Claude las convenciones del proyecto y el trabajo reciente. Añade esto a `.claude/settings.json` en la raíz de tu proyecto:

```

{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}

```

Puedes reemplazar el `echo` con cualquier comando que produzca salida dinámica, como `git log --oneline -5` para mostrar commits recientes. Para inyectar contexto en cada inicio de sesión, considera usar [CLAUDE.md](#) en su lugar. Para variables de entorno, consulta [CLAUDE_ENV_FILE](#) en la referencia.

Audita cambios de configuración

Realiza un seguimiento de cuándo los archivos de configuración o skills cambian durante una sesión. El evento `ConfigChange` se activa cuando un proceso externo o editor modifica un archivo de configuración, para que puedas registrar cambios para cumplimiento o bloquear modificaciones no autorizadas.

Este ejemplo añade cada cambio a un registro de auditoría. Añade esto a `~/.claude/settings.json`:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

El matcher filtra por tipo de configuración: `user_settings`, `project_settings`, `local_settings`, `policy_settings`, o `skills`. Para bloquear que un cambio tenga efecto, sal con código 2 o devuelve `{"decision": "block"}`. Consulta la [referencia de Config-Change](#) para el esquema de entrada completo.

Cómo funcionan los hooks

Los eventos de hook se activan en puntos específicos del ciclo de vida de Claude Code. Cuando se activa un evento, todos los hooks coincidentes se ejecutan en paralelo, y los comandos de hook idénticos se deduplicarán automáticamente. La tabla a continuación muestra cada evento y cuándo se activa:

Event	When it fires
<code>SessionStart</code>	When a session begins or resumes
<code>UserPromptSubmit</code>	When you submit a prompt, before Claude processes it
<code>PreToolUse</code>	Before a tool call executes. Can block it
<code>PermissionRequest</code>	When a permission dialog appears
<code>PostToolUse</code>	After a tool call succeeds

Event	When it fires
<code>PostToolUseFailure</code>	After a tool call fails
<code>Notification</code>	When Claude Code sends a notification
<code>SubagentStart</code>	When a subagent is spawned
<code>SubagentStop</code>	When a subagent finishes
<code>Stop</code>	When Claude finishes responding
<code>TeammateIdle</code>	When an agent team teammate is about to go idle
<code>TaskCompleted</code>	When a task is being marked as completed
<code>InstructionsLoaded</code>	When a <code>CLAUDE.md</code> or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
<code>ConfigChange</code>	When a configuration file changes during a session
<code>WorktreeCreate</code>	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
<code>WorktreeRemove</code>	When a worktree is being removed, either at session exit or when a subagent finishes
<code>PreCompact</code>	Before context compaction
<code>PostCompact</code>	After context compaction completes
<code>Elicitation</code>	When an MCP server requests user input during a tool call
<code>ElicitationResult</code>	After a user responds to an MCP elicitation, before the response is sent back to the server
<code>SessionEnd</code>	When a session terminates

Cada hook tiene un `type` que determina cómo se ejecuta. La mayoría de los hooks usan `"type": "command"`, que ejecuta un comando de shell. Hay otros tres tipos disponibles:

- `"type": "http"`: POST de datos de evento a una URL. Consulta [HTTP hooks](#).
- `"type": "prompt"`: evaluación LLM de un solo turno. Consulta [Hooks basados en prompts](#).

- **"type": "agent"** : verificación multi-turno con acceso a herramientas. Consulta [Hooks basados en agentes](#).

Lee entrada y devuelve salida

Los hooks se comunican con Claude Code a través de stdin, stdout, stderr y códigos de salida. Cuando se activa un evento, Claude Code pasa datos específicos del evento como JSON a stdin de tu script. Tu script lee esos datos, hace su trabajo, y le dice a Claude Code qué hacer a continuación a través del código de salida.

Entrada del hook

Cada evento incluye campos comunes como `session_id` y `cwd`, pero cada tipo de evento añade datos diferentes. Por ejemplo, cuando Claude ejecuta un comando Bash, un hook `PreToolUse` recibe algo como esto en stdin:

```
{
  "session_id": "abc123",           // ID único para esta sesión
  "cwd": "/Users/sarah/myproject", // directorio de trabajo cuando se activó el ev
ento
  "hook_event_name": "PreToolUse", // qué evento activó este hook
  "tool_name": "Bash",             // la herramienta que Claude está a punto de us
ar
  "tool_input": {                  // los argumentos que Claude pasó a la herramie
nta
    "command": "npm test"         // para Bash, este es el comando de shell
  }
}
```

Tu script puede analizar ese JSON y actuar sobre cualquiera de esos campos. Los hooks `UserPromptSubmit` obtienen el texto `prompt` en su lugar, los hooks `SessionStart` obtienen la `source` (startup, resume, clear, compact), y así sucesivamente. Consulta [Campos de entrada comunes](#) en la referencia para campos compartidos, y la sección de cada evento para esquemas específicos del evento.

Salida del hook

Tu script le dice a Claude Code qué hacer a continuación escribiendo en stdout o stderr y saliendo con un código específico. Por ejemplo, un hook `PreToolUse` que quiere bloquear un comando:

```
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q "drop table"; then
    echo "Blocked: dropping tables is not allowed" >&2 # stderr se convierte en
    retroalimentación de Claude
    exit 2 # exit 2 = bloquea la acción
fi

exit 0 # exit 0 = permite que proceda
```

El código de salida determina qué sucede a continuación:

- **Exit 0:** la acción procede. Para hooks `UserPromptSubmit` y `SessionStart`, cualquier cosa que escribas en stdout se añade al contexto de Claude.
- **Exit 2:** la acción se bloquea. Escribe una razón en stderr, y Claude la recibe como retroalimentación para que pueda ajustar.
- **Cualquier otro código de salida:** la acción procede. Stderr se registra pero no se muestra a Claude. Alterna el modo detallado con `Ctrl+O` para ver estos mensajes en la transcripción.

Salida JSON estructurada

Los códigos de salida te dan dos opciones: permitir o bloquear. Para más control, sal con 0 e imprime un objeto JSON a stdout en su lugar.

Note:

Usa exit 2 para bloquear con un mensaje stderr, o exit 0 con JSON para control estructurado. No los mezcles: Claude Code ignora JSON cuando sales con 2.

Por ejemplo, un hook `PreToolUse` puede negar una llamada a herramienta y decirle a Claude por qué, o escalarlo al usuario para aprobación:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code lee `permissionDecision` y cancela la llamada a herramienta, luego alimenta `permissionDecisionReason` de vuelta a Claude como retroalimentación. Estas tres opciones son específicas de `PreToolUse`:

- `"allow"`: procede sin mostrar un aviso de permiso
- `"deny"`: cancela la llamada a herramienta y envía la razón a Claude
- `"ask"`: muestra el aviso de permiso al usuario como es normal

Otros eventos usan patrones de decisión diferentes. Por ejemplo, los hooks `PostToolUse` y `Stop` usan un campo `decision: "block"` de nivel superior, mientras que `PermissionRequest` usa `hookSpecificOutput.decision.behavior`. Consulta la [tabla de resumen](#) en la referencia para un desglose completo por evento.

Para hooks `UserPromptSubmit`, usa `additionalContext` en su lugar para inyectar texto en el contexto de Claude. Los hooks basados en prompts (`type: "prompt"`) manejan la salida de manera diferente: consulta [Hooks basados en prompts](#).

Filtra hooks con matchers

Sin un matcher, un hook se activa en cada ocurrencia de su evento. Los matchers te permiten estrecharlo. Por ejemplo, si quieres ejecutar un formateador solo después de ediciones de archivos (no después de cada llamada a herramienta), añade un matcher a tu hook `PostToolUse`:


```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

El matcher `"Edit|Write"` es un patrón regex que coincide con el nombre de la herramienta. El hook solo se activa cuando Claude usa la herramienta `Edit` o `Write`, no cuando usa `Bash`, `Read`, u otra herramienta.

Cada tipo de evento coincide en un campo específico. Los matchers soportan cadenas exactas y patrones regex:

Evento	En qué filtra el matcher	Valores de matcher de ejemplo
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	nombre de herramienta	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	cómo comenzó la sesión	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	por qué terminó la sesión	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_disabled</code> , <code>other</code>
<code>Notification</code>	tipo de notificación	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>

Evento	En qué filtra el matcher	Valores de matcher de ejemplo
<code>SubagentStart</code>	tipo de agente	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> , o nombres de agentes personalizados
<code>PreCompact</code>	qué activó la compactación	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	tipo de agente	los mismos valores que <code>SubagentStart</code>
<code>ConfigChange</code>	fuelle de configuración	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>
<code>UserPromptSubmit</code> , <code>Stop</code> , <code>TeammateIdle</code> , <code>TaskCompleted</code> , <code>WorktreeCreate</code> , <code>WorktreeRemove</code>	sin soporte de matcher	siempre se activa en cada ocurrencia

Algunos ejemplos más mostrando matchers en diferentes tipos de eventos:

Registra cada comando Bash

Coincide solo con llamadas a herramienta `Bash` y registra cada comando en un archivo. El evento `PostToolUse` se activa después de que el comando se completa, por lo que `tool_input.command` contiene lo que se ejecutó. El hook recibe los datos del evento como JSON en stdin, y `jq -r '.tool_input.command'` extrae solo la cadena de comando, que `>>` añade al archivo de registro:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

Coincide con herramientas MCP

Las herramientas MCP usan una convención de nombres diferente a las herramientas integradas: `mcp__<server>__<tool>`, donde `<server>` es el nombre del servidor MCP y `<tool>` es la herramienta que proporciona. Por ejemplo, `mcp__github__search_repositories` o `mcp__filesystem__read_file`. Usa un matcher regex para dirigirse a todas las herramientas de un servidor específico, o coincide entre servidores con un patrón como `mcp__.*__write.*`. Consulta [Coincide con herramientas MCP](#) en la referencia para la lista completa de ejemplos.

El comando a continuación extrae el nombre de la herramienta de la entrada JSON del hook con `jq` y lo escribe en stderr, donde aparece en modo detallado (`Ctrl+O`):

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp_github_.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

Limpia al final de la sesión

El evento `SessionEnd` soporta matchers en la razón por la que terminó la sesión. Este hook solo se activa en `clear` (cuando ejecutas `/clear`), no en salidas normales:

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

Para la sintaxis completa del matcher, consulta la [referencia de Hooks](#).

Configura la ubicación del hook

Dónde añadas un hook determina su ámbito:

Ubicación	Ámbito	Compatible
<code>~/.claude/ settings.json</code>	Todos tus proyectos	No, local a tu máquina
<code>.claude/ settings.json</code>	Proyecto único	Sí, puede ser confirmado en el repositorio
<code>.claude/ settings.local.json</code>	Proyecto único	No, gitignored
Configuración de política gestionada	Organización completa	Sí, controlado por administrador
<code>Plugin hooks/ hooks.json</code>	Cuando el plugin está habilitado	Sí, incluido con el plugin
<code>Skill</code> o <code>agente</code> frontmatter	Mientras el skill o agente está activo	Sí, definido en el archivo del componente

También puedes usar el menú `/hooks` en Claude Code para añadir, eliminar y ver hooks interactivamente. Para desactivar todos los hooks a la vez, usa el botón de alternancia en la parte inferior del menú `/hooks` o establece `"disableAllHooks": true` en tu archivo de configuración.

Los hooks añadidos a través del menú `/hooks` tienen efecto inmediatamente. Si editas archivos de configuración directamente mientras Claude Code está ejecutándose, los cambios no tendrán efecto hasta que los revises en el menú `/hooks` o reinicies tu sesión.

Hooks basados en prompts

Para decisiones que requieren criterio en lugar de reglas deterministas, usa hooks `type: "prompt"`. En lugar de ejecutar un comando de shell, Claude Code envía tu prompt y los datos de entrada del hook a un modelo Claude (Haiku por defecto) para tomar la decisión. Puedes especificar un modelo diferente con el campo `model` si necesitas más capacidad.

El único trabajo del modelo es devolver una decisión sí/no como JSON:

- `"ok": true` : la acción procede
- `"ok": false` : la acción se bloquea. La `"reason"` del modelo se alimenta de vuelta a Claude para que pueda ajustar.

Este ejemplo usa un hook **Stop** para preguntarle al modelo si todas las tareas solicitadas están completas. Si el modelo devuelve **"ok": false**, Claude sigue trabajando y usa la **reason** como su siguiente instrucción:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with
{\\"ok\\": false, \\"reason\\": \\"what remains to be done\\"}."
          }
        ]
      }
    ]
  }
}
```

Para opciones de configuración completas, consulta [Hooks basados en prompts](#) en la referencia.

Hooks basados en agentes

Cuando la verificación requiere inspeccionar archivos o ejecutar comandos, usa hooks **type: "agent"**. A diferencia de los hooks de prompt que hacen una única llamada LLM, los hooks de agente generan un subagente que puede leer archivos, buscar código y usar otras herramientas para verificar condiciones antes de devolver una decisión.

Los hooks de agente usan el mismo formato de respuesta **"ok"** / **"reason"** que los hooks de prompt, pero con un tiempo de espera predeterminado más largo de 60 segundos y hasta 50 turnos de uso de herramientas.

Este ejemplo verifica que las pruebas pasen antes de permitir que Claude se detenga:

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Usa hooks de prompt cuando los datos de entrada del hook son suficientes para tomar una decisión. Usa hooks de agente cuando necesites verificar algo contra el estado real del código base.

Para opciones de configuración completas, consulta [Hooks basados en agentes](#) en la referencia.

HTTP hooks

Usa hooks `type: "http"` para POST de datos de evento a un punto final HTTP en lugar de ejecutar un comando de shell. El punto final recibe el mismo JSON que un hook de comando recibiría en stdin, y devuelve resultados a través del cuerpo de respuesta HTTP usando el mismo formato JSON.

Los HTTP hooks son útiles cuando quieres que un servidor web, función en la nube o servicio externo maneje la lógica del hook: por ejemplo, un servicio de auditoría compartido que registra eventos de uso de herramientas en todo un equipo.

Este ejemplo publica cada uso de herramienta en un servicio de registro local:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}

```

El punto final debe devolver un cuerpo de respuesta JSON usando el mismo [formato de salida](#) que los hooks de comando. Para bloquear una llamada a herramienta, devuelve una respuesta 2xx con los campos `hookSpecificOutput` apropiados. Los códigos de estado HTTP por sí solos no pueden bloquear acciones.

Los valores de encabezado soportan interpolación de variables de entorno usando la sintaxis `$VAR_NAME` o `${VAR_NAME}`. Solo las variables listadas en el array `allowedEnvVars` se resuelven; todas las otras referencias `$VAR` permanecen vacías.

Note:

Los HTTP hooks deben configurarse editando tu JSON de configuración directamente. El menú interactivo `/hooks` solo soporta añadir hooks de comando.

Para opciones de configuración completas y manejo de respuestas, consulta [HTTP hooks](#) en la referencia.

Limitaciones y solución de problemas

Limitaciones

- Los hooks de comando se comunican solo a través de stdout, stderr y códigos de salida. No pueden activar comandos o llamadas a herramientas directamente. Los HTTP hooks se comunican a través del cuerpo de respuesta en su lugar.
- El tiempo de espera del hook es 10 minutos por defecto, configurable por hook con el campo `timeout` (en segundos).
- Los hooks `PostToolUse` no pueden deshacer acciones ya que la herramienta ya se ha ejecutado.
- Los hooks `PermissionRequest` no se activan en [modo no interactivo](#) (`-p`). Usa hooks `PreToolUse` para decisiones de permiso automatizadas.
- Los hooks `Stop` se activan cada vez que Claude termina de responder, no solo en la finalización de tareas. No se activan en interrupciones del usuario.

Hook no se activa

El hook está configurado pero nunca se ejecuta.

- Ejecuta `/hooks` y confirma que el hook aparece bajo el evento correcto
- Verifica que el patrón del matcher coincida exactamente con el nombre de la herramienta (los matchers distinguen mayúsculas de minúsculas)
- Verifica que estés activando el tipo de evento correcto (por ejemplo, `PreToolUse` se activa antes de la ejecución de la herramienta, `PostToolUse` se activa después)
- Si usas hooks `PermissionRequest` en modo no interactivo (`-p`), cambia a `PreToolUse` en su lugar

Error de hook en la salida

Ves un mensaje como “PreToolUse hook error: ...” en la transcripción.

- Tu script salió con un código no cero inesperadamente. Pruébalo manualmente canalizando JSON de muestra:

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $? # Verifica el código de salida
```

- Si ves “command not found”, usa rutas absolutas o `$CLAUDE_PROJECT_DIR` para referenciar scripts
- Si ves “jq: command not found”, instala `jq` o usa Python/Node.js para análisis JSON

- Si el script no se ejecuta en absoluto, hazlo ejecutable: `chmod +x ./my-hook.sh`

`/hooks` no muestra hooks configurados

Editaste un archivo de configuración pero los hooks no aparecen en el menú.

- Reinicia tu sesión o abre `/hooks` para recargar. Los hooks añadidos a través del menú `/hooks` tienen efecto inmediatamente, pero las ediciones manuales de archivos requieren una recarga.
- Verifica que tu JSON sea válido (las comas finales y comentarios no están permitidos)
- Confirma que el archivo de configuración está en la ubicación correcta: `.claude/settings.json` para hooks de proyecto, `~/.claude/settings.json` para hooks globales

El hook Stop se ejecuta para siempre

Claude sigue trabajando en un bucle infinito en lugar de detenerse.

Tu script de hook Stop necesita verificar si ya activó una continuación. Analiza el campo `stop_hook_active` de la entrada JSON y sal temprano si es `true`:

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
    exit 0 # Permite que Claude se detenga
fi
## ... resto de tu lógica de hook
```

Falló la validación JSON

Claude Code muestra un error de análisis JSON aunque tu script de hook produzca JSON válido.

Cuando Claude Code ejecuta un hook, genera un shell que obtiene tu perfil (`~/.zshrc` o `~/.bashrc`). Si tu perfil contiene declaraciones `echo` incondicionales, esa salida se antepone a tu JSON del hook:

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code intenta analizar esto como JSON y falla. Para arreglarlo, envuelve las declaraciones `echo` en tu perfil de shell para que solo se ejecuten en shells interactivos:

```
## En ~/.zshrc o ~/.bashrc
if [[ $- == *i* ]]; then
    echo "Shell ready"
fi
```

La variable `$-` contiene banderas de shell, e `i` significa interactivo. Los hooks se ejecutan en shells no interactivos, por lo que el echo se omite.

Técnicas de depuración

Alterna el modo detallado con `Ctrl+O` para ver la salida del hook en la transcripción, o ejecuta `claude --debug` para detalles de ejecución completos incluyendo qué hooks coincidieron y sus códigos de salida.

Aprende más

- [Referencia de Hooks](#): esquemas de eventos completos, formato de salida JSON, hooks asíncronos y hooks de herramientas MCP
- [Consideraciones de seguridad](#): revisa antes de desplegar hooks en entornos compartidos o de producción
- [Ejemplo de validador de comandos Bash](#): implementación de referencia completa

Conectar Claude Code a herramientas mediante MCP

Aprenda cómo conectar Claude Code a sus herramientas con el Model Context Protocol.

Claude Code puede conectarse a cientos de herramientas externas y fuentes de datos a través del [Model Context Protocol \(MCP\)](#), un estándar de código abierto para integración de IA con herramientas. Los servidores MCP dan a Claude Code acceso a sus herramientas, bases de datos y APIs.

Qué puede hacer con MCP

Con servidores MCP conectados, puede pedirle a Claude Code que:

- **Implemente características desde rastreadores de problemas:** “Agregue la característica descrita en el problema JIRA ENG-4521 y cree un PR en GitHub.”
- **Analice datos de monitoreo:** “Verifique Sentry y Statsig para verificar el uso de la característica descrita en ENG-4521.”
- **Consulte bases de datos:** “Encuentre correos electrónicos de 10 usuarios aleatorios que utilizaron la característica ENG-4521, basándose en nuestra base de datos PostgreSQL.”
- **Integre diseños:** “Actualice nuestra plantilla de correo electrónico estándar basándose en los nuevos diseños de Figma que se publicaron en Slack”
- **Automatice flujos de trabajo:** “Cree borradores de Gmail invitando a estos 10 usuarios a una sesión de retroalimentación sobre la nueva característica.”

Servidores MCP populares

Aquí hay algunos servidores MCP comúnmente utilizados que puede conectar a Claude Code:

Warning:

Use servidores MCP de terceros bajo su propio riesgo - Anthropic no ha verificado la corrección o seguridad de todos estos servidores. Asegúrese de confiar en los servidores MCP que está instalando. Tenga especial cuidado al usar servidores MCP que podrían obtener contenido no confiable, ya que estos pueden exponerlo al riesgo de inyección de indicaciones.

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	<pre>claude mcp add -- transport http notion https:// mcp.notion.com/mcp</pre>
Canva	Search, create, autofill, and export Canva designs	<pre>claude mcp add -- transport http canva https:// mcp.canva.com/mcp</pre>
Figma	Generate diagrams and better code from Figma context	<pre>claude mcp add -- transport http figma- remote-mcp https:// mcp.figma.com/mcp</pre>
Atlassian	Access Jira & Confluence from Claude	<pre>claude mcp add -- transport http atlassian https:// mcp.atlassian.com/v1/ mcp</pre>
Linear	Manage issues, projects & team workflows in Linear	<pre>claude mcp add -- transport http linear https:// mcp.linear.app/mcp</pre>
monday.com	Manage projects, boards, and workflows in monday.com	<pre>claude mcp add -- transport http monday https:// mcp.monday.com/mcp</pre>
Intercom	Access to Intercom data for better customer insights	<pre>claude mcp add -- transport http intercom https:// mcp.intercom.com/mcp</pre>

Server	Description	Command
Vercel	Analyze, debug, and manage projects and deployments	<code>claude mcp add --transport http vercel https://mcp.vercel.com</code>
Granola	The AI notepad for meetings	<code>claude mcp add --transport http granola https://mcp.granola.ai/mcp</code>
Asana	Connect to Asana to coordinate tasks, projects, and goals	<code>claude mcp add --transport streamable-http asana https://mcp.asana.com/v2/mcp</code>
Miro	Access and create new content on Miro boards	<code>claude mcp add --transport http miro https://mcp.miro.com/</code>
Sentry	Search, query, and debug errors intelligently	<code>claude mcp add --transport http sentry https://mcp.sentry.dev/mcp</code>
Supabase	Manage databases, authentication, and storage	<code>claude mcp add --transport http supabase https://mcp.supabase.com/mcp</code>
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	<code>claude mcp add --transport http hugging-face https://huggingface.co/mcp</code>
Context7	Up-to-date docs for LLMs and AI code editors	<code>claude mcp add --transport http context7 https://mcp.context7.com/mcp</code>

Server	Description	Command
Stripe	Payment processing and financial infrastructure tools	<code>claude mcp add --transport http stripe https://mcp.stripe.com</code>
Microsoft Learn	Search trusted Microsoft docs to power your development	<code>claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp</code>
Clay	Find prospects. Research accounts. Personalize outreach	<code>claude mcp add --transport http clay https://api.clay.com/v3/mcp</code>
Webflow	Manage Webflow CMS, pages, assets and sites	<code>claude mcp add --transport http webflow https://mcp.webflow.com/mcp</code>
Cloudflare	Build applications with compute, storage, and AI	<code>claude mcp add --transport http cloudflare https://bindings.mcp.cloudflare.com/mcp</code>
Ramp	Search, access, and analyze your Ramp financial data	<code>claude mcp add --transport http ramp https://ramp-mcp-remote.ramp.com/mcp</code>
ZoomInfo	Enrich contacts & accounts with GTM intelligence	<code>claude mcp add --transport http zoominfo https://mcp.zoominfo.com/mcp</code>
Netlify	Create, deploy, manage, and secure websites on Netlify	<code>claude mcp add --transport http netlify https://netlify-mcp.netlify.app/mcp</code>

Server	Description	Command
Make	Run Make scenarios and manage your Make account	<code>claude mcp add --transport http make https://mcp.make.com</code>
GoDaddy	Search domains and check availability	<code>claude mcp add --transport http godaddy https://api.godaddy.com/v1/domains/mcp</code>
Google Cloud BigQuery	BigQuery: Advanced analytical insights for agents	<code>claude mcp add --transport http bigquery https://bigquery.googleapis.com/mcp</code>
PayPal	Access PayPal payments platform	<code>claude mcp add --transport http paypal https://mcp.paypal.com/mcp</code>
PostHog	Query, analyze, and manage your PostHog insights	<code>claude mcp add --transport http posthog https://mcp.posthog.com/mcp</code>
Similarweb	Real time web, mobile app, and market data	<code>claude mcp add --transport http similarweb https://mcp.similarweb.com</code>
Crypto.com	Real time prices, orders, charts, and more for crypto	<code>claude mcp add --transport http crypto.com https://mcp.crypto.com/market-data/mcp</code>
Attio	Search, manage, and update your Attio CRM from Claude	<code>claude mcp add --transport http attio https://mcp.attio.com/mcp</code>

Server	Description	Command
Trivago	Find your ideal hotel at the best price	<code>claude mcp add --transport http trivago https://mcp.trivago.com/mcp</code>
Jam	Record screen and collect automatic context for issues	<code>claude mcp add --transport http jam https://mcp.jam.dev/mcp</code>
Consensus	Explore scientific research	<code>claude mcp add --transport http consensus https://mcp.consensus.app/mcp</code>
Clockwise	Advanced scheduling and time management for work	<code>claude mcp add --transport http clockwise https://mcp.getclockwise.com/mcp</code>
Square	Search and manage transaction, merchant, and payment data	<code>claude mcp add --transport sse square https://mcp.squareup.com/sse</code>
Egnyte	Securely access and analyze Egnyte content	<code>claude mcp add --transport http egnyte https://mcp-server.egnyte.com/mcp</code>
Pylon	Search and manage Pylon support issues	<code>claude mcp add --transport http pylon https://mcp.usepylon.com/</code>
Honeycomb	Query and explore observability data and SLOs	<code>claude mcp add --transport http honeycomb https://mcp.honeycomb.io/mcp</code>

Note:

¿Necesita una integración específica? [Encuentre cientos más servidores MCP en GitHub](#), o cree el suyo propio usando el [MCP SDK](#).

Instalación de servidores MCP

Los servidores MCP se pueden configurar de tres formas diferentes según sus necesidades:

Opción 1: Agregar un servidor HTTP remoto

Los servidores HTTP son la opción recomendada para conectarse a servidores MCP remotos. Este es el transporte más ampliamente soportado para servicios basados en la nube.

```
## Sintaxis básica
claude mcp add --transport http <name> <url>

## Ejemplo real: Conectar a Notion
claude mcp add --transport http notion https://mcp.notion.com/mcp

## Ejemplo con token Bearer
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

Opción 2: Agregar un servidor SSE remoto

Warning:

El transporte SSE (Server-Sent Events) está deprecado. Use servidores HTTP en su lugar, donde estén disponibles.

```
## Sintaxis básica
claude mcp add --transport sse <name> <url>

## Ejemplo real: Conectar a Asana
claude mcp add --transport sse asana https://mcp.asana.com/sse

## Ejemplo con encabezado de autenticación
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

Opción 3: Agregar un servidor stdio local

Los servidores stdio se ejecutan como procesos locales en su máquina. Son ideales para herramientas que necesitan acceso directo al sistema o scripts personalizados.

```
## Sintaxis básica
claude mcp add [options] <name> -- <command> [args...]

## Ejemplo real: Agregar servidor Airtable
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

Note:

Importante: Orden de opciones

Todas las opciones (`--transport` , `--env` , `--scope` , `--header`) deben venir **antes** del nombre del servidor. El `--` (doble guión) luego separa el nombre del servidor del comando y los argumentos que se pasan al servidor MCP.

Por ejemplo:

- `claude mcp add --transport stdio myserver -- npx server` → ejecuta `npx server`
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → ejecuta `python server.py --port 8080` con `KEY=value` en el entorno

Esto evita conflictos entre las banderas de Claude y las banderas del servidor.

Gestión de sus servidores

Una vez configurados, puede gestionar sus servidores MCP con estos comandos:

```
## Listar todos los servidores configurados
claude mcp list

## Obtener detalles para un servidor específico
claude mcp get github

## Eliminar un servidor
claude mcp remove github

## (dentro de Claude Code) Verificar estado del servidor
/mcp
```

Actualizaciones dinámicas de herramientas

Claude Code admite notificaciones `list_changed` de MCP, permitiendo que los servidores MCP actualicen dinámicamente sus herramientas disponibles, indicaciones y recursos sin requerir que se desconecte y reconecte. Cuando un servidor MCP envía una notificación `list_changed`, Claude Code actualiza automáticamente las capacidades disponibles de ese servidor.

Tip:

Consejos:

- Use la bandera `--scope` para especificar dónde se almacena la configuración:
- `local` (predeterminado): Disponible solo para usted en el proyecto actual (se llamaba `project` en versiones anteriores)
- `project`: Compartido con todos en el proyecto a través del archivo `.mcp.json`
- `user`: Disponible para usted en todos los proyectos (se llamaba `global` en versiones anteriores)
- Establezca variables de entorno con banderas `--env` (por ejemplo, `--env KEY=value`)
- Configure el tiempo de espera de inicio del servidor MCP usando la variable de entorno `MCP_TIMEOUT` (por ejemplo, `MCP_TIMEOUT=10000 claude` establece un tiempo de espera de 10 segundos)
- Claude Code mostrará una advertencia cuando la salida de la herramienta MCP exceda 10,000 tokens. Para aumentar este límite, establezca la variable de entorno `MAX_MCP_OUTPUT_TOKENS` (por ejemplo, `MAX_MCP_OUTPUT_TOKENS=50000`)

- Use `/mcp` para autenticarse con servidores remotos que requieren autenticación OAuth 2.0

Warning:

Usuarios de Windows: En Windows nativo (no WSL), los servidores MCP locales que usan `npx` requieren el contenedor `cmd /c` para garantizar la ejecución adecuada.

```
## Esto crea command="cmd" que Windows puede ejecutar
claude mcp add --transport stdio my-server -- cmd /c npx -y @some/package
```

Sin el contenedor `cmd /c`, encontrará errores de “Connection closed” porque Windows no puede ejecutar directamente `npx`. (Vea la nota anterior para una explicación del parámetro `--`.)

Servidores MCP proporcionados por plugins

Los [plugins](#) pueden agrupar servidores MCP, proporcionando automáticamente herramientas e integraciones cuando el plugin está habilitado. Los servidores MCP de plugins funcionan de manera idéntica a los servidores configurados por el usuario.

Cómo funcionan los servidores MCP de plugins:

- Los plugins definen servidores MCP en `.mcp.json` en la raíz del plugin o en línea en `plugin.json`
- Cuando un plugin está habilitado, sus servidores MCP se inician automáticamente
- Las herramientas MCP del plugin aparecen junto a las herramientas MCP configuradas manualmente
- Los servidores de plugins se gestionan a través de la instalación de plugins (no mediante comandos `/mcp`)

Ejemplo de configuración MCP de plugin:

En `.mcp.json` en la raíz del plugin:

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

O en línea en `plugin.json` :

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

Características de MCP de plugins:

- **Ciclo de vida automático:** Los servidores se inician cuando el plugin se habilita, pero debe reiniciar Claude Code para aplicar cambios en el servidor MCP (habilitación o deshabilitación)
- **Variables de entorno:** Use `${CLAUDE_PLUGIN_ROOT}` para rutas relativas al plugin
- **Acceso a variables de entorno del usuario:** Acceso a las mismas variables de entorno que los servidores configurados manualmente
- **Múltiples tipos de transporte:** Soporte para transportes stdio, SSE e HTTP (el soporte de transporte puede variar según el servidor)

Visualización de servidores MCP de plugins:

```
## Dentro de Claude Code, vea todos los servidores MCP incluyendo los de plugins  
/mcp
```

Los servidores de plugins aparecen en la lista con indicadores que muestran que provienen de plugins.

Beneficios de los servidores MCP de plugins:

- **Distribución agrupada:** Herramientas y servidores empaquetados juntos
- **Configuración automática:** No se necesita configuración manual de MCP
- **Consistencia del equipo:** Todos obtienen las mismas herramientas cuando se instala el plugin

Vea la [referencia de componentes de plugins](#) para detalles sobre cómo agrupar servidores MCP con plugins.

Alcances de instalación de MCP

Los servidores MCP se pueden configurar en tres niveles de alcance diferentes, cada uno sirviendo propósitos distintos para gestionar la accesibilidad del servidor y el intercambio. Comprender estos alcances le ayuda a determinar la mejor manera de configurar servidores para sus necesidades específicas.

Alcance local

Los servidores con alcance local representan el nivel de configuración predeterminado y se almacenan en `~/.claude.json` bajo la ruta de su proyecto. Estos servidores permanecen privados para usted y solo son accesibles cuando trabaja dentro del directorio del proyecto actual. Este alcance es ideal para servidores de desarrollo personal, configuraciones experimentales o servidores que contienen credenciales sensibles que no deben compartirse.

Note:

El término “alcance local” para servidores MCP difiere de la configuración local general. Los servidores MCP con alcance local se almacenan en `~/.claude.json` (su directorio de inicio), mientras que la configuración local general usa `.claude/settings.local.json` (en el directorio del proyecto). Vea [Configuración](#) para detalles sobre ubicaciones de archivos de configuración.

```
## Agregar un servidor con alcance local (predeterminado)
claude mcp add --transport http stripe https://mcp.stripe.com

## Especificar explícitamente alcance local
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

Alcance de proyecto

Los servidores con alcance de proyecto habilitan la colaboración en equipo al almacenar configuraciones en un archivo `.mcp.json` en el directorio raíz de su proyecto. Este archivo está diseñado para ser verificado en el control de versiones, asegurando que todos los miembros del equipo tengan acceso a las mismas herramientas y servicios MCP. Cuando agrega un servidor con alcance de proyecto, Claude Code crea o actualiza automáticamente este archivo con la estructura de configuración apropiada.

```
## Agregar un servidor con alcance de proyecto
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

El archivo `.mcp.json` resultante sigue un formato estandarizado:

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

Por razones de seguridad, Claude Code solicita aprobación antes de usar servidores con alcance de proyecto desde archivos `.mcp.json`. Si necesita restablecer estas opciones de aprobación, use el comando `claude mcp reset-project-choices`.

Alcance de usuario

Los servidores con alcance de usuario se almacenan en `~/.claude.json` y proporcionan accesibilidad entre proyectos, haciéndolos disponibles en todos los proyectos en su máquina mientras permanecen privados para su cuenta de usuario. Este alcance funciona bien para servidores de utilidad personal, herramientas de desarrollo o servicios que usa frecuentemente en diferentes proyectos.

```
## Agregar un servidor de usuario
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/anthropic
```

Elegir el alcance correcto

Seleccione su alcance basándose en:

- **Alcance local:** Servidores personales, configuraciones experimentales o credenciales sensibles específicas de un proyecto
- **Alcance de proyecto:** Servidores compartidos por el equipo, herramientas específicas del proyecto o servicios requeridos para la colaboración
- **Alcance de usuario:** Utilidades personales necesarias en múltiples proyectos, herramientas de desarrollo o servicios frecuentemente utilizados

Note:

¿Dónde se almacenan los servidores MCP?

- **Alcance de usuario y local:** `~/.claude.json` (en el campo `mcpServers` o bajo rutas de proyecto)
- **Alcance de proyecto:** `.mcp.json` en la raíz de su proyecto (verificado en el control de versiones)
- **Gestionado:** `managed-mcp.json` en directorios del sistema (vea [Configuración MCP gestionada](#))

Jerarquía de alcance y precedencia

Las configuraciones de servidores MCP siguen una jerarquía de precedencia clara. Cuando existen servidores con el mismo nombre en múltiples alcances, el sistema resuelve conflictos priorizando primero los servidores con alcance local, seguidos por los servidores con alcance de proyecto y finalmente los servidores con alcance de usuario. Este diseño asegura que las configuraciones personales puedan anular las compartidas cuando sea necesario.

Expansión de variables de entorno en `.mcp.json`

Claude Code admite la expansión de variables de entorno en archivos `.mcp.json`, permitiendo que los equipos compartan configuraciones mientras mantienen flexibilidad para rutas específicas de máquinas y valores sensibles como claves API.

Sintaxis soportada:

- `${VAR}` - Se expande al valor de la variable de entorno `VAR`
- `${VAR:-default}` - Se expande a `VAR` si está establecida, de lo contrario usa `default`

Ubicaciones de expansión: Las variables de entorno se pueden expandir en:

- `command` - La ruta del ejecutable del servidor
- `args` - Argumentos de línea de comandos
- `env` - Variables de entorno pasadas al servidor
- `url` - Para tipos de servidor HTTP
- `headers` - Para autenticación de servidor HTTP

Ejemplo con expansión de variables:

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

Si una variable de entorno requerida no está establecida y no tiene un valor predeterminado, Claude Code no podrá analizar la configuración.

Ejemplos prácticos

```
{/* ### Ejemplo: Automatizar pruebas de navegador con Playwright
```

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

Luego escriba y ejecute pruebas de navegador:

```
Test if the login flow works with test@example.com
```

```
Take a screenshot of the checkout page on mobile
```

```
Verify that the search feature returns results
``` */}

Ejemplo: Monitorear errores con Sentry

```bash theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Autentíquese con su cuenta de Sentry:

```
/mcp
```

Luego depure problemas de producción:

```
¿Cuáles son los errores más comunes en las últimas 24 horas?
```

```
Muéstrame el seguimiento de pila para el error ID abc123
```

```
¿Qué despliegue introdujo estos nuevos errores?
```

Ejemplo: Conectar a GitHub para revisiones de código

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Autentíquese si es necesario seleccionando “Authenticate” para GitHub:

```
/mcp
```

Luego trabaje con GitHub:

```
Revise el PR #456 y sugiera mejoras
```

```
Cree un nuevo problema para el error que acabamos de encontrar
```

```
Muéstrame todos los PR abiertos asignados a mí
```

Ejemplo: Consultar su base de datos PostgreSQL

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \  
--dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

Luego consulte su base de datos de forma natural:

```
¿Cuál es nuestro ingreso total este mes?
```

```
Muéstrame el esquema para la tabla de pedidos
```

```
Encuentre clientes que no han realizado una compra en 90 días
```

Autenticarse con servidores MCP remotos

Muchos servidores MCP basados en la nube requieren autenticación. Claude Code admite OAuth 2.0 para conexiones seguras.

Step 1: Agregar el servidor que requiere autenticación

Por ejemplo:

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Use el comando `/mcp` dentro de Claude Code

En Claude Code, use el comando:

```
/mcp
```

Luego siga los pasos en su navegador para iniciar sesión.

Tip:

Consejos:

- Los tokens de autenticación se almacenan de forma segura y se actualizan automáticamente
- Use “Clear authentication” en el menú `/mcp` para revocar el acceso
- Si su navegador no se abre automáticamente, copie la URL proporcionada y ábrala manualmente
- Si el redireccionamiento del navegador falla con un error de conexión después de autenticarse, pegue la URL de devolución de llamada completa de la barra de direcciones de su navegador en el indicador de URL que aparece en Claude Code
- La autenticación OAuth funciona con servidores HTTP

Usar un puerto de devolución de llamada OAuth fijo

Algunos servidores MCP requieren un URI de redireccionamiento específico registrado de antemano. De forma predeterminada, Claude Code elige un puerto disponible aleatorio para la devolución de llamada de OAuth. Use `--callback-port` para fijar el puerto de modo que coincida con un URI de redireccionamiento preregistrado de la forma `http://localhost:PORT/callback`.

Puede usar `--callback-port` por sí solo (con registro dinámico de clientes) o junto con `--client-id` (con credenciales preconfiguradas).

```
## Puerto de devolución de llamada fijo con registro dinámico de clientes
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

Usar credenciales OAuth preconfiguradas

Algunos servidores MCP no admiten configuración automática de OAuth. Si ve un error como “Incompatible auth server: does not support dynamic client registration”, el servidor requiere credenciales preconfiguradas. Primero registre una aplicación OAuth a través del portal de desarrolladores del servidor, luego proporcione las credenciales al agregar el servidor.

Step 1: Registrar una aplicación OAuth con el servidor

Cree una aplicación a través del portal de desarrolladores del servidor y anote su ID de cliente y secreto de cliente.

Muchos servidores también requieren un URI de redireccionamiento. Si es así, elija un puerto y registre un URI de redireccionamiento en el formato `http://localhost:PORT/callback`. Use ese mismo puerto con `--callback-port` en el siguiente paso.

Step 2: Agregar el servidor con sus credenciales

Elija uno de los siguientes métodos. El puerto utilizado para `--callback-port` puede ser cualquier puerto disponible. Solo necesita coincidir con el URI de redireccionamiento que registró en el paso anterior.

claude mcp add

Use `--client-id` para pasar el ID de cliente de su aplicación. La bandera `--client-secret` solicita el secreto con entrada enmascarada:

```
claude mcp add --transport http \
  --client-id your-client-id --client-secret --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

claude mcp add-json

Incluya el objeto `oauth` en la configuración JSON y pase `--client-secret` como una bandera separada:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' \
  --client-secret
```

claude mcp add-json (solo puerto de devolución de llamada)

Use `--callback-port` sin un ID de cliente para fijar el puerto mientras usa registro dinámico de clientes:

```
claude mcp add-json my-server \  
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":  
  {"callbackPort":8080}}'
```

CI / variable de entorno

Establezca el secreto a través de una variable de entorno para omitir el indicador interactivo:

```
MCP_CLIENT_SECRET=your-secret claude mcp add --transport http \  
  --client-id your-client-id --client-secret --callback-port 8080 \  
  my-server https://mcp.example.com/mcp
```

Step 3: Autenticarse en Claude Code

Ejecute `/mcp` en Claude Code y siga el flujo de inicio de sesión del navegador.

Tip:

Consejos:

- El secreto del cliente se almacena de forma segura en su llavero del sistema (macOS) o un archivo de credenciales, no en su configuración
- Si el servidor usa un cliente OAuth público sin secreto, use solo `--client-id` sin `--client-secret`
- `--callback-port` se puede usar con o sin `--client-id`
- Estas banderas solo se aplican a transportes HTTP y SSE. No tienen efecto en servidores stdio
- Use `claude mcp get <name>` para verificar que las credenciales OAuth estén configuradas para un servidor

Anular el descubrimiento de metadatos de OAuth

Si su servidor MCP devuelve errores en el punto final de metadatos de OAuth estándar (`/.well-known/oauth-authorization-server`) pero expone un punto final OIDC funcional, puede indicarle a Claude Code que obtenga metadatos de OAuth directamente desde una URL que especifique, omitiendo la cadena de descubrimiento estándar.

Establezca `authServerMetadataUrl` en el objeto `oauth` de la configuración de su servidor en `.mcp.json`:

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-configuration"
      }
    }
  }
}
```

La URL debe usar `https://`. Esta opción requiere Claude Code v2.1.64 o posterior.

Agregar servidores MCP desde configuración JSON

Si tiene una configuración JSON para un servidor MCP, puede agregarla directamente:

Step 1: Agregar un servidor MCP desde JSON

```
## Sintaxis básica
claude mcp add-json <name> '<json>'

## Ejemplo: Agregar un servidor HTTP con configuración JSON
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/mcp","headers":{"Authorization":"Bearer token"}}'

## Ejemplo: Agregar un servidor stdio con configuración JSON
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## Ejemplo: Agregar un servidor HTTP con credenciales OAuth preconfiguradas
claude mcp add-json my-server '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' --client-secret
```


Step 2: Verificar que el servidor fue agregado

```
claude mcp get weather-api
```

Tip:

Consejos:

- Asegúrese de que el JSON esté correctamente escapado en su shell
- El JSON debe cumplir con el esquema de configuración del servidor MCP
- Puede usar `--scope user` para agregar el servidor a su configuración de usuario en lugar de la específica del proyecto

Importar servidores MCP desde Claude Desktop

Si ya ha configurado servidores MCP en Claude Desktop, puede importarlos:

Step 1: Importar servidores desde Claude Desktop

```
## Sintaxis básica
claude mcp add-from-claude-desktop
```

Step 2: Seleccionar qué servidores importar

Después de ejecutar el comando, verá un diálogo interactivo que le permite seleccionar qué servidores desea importar.

Step 3: Verificar que los servidores fueron importados

```
claude mcp list
```

Tip:

Consejos:

- Esta característica solo funciona en macOS y Windows Subsystem for Linux (WSL)
- Lee el archivo de configuración de Claude Desktop desde su ubicación estándar en esas plataformas
- Use la bandera `--scope user` para agregar servidores a su configuración de usuario
- Los servidores importados tendrán los mismos nombres que en Claude Desktop

- Si ya existen servidores con los mismos nombres, obtendrán un sufijo numérico (por ejemplo, `server_1`)

Usar servidores MCP desde Claude.ai

Si ha iniciado sesión en Claude Code con una cuenta de [Claude.ai](https://claude.ai), los servidores MCP que ha agregado en Claude.ai están automáticamente disponibles en Claude Code:

Step 1: Configurar servidores MCP en Claude.ai

Agregue servidores en claude.ai/settings/connectors. En planes de Equipo y Empresa, solo los administradores pueden agregar servidores.

Step 2: Autenticar el servidor MCP

Complete los pasos de autenticación requeridos en Claude.ai.

Step 3: Ver y gestionar servidores en Claude Code

En Claude Code, use el comando:

```
/mcp
```

Los servidores de Claude.ai aparecen en la lista con indicadores que muestran que provienen de Claude.ai.

Para desactivar servidores MCP de Claude.ai en Claude Code, establezca la variable de entorno `ENABLE_CLAUDEAI_MCP_SERVERS` en `false` :

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Usar Claude Code como servidor MCP

Puede usar Claude Code mismo como servidor MCP al que otras aplicaciones pueden conectarse:

```
## Iniciar Claude como servidor MCP stdio
claude mcp serve
```

Puede usar esto en Claude Desktop agregando esta configuración a `claude_desktop_config.json`:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

Configurar la ruta del ejecutable: El campo `command` debe hacer referencia al ejecutable de Claude Code. Si el comando `claude` no está en el PATH del sistema, deberá especificar la ruta completa al ejecutable.

Para encontrar la ruta completa:

```
which claude
```

Luego use la ruta completa en su configuración:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Sin la ruta correcta del ejecutable, encontrará errores como `spawn claude ENOENT`.

Tip:

Consejos:

- El servidor proporciona acceso a las herramientas de Claude como View, Edit, LS, etc.

- En Claude Desktop, intente pedirle a Claude que lea archivos en un directorio, haga ediciones y más.
- Tenga en cuenta que este servidor MCP solo expone las herramientas de Claude Code a su cliente MCP, por lo que su propio cliente es responsable de implementar la confirmación del usuario para llamadas de herramientas individuales.

Límites de salida de MCP y advertencias

Cuando las herramientas MCP producen salidas grandes, Claude Code ayuda a gestionar el uso de tokens para evitar abrumar el contexto de su conversación:

- **Umbral de advertencia de salida:** Claude Code muestra una advertencia cuando la salida de cualquier herramienta MCP excede 10,000 tokens
- **Límite configurable:** Puede ajustar los tokens de salida MCP máximos permitidos usando la variable de entorno `MAX_MCP_OUTPUT_TOKENS`
- **Límite predeterminado:** El máximo predeterminado es 25,000 tokens

Para aumentar el límite para herramientas que producen salidas grandes:

```
## Establecer un límite más alto para salidas de herramientas MCP
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

Esto es particularmente útil cuando se trabaja con servidores MCP que:

- Consultan grandes conjuntos de datos o bases de datos
- Generan reportes o documentación detallados
- Procesan archivos de registro extensos o información de depuración

Warning:

Si frecuentemente encuentra advertencias de salida con servidores MCP específicos, considere aumentar el límite o configurar el servidor para paginar o filtrar sus respuestas.

Usar recursos MCP

Los servidores MCP pueden exponer recursos que puede referenciar usando menciones @, similar a cómo referencia archivos.

Referenciar recursos MCP

Step 1: Listar recursos disponibles

Escriba `@` en su indicación para ver los recursos disponibles de todos los servidores MCP conectados. Los recursos aparecen junto a los archivos en el menú de autocompletado.

Step 2: Referenciar un recurso específico

Use el formato `@server:protocol://resource/path` para referenciar un recurso:

¿Puede analizar `@github:issue://123` y sugerir una solución?

Por favor revise la documentación de API en `@docs:file://api/authentication`

Step 3: Múltiples referencias de recursos

Puede referenciar múltiples recursos en una sola indicación:

Compare `@postgres:schema://users` con `@docs:file://database/user-model`

Tip:

Consejos:

- Los recursos se obtienen automáticamente e incluyen como adjuntos cuando se referencian
- Las rutas de recursos son búsquedas difusas en el autocompletado de menciones `@`
- Claude Code proporciona automáticamente herramientas para listar y leer recursos MCP cuando los servidores los admiten
- Los recursos pueden contener cualquier tipo de contenido que proporcione el servidor MCP (texto, JSON, datos estructurados, etc.)

Escalar con MCP Tool Search

Cuando tiene muchos servidores MCP configurados, las definiciones de herramientas pueden consumir una porción significativa de su ventana de contexto. MCP Tool Search resuelve esto cargando herramientas bajo demanda en lugar de precargarlas todas.

Cómo funciona

Claude Code habilita automáticamente Tool Search cuando las descripciones de sus herramientas MCP consumirían más del 10% de la ventana de contexto. Puede [ajustar este umbral](#) o desactivar completamente la búsqueda de herramientas. Cuando se activa:

1. Las herramientas MCP se difieren en lugar de cargarse en el contexto de antemano
2. Claude usa una herramienta de búsqueda para descubrir herramientas MCP relevantes cuando sea necesario
3. Solo se cargan en el contexto las herramientas que Claude realmente necesita
4. Las herramientas MCP continúan funcionando exactamente como antes desde su perspectiva

Para autores de servidores MCP

Si está construyendo un servidor MCP, el campo de instrucciones del servidor se vuelve más útil con Tool Search habilitado. Las instrucciones del servidor ayudan a Claude a entender cuándo buscar sus herramientas, similar a cómo funcionan las [skills](#).

Agregue instrucciones claras y descriptivas del servidor que expliquen:

- Qué categoría de tareas manejan sus herramientas
- Cuándo Claude debe buscar sus herramientas
- Capacidades clave que proporciona su servidor

Configurar búsqueda de herramientas

La búsqueda de herramientas se ejecuta en modo automático de forma predeterminada, lo que significa que se activa solo cuando las definiciones de sus herramientas MCP exceden el umbral de contexto. Si tiene pocas herramientas, se cargan normalmente sin búsqueda de herramientas. Esta característica requiere modelos que admitan bloques

`tool_reference` : Sonnet 4 y posterior, u Opus 4 y posterior. Los modelos Haiku no admiten búsqueda de herramientas.

Controle el comportamiento de búsqueda de herramientas con la variable de entorno

`ENABLE_TOOL_SEARCH` :

Valor	Comportamiento
<code>auto</code>	Se activa cuando las herramientas MCP exceden el 10% del contexto (predeterminado)
<code>auto:<N></code>	Se activa en umbral personalizado, donde <code><N></code> es un porcentaje (p. ej., <code>auto:5</code> para 5%)

Valor	Comportamiento
<code>true</code>	Siempre habilitado
<code>false</code>	Deshabilitado, todas las herramientas MCP cargadas de antemano

```
## Usar un umbral personalizado del 5%
ENABLE_TOOL_SEARCH=auto:5 claude

## Desactivar búsqueda de herramientas completamente
ENABLE_TOOL_SEARCH=false claude
```

O establezca el valor en su [campo env de settings.json](#).

También puede desactivar la herramienta MCPSearch específicamente usando la configuración `disallowedTools`:

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

Usar indicaciones MCP como comandos

Los servidores MCP pueden exponer indicaciones que se vuelven disponibles como comandos en Claude Code.

Ejecutar indicaciones MCP

Step 1: Descubrir indicaciones disponibles

Escriba `/` para ver todos los comandos disponibles, incluyendo los de servidores MCP. Las indicaciones MCP aparecen con el formato `/mcp_servername_promptname`.

Step 2: Ejecutar una indicación sin argumentos

```
/mcp_github_list_prs
```

Step 3: Ejecutar una indicación con argumentos

Muchas indicaciones aceptan argumentos. Páselos separados por espacios después del comando:

```
/mcp_github__pr_review 456
```

```
/mcp_jira__create_issue "Bug en flujo de inicio de sesión" high
```

Tip:

Consejos:

- Las indicaciones MCP se descubren dinámicamente desde servidores conectados
- Los argumentos se analizan basándose en los parámetros definidos de la indicación
- Los resultados de la indicación se inyectan directamente en la conversación
- Los nombres de servidor e indicación se normalizan (los espacios se convierten en guiones bajos)

Configuración MCP gestionada

Para organizaciones que necesitan control centralizado sobre servidores MCP, Claude Code admite dos opciones de configuración:

1. **Control exclusivo con `managed-mcp.json`** : Implemente un conjunto fijo de servidores MCP que los usuarios no pueden modificar ni extender
2. **Control basado en políticas con listas de permitidos/bloqueados**: Permita que los usuarios agreguen sus propios servidores, pero restrinja cuáles están permitidos

Estas opciones permiten a los administradores de TI:

- **Controlar a qué servidores MCP pueden acceder los empleados**: Implemente un conjunto estandarizado de servidores MCP aprobados en toda la organización
- **Prevenir servidores MCP no autorizados**: Restrinja a los usuarios de agregar servidores MCP no aprobados
- **Desactivar MCP completamente**: Elimine completamente la funcionalidad MCP si es necesario

Opción 1: Control exclusivo con managed-mcp.json

Cuando implementa un archivo `managed-mcp.json`, toma **control exclusivo** sobre todos los servidores MCP. Los usuarios no pueden agregar, modificar ni usar ningún servidor MCP que no esté definido en este archivo. Este es el enfoque más simple para organizaciones que desean control completo.

Los administradores del sistema implementan el archivo de configuración en un directorio de todo el sistema:

- macOS: `/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux y WSL: `/etc/claude-code/managed-mcp.json`
- Windows: `C:\Program Files\ClaudeCode\managed-mcp.json`

Note:

Estas son rutas de todo el sistema (no directorios de inicio de usuario como `~/Library/...`) que requieren privilegios de administrador. Están diseñadas para ser implementadas por administradores de TI.

El archivo `managed-mcp.json` usa el mismo formato que un archivo `.mcp.json` estándar:

```
{
  "mcpServers": {
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "sentry": {
      "type": "http",
      "url": "https://mcp.sentry.dev/mcp"
    },
    "company-internal": {
      "type": "stdio",
      "command": "/usr/local/bin/company-mcp-server",
      "args": ["--config", "/etc/company/mcp-config.json"],
      "env": {
        "COMPANY_API_URL": "https://internal.company.com"
      }
    }
  }
}
```

Opción 2: Control basado en políticas con listas de permitidos y bloqueados

En lugar de tomar control exclusivo, los administradores pueden permitir que los usuarios configuren sus propios servidores MCP mientras aplican restricciones sobre qué servidores están permitidos. Este enfoque usa `allowedMcpServers` y `deniedMcpServers` en el [archivo de configuración gestionada](#).

Note:

Elegir entre opciones: Use la Opción 1 (`managed-mcp.json`) cuando desee implementar un conjunto fijo de servidores sin personalización del usuario. Use la Opción 2 (listas de permitidos/bloqueados) cuando desee permitir que los usuarios agreguen sus propios servidores dentro de restricciones de política.

Opciones de restricción

Cada entrada en la lista de permitidos o bloqueados puede restringir servidores de tres formas:

1. **Por nombre de servidor** (`serverName`): Coincide con el nombre configurado del servidor
2. **Por comando** (`serverCommand`): Coincide con el comando exacto y los argumentos utilizados para iniciar servidores stdio
3. **Por patrón de URL** (`serverUrl`): Coincide con URLs de servidor remoto con soporte de comodín

Importante: Cada entrada debe tener exactamente uno de `serverName` , `serverCommand` o `serverUrl` .

Configuración de ejemplo

```
{
  "allowedMcpServers": [
    // Permitir por nombre de servidor
    { "serverName": "github" },
    { "serverName": "sentry" },

    // Permitir por comando exacto (para servidores stdio)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // Permitir por patrón de URL (para servidores remotos)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // Bloquear por nombre de servidor
    { "serverName": "dangerous-server" },

    // Bloquear por comando exacto (para servidores stdio)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // Bloquear por patrón de URL (para servidores remotos)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

Cómo funcionan las restricciones basadas en comandos

Coincidencia exacta:

- Los arrays de comandos deben coincidir **exactamente** - tanto el comando como todos los argumentos en el orden correcto
- Ejemplo: `["npx", "-y", "server"]` NO coincidirá con `["npx", "server"]` o `["npx", "-y", "server", "--flag"]`

Comportamiento del servidor stdio:

- Cuando la lista de permitidos contiene **cualquier** entrada `serverCommand`, los servidores stdio **deben** coincidir con uno de esos comandos

- Los servidores stdio no pueden pasar solo por nombre cuando hay restricciones de comando presentes
- Esto asegura que los administradores puedan aplicar qué comandos están permitidos ejecutarse

Comportamiento del servidor no-stdio:

- Los servidores remotos (HTTP, SSE, WebSocket) usan coincidencia basada en URL cuando existen entradas `serverUrl` en la lista de permitidos
- Si no existen entradas de URL, los servidores remotos recurren a coincidencia basada en nombre
- Las restricciones de comando no se aplican a servidores remotos

Cómo funcionan las restricciones basadas en URL

Los patrones de URL admiten comodines usando `*` para coincidir con cualquier secuencia de caracteres. Esto es útil para permitir dominios completos o subdominios.

Ejemplos de comodín:

- `https://mcp.company.com/*` - Permitir todas las rutas en un dominio específico
- `https://*.example.com/*` - Permitir cualquier subdominio de example.com
- `http://localhost:*/*` - Permitir cualquier puerto en localhost

Comportamiento del servidor remoto:

- Cuando la lista de permitidos contiene **cualquier** entrada `serverUrl`, los servidores remotos **deben** coincidir con uno de esos patrones de URL
- Los servidores remotos no pueden pasar solo por nombre cuando hay restricciones de URL presentes
- Esto asegura que los administradores puedan aplicar qué puntos finales remotos están permitidos

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

Resultado:

- Servidor HTTP en `https://mcp.company.com/api` :  Permitido (coincide con patrón de URL)
- Servidor HTTP en `https://api.internal.corp/mcp` :  Permitido (coincide con subdominio comodín)
- Servidor HTTP en `https://external.com/mcp` :  Bloqueado (no coincide con ningún patrón de URL)
- Servidor stdio con cualquier comando:  Bloqueado (sin entradas de nombre o comando para coincidir)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Resultado:

- Servidor stdio con `["npx", "-y", "approved-package"]` :  Permitido (coincide con comando)
- Servidor stdio con `["node", "server.js"]` :  Bloqueado (no coincide con comando)
- Servidor HTTP llamado "my-api":  Bloqueado (sin entradas de nombre para coincidir)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Resultado:

- Servidor stdio llamado "local-tool" con `["npx", "-y", "approved-package"]` :  Permitido (coincide con comando)
- Servidor stdio llamado "local-tool" con `["node", "server.js"]` :  Bloqueado (existen entradas de comando pero no coincide)

- Servidor stdio llamado “github” con `["node", "server.js"]`: ❌ Bloqueado (los servidores stdio deben coincidir con comandos cuando existen entradas de comando)
- Servidor HTTP llamado “github”: ✅ Permitido (coincide con nombre)
- Servidor HTTP llamado “other-api”: ❌ Bloqueado (el nombre no coincide)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

Resultado:

- Servidor stdio llamado “github” con cualquier comando: ✅ Permitido (sin restricciones de comando)
- Servidor stdio llamado “internal-tool” con cualquier comando: ✅ Permitido (sin restricciones de comando)
- Servidor HTTP llamado “github”: ✅ Permitido (coincide con nombre)
- Cualquier servidor llamado “other”: ❌ Bloqueado (el nombre no coincide)

Comportamiento de la lista de permitidos (`allowedMcpServers`)

- `undefined` (predeterminado): Sin restricciones - los usuarios pueden configurar cualquier servidor MCP
- Array vacío `[]` : Bloqueo completo - los usuarios no pueden configurar ningún servidor MCP
- Lista de entradas: Los usuarios solo pueden configurar servidores que coincidan por nombre, comando o patrón de URL

Comportamiento de la lista de bloqueados (`deniedMcpServers`)

- `undefined` (predeterminado): Ningún servidor está bloqueado
- Array vacío `[]` : Ningún servidor está bloqueado
- Lista de entradas: Los servidores especificados están explícitamente bloqueados en todos los alcances

Notas importantes

- **La Opción 1 y la Opción 2 se pueden combinar:** Si existe `managed-mcp.json`, tiene control exclusivo y los usuarios no pueden agregar servidores. Las listas de permitidos/bloqueados aún se aplican a los servidores gestionados mismos.
- **La lista de bloqueados tiene precedencia absoluta:** Si un servidor coincide con una entrada de lista de bloqueados (por nombre, comando o URL), será bloqueado incluso si está en la lista de permitidos
- **Las restricciones basadas en nombre, comando y URL funcionan juntas:** un servidor pasa si coincide con **cualquiera** de una entrada de nombre, una entrada de comando o un patrón de URL (a menos que esté bloqueado por lista de bloqueados)

Note:

Cuando se usa `managed-mcp.json`: Los usuarios no pueden agregar servidores MCP a través de `claude mcp add` o archivos de configuración. La configuración `allowedMcpServers` y `deniedMcpServers` aún se aplica para filtrar qué servidores gestionados se cargan realmente.

Ampliar Claude con skills

Cree, gestione y comparta skills para ampliar las capacidades de Claude en Claude Code. Incluye comandos personalizados y skills agrupados.

Los skills amplían lo que Claude puede hacer. Cree un archivo `SKILL.md` con instrucciones, y Claude lo añade a su kit de herramientas. Claude utiliza skills cuando es relevante, o puede invocar uno directamente con `/skill-name`.

Note:

Para comandos integrados como `/help` y `/compact`, consulte [modo interactivo](#).

Los comandos personalizados se han fusionado en skills. Un archivo en `.claude/commands/deploy.md` y un skill en `.claude/skills/deploy/SKILL.md` crean ambos `/deploy` y funcionan de la misma manera. Sus archivos existentes en `.claude/commands/` siguen funcionando. Los skills añaden características opcionales: un directorio para archivos de apoyo, frontmatter para [controlar si usted o Claude los invoca](#), y la capacidad de que Claude los cargue automáticamente cuando sea relevante.

Los skills de Claude Code siguen el estándar abierto [Agent Skills](#), que funciona en múltiples herramientas de IA. Claude Code extiende el estándar con características adicionales como [control de invocación](#), [ejecución de subagent](#), e [inyección de contexto dinámico](#).

Skills agrupados

Los skills agrupados se envían con Claude Code y están disponibles en cada sesión. A diferencia de los [comandos integrados](#), que ejecutan lógica fija directamente, los skills agrupados se basan en prompts: le dan a Claude un manual detallado y le permiten orquestar el trabajo utilizando sus herramientas. Esto significa que los skills agrupados pueden generar agentes paralelos, leer archivos y adaptarse a su base de código.

Invoca los skills agrupados de la misma manera que cualquier otro skill: escribe `/` seguido del nombre del skill.

- `/simplify`: revisa sus archivos modificados recientemente para problemas de reutilización de código, calidad y eficiencia, y luego los corrige. Ejecútelo después de implementar una característica o corrección de errores para limpiar su trabajo. Genera tres agentes de revisión en paralelo (reutilización de código, calidad de código,

eficiencia), agrega sus hallazgos y aplica correcciones. Pase texto opcional para enfocarse en preocupaciones específicas: `/simplify focus on memory efficiency`.

- `/batch <instruction>`: orquesta cambios a gran escala en una base de código en paralelo. Proporcione una descripción del cambio y `/batch` investiga la base de código, descompone el trabajo en 5 a 30 unidades independientes, y presenta un plan para su aprobación. Una vez aprobado, genera un agente de fondo por unidad, cada uno en un [git worktree](#) aislado. Cada agente implementa su unidad, ejecuta pruebas y abre una solicitud de extracción. Requiere un repositorio git. Ejemplo: `/batch migrate src/ from Solid to React`.
- `/debug [description]`: soluciona problemas de su sesión actual de Claude Code leyendo el registro de depuración de la sesión. Opcionalmente describa el problema para enfocar el análisis.
- `/loop [interval] <prompt>`: ejecuta un prompt repetidamente en un intervalo mientras la sesión permanece abierta. Claude analiza el intervalo, programa una tarea cron recurrente y confirma la cadencia. Útil para sondear un despliegue, cuidar una solicitud de extracción o ejecutar periódicamente otro skill. Ejemplo: `/loop 5m check if the deploy finished`. Consulte [Ejecutar prompts en un horario](#).
- `/claude-api`: carga material de referencia de la API de Claude para el idioma de su proyecto (Python, TypeScript, Java, Go, Ruby, C#, PHP o cURL) y referencia de Agent SDK para Python y TypeScript. Cubre uso de herramientas, streaming, lotes, salidas estructuradas y errores comunes. También se activa automáticamente cuando su código importa `anthropic`, `@anthropic-ai/sdk` o `claude_agent_sdk`.

Primeros pasos

Cree su primer skill

Este ejemplo crea un skill que enseña a Claude a explicar código usando diagramas visuales y analogías. Dado que utiliza frontmatter predeterminado, Claude puede cargarlo automáticamente cuando pregunta cómo funciona algo, o puede invocarlo directamente con `/explain-code`.

Step 1: Cree el directorio del skill

Cree un directorio para el skill en su carpeta de skills personales. Los skills personales están disponibles en todos sus proyectos.

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: Escriba SKILL.md

Cada skill necesita un archivo `SKILL.md` con dos partes: frontmatter YAML (entre marcadores `---`) que le dice a Claude cuándo usar el skill, y contenido markdown con instrucciones que Claude sigue cuando se invoca el skill. El campo `name` se convierte en el `/slash-command`, y la `description` ayuda a Claude a decidir cuándo cargarlo automáticamente.

Cree `~/.claude/skills/explain-code/SKILL.md`:

```
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining
how code works, teaching about a codebase, or when the user asks "how does this
work?"
---
```

When explaining code, always include:

- **Start with an analogy****: Compare the code to something from everyday life
- **Draw a diagram****: Use ASCII art to show the flow, structure, or relationships
- **Walk through the code****: Explain step-by-step what happens
- **Highlight a gotcha****: What's a common mistake or misconception?

Keep explanations conversational. For complex concepts, use multiple analogies.

Step 3: Pruebe el skill

Puede probarlo de dos maneras:

Deje que Claude lo invoque automáticamente preguntando algo que coincida con la descripción:

```
How does this code work?
```

O invóquelo directamente con el nombre del skill:

```
/explain-code src/auth/login.ts
```

De cualquier manera, Claude debe incluir una analogía y un diagrama ASCII en su explicación.

Dónde viven los skills

Dónde almacena un skill determina quién puede usarlo:

Ubicación	Ruta	Se aplica a
Empresa	Consulte configuración administrada	Todos los usuarios de su organización
Personal	<code>~/.claude/skills/<skill-name>/SKILL.md</code>	Todos sus proyectos
Proyecto	<code>.claude/skills/<skill-name>/SKILL.md</code>	Solo este proyecto
Plugin	<code><plugin>/skills/<skill-name>/SKILL.md</code>	Donde el plugin está habilitado

Cuando los skills comparten el mismo nombre en diferentes niveles, las ubicaciones de mayor prioridad ganan: empresa > personal > proyecto. Los skills de plugin utilizan un espacio de nombres `plugin-name:skill-name`, por lo que no pueden entrar en conflicto con otros niveles. Si tiene archivos en `.claude/commands/`, funcionan de la misma manera, pero si un skill y un comando comparten el mismo nombre, el skill tiene prioridad.

Descubrimiento automático desde directorios anidados

Cuando trabaja con archivos en subdirectorios, Claude Code descubre automáticamente skills de directorios `.claude/skills/` anidados. Por ejemplo, si está editando un archivo en `packages/frontend/`, Claude Code también busca skills en `packages/frontend/.claude/skills/`. Esto admite configuraciones de monorepo donde los paquetes tienen sus propios skills.

Cada skill es un directorio con `SKILL.md` como punto de entrada:

```
my-skill/
├─ SKILL.md           # Main instructions (required)
├─ template.md        # Template for Claude to fill in
├─ examples/
│   └─ sample.md      # Example output showing expected format
└─ scripts/
    └─ validate.sh     # Script Claude can execute
```

El `SKILL.md` contiene las instrucciones principales y es obligatorio. Otros archivos son opcionales y le permiten crear skills más potentes: plantillas para que Claude las complete, salidas de ejemplo que muestren el formato esperado, scripts que Claude puede ejecutar o documentación de referencia detallada. Haga referencia a estos archivos desde su `SKILL.md` para que Claude sepa qué contienen y cuándo cargarlos. Consulte [Añadir archivos de apoyo](#) para más detalles.

Note:

Los archivos en `.claude/commands/` siguen funcionando y admiten el mismo [frontmatter](#). Los skills se recomiendan ya que admiten características adicionales como archivos de apoyo.

Skills de directorios adicionales

Los skills definidos en `.claude/skills/` dentro de directorios añadidos a través de `--add-dir` se cargan automáticamente y se detectan mediante detección de cambios en vivo, por lo que puede editarlos durante una sesión sin reiniciar.

Note:

Los archivos `CLAUDE.md` de directorios `--add-dir` no se cargan de forma predeterminada. Para cargarlos, establezca `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`. Consulte [Cargar desde directorios adicionales](#).

Configurar skills

Los skills se configuran a través de frontmatter YAML en la parte superior de `SKILL.md` y el contenido markdown que sigue.

Tipos de contenido de skill

Los archivos de skill pueden contener cualquier instrucción, pero pensar en cómo desea invocarlos ayuda a guiar qué incluir:

Contenido de referencia añade conocimiento que Claude aplica a su trabajo actual. Convenciones, patrones, guías de estilo, conocimiento del dominio. Este contenido se ejecuta en línea para que Claude pueda usarlo junto con el contexto de su conversación.

```

---
name: api-conventions
description: API design patterns for this codebase
---

When writing API endpoints:
- Use RESTful naming conventions
- Return consistent error formats
- Include request validation

```

Contenido de tarea le da a Claude instrucciones paso a paso para una acción específica, como despliegues, commits o generación de código. Estas son a menudo acciones que desea invocar directamente con `/skill-name` en lugar de dejar que Claude decida cuándo ejecutarlas. Añade `disable-model-invocation: true` para evitar que Claude la dispare automáticamente.

```

---
name: deploy
description: Deploy the application to production
context: fork
disable-model-invocation: true
---

Deploy the application:
1. Run the test suite
2. Build the application
3. Push to the deployment target

```

Su `SKILL.md` puede contener cualquier cosa, pero pensar en cómo desea que se invoque el skill (por usted, por Claude, o ambos) y dónde desea que se ejecute (en línea o en un subagente) ayuda a guiar qué incluir. Para skills complejos, también puede [añadir archivos de apoyo](#) para mantener el skill principal enfocado.

Referencia de frontmatter

Más allá del contenido markdown, puede configurar el comportamiento del skill utilizando campos de frontmatter YAML entre marcadores `---` en la parte superior de su archivo `SKILL.md`:

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

Todos los campos son opcionales. Solo se recomienda `description` para que Claude sepa cuándo usar el skill.

Campo	Requerido	Descripción
<code>name</code>	No	Nombre para mostrar del skill. Si se omite, utiliza el nombre del directorio. Solo letras minúsculas, números y guiones (máximo 64 caracteres).
<code>description</code>	Recomendado	Qué hace el skill y cuándo usarlo. Claude utiliza esto para decidir cuándo aplicar el skill. Si se omite, utiliza el primer párrafo del contenido markdown.
<code>argument-hint</code>	No	Sugerencia mostrada durante el autocompletado para indicar argumentos esperados. Ejemplo: <code>[issue-number]</code> o <code>[filename] [format]</code> .
<code>disable-model-invocation</code>	No	Establezca en <code>true</code> para evitar que Claude cargue automáticamente este skill. Utilice para flujos de trabajo que desea activar manualmente con <code>/name</code> . Predeterminado: <code>false</code> .

Campo	Requerido	Descripción
<code>user-invocable</code>	No	Establezca en <code>false</code> para ocultar del menú <code>/</code> . Utilícelo para conocimiento de fondo que los usuarios no deberían invocar directamente. Predeterminado: <code>true</code> .
<code>allowed-tools</code>	No	Herramientas que Claude puede usar sin pedir permiso cuando este skill está activo.
<code>model</code>	No	Modelo a usar cuando este skill está activo.
<code>context</code>	No	Establezca en <code>fork</code> para ejecutar en un contexto de subagent bifurcado.
<code>agent</code>	No	Qué tipo de subagent usar cuando <code>context: fork</code> está establecido.
<code>hooks</code>	No	Hooks limitados al ciclo de vida de este skill. Consulte Hooks en skills y agentes para el formato de configuración.

Sustituciones de cadena disponibles

Los skills admiten sustitución de cadena para valores dinámicos en el contenido del skill:

Variable	Descripción
<code>\$ARGUMENTS</code>	Todos los argumentos pasados al invocar el skill. Si <code>\$ARGUMENTS</code> no está presente en el contenido, los argumentos se añaden como <code>ARGUMENTS: <value></code> .
<code>\$ARGUMENTS[N]</code>	Acceda a un argumento específico por índice basado en 0, como <code>\$ARGUMENTS[0]</code> para el primer argumento.

Variable	Descripción
<code>\$N</code>	Abreviatura para <code>\$ARGUMENTS[N]</code> , como <code>\$0</code> para el primer argumento o <code>\$1</code> para el segundo.
<code>\$ {CLAUDE_SESSION_ID }</code>	El ID de sesión actual. Útil para registro, creación de archivos específicos de sesión o correlación de salida de skill con sesiones.
<code>\$ {CLAUDE_SKILL_DIR}</code>	El directorio que contiene el archivo <code>SKILL.md</code> del skill. Para skills de plugin, este es el subdirectorío del skill dentro del plugin, no la raíz del plugin. Utilice esto en comandos de inyección bash para hacer referencia a scripts o archivos incluidos con el skill, independientemente del directorio de trabajo actual.

Ejemplo usando sustituciones:

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

Añadir archivos de apoyo

Los skills pueden incluir múltiples archivos en su directorio. Esto mantiene `SKILL.md` enfocado en lo esencial mientras permite que Claude acceda a material de referencia detallado solo cuando sea necesario. Documentos de referencia grandes, especificaciones de API o colecciones de ejemplos no necesitan cargarse en contexto cada vez que se ejecuta el skill.

```
my-skill/
├─ SKILL.md (required - overview and navigation)
├─ reference.md (detailed API docs - loaded when needed)
├─ examples.md (usage examples - loaded when needed)
└─ scripts/
    └─ helper.py (utility script - executed, not loaded)
```

Haga referencia a archivos de apoyo desde `SKILL.md` para que Claude sepa qué contiene cada archivo y cuándo cargarlo:

```
### Additional resources
```

- For complete API details, see `[reference.md](reference.md)`
- For usage examples, see `[examples.md](examples.md)`

Tip:

Mantenga `SKILL.md` por debajo de 500 líneas. Mueva material de referencia detallado a archivos separados.

Controlar quién invoca un skill

De forma predeterminada, tanto usted como Claude pueden invocar cualquier skill. Puede escribir `/skill-name` para invocarlo directamente, y Claude puede cargarlo automáticamente cuando sea relevante para su conversación. Dos campos de frontmatter le permiten restringir esto:

- `disable-model-invocation: true` : Solo usted puede invocar el skill. Utilice esto para flujos de trabajo con efectos secundarios o que desea controlar el tiempo, como `/commit`, `/deploy` o `/send-slack-message`. No desea que Claude decida desplegar porque su código se ve listo.
- `user-invocable: false` : Solo Claude puede invocar el skill. Utilice esto para conocimiento de fondo que no es accionable como comando. Un skill `legacy-system-context` explica cómo funciona un sistema antiguo. Claude debería saber esto cuando sea relevante, pero `/legacy-system-context` no es una acción significativa para que los usuarios realicen.

Este ejemplo crea un skill de despliegue que solo usted puede activar. El campo `disable-model-invocation: true` evita que Claude lo ejecute automáticamente:

```
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---
```

Deploy \$ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deployment target
4. Verify the deployment succeeded

Aquí se muestra cómo los dos campos afectan la invocación y la carga de contexto:

Frontmatter	Puede invocar	Claude puede invocar	Cuándo se carga en contexto
(predeterminado)	Sí	Sí	La descripción siempre en contexto, el skill completo se carga cuando se invoca
disable-model-invocation: true	Sí	No	La descripción no está en contexto, el skill completo se carga cuando lo invoca
user-invokable: false	No	Sí	La descripción siempre en contexto, el skill completo se carga cuando se invoca

Note:

En una sesión regular, las descripciones de skills se cargan en contexto para que Claude sepa qué está disponible, pero el contenido completo del skill solo se carga cuando se invoca. Los [subagents con skills precargados](#) funcionan de manera diferente: el contenido completo del skill se inyecta al inicio.

Restringir acceso a herramientas

Utilice el campo `allowed-tools` para limitar qué herramientas puede usar Claude cuando un skill está activo. Este skill crea un modo de solo lectura donde Claude puede explorar archivos pero no modificarlos:

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

Pasar argumentos a skills

Tanto usted como Claude pueden pasar argumentos al invocar un skill. Los argumentos están disponibles a través del marcador de posición `$ARGUMENTS`.

Este skill corrige un problema de GitHub por número. El marcador de posición `$ARGUMENTS` se reemplaza con lo que sigue al nombre del skill:

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---
```

Fix GitHub issue \$ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write tests
5. Create a commit

Cuando ejecuta `/fix-issue 123`, Claude recibe “Fix GitHub issue 123 following our coding standards...”

Si invoca un skill con argumentos pero el skill no incluye `$ARGUMENTS`, Claude Code añade `ARGUMENTS: <your input>` al final del contenido del skill para que Claude siga viendo lo que escribió.

Para acceder a argumentos individuales por posición, utilice `$ARGUMENTS[N]` o la forma más corta `$N` :

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

Ejecutar `/migrate-component searchBar React Vue` reemplaza `$ARGUMENTS[0]` con `searchBar`, `$ARGUMENTS[1]` con `React` y `$ARGUMENTS[2]` con `Vue`. El mismo skill usando la abreviatura `$N` :

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

Patrones avanzados

Injectar contexto dinámico

La sintaxis `! command``` ejecuta comandos de shell antes de que el contenido del skill se envíe a Claude. La salida del comando reemplaza el marcador de posición, por lo que Claude recibe datos reales, no el comando en sí.

Este skill resume una solicitud de extracción obteniendo datos de PR en vivo con la CLI de GitHub. Los comandos `! gh pr diff``` y otros se ejecutan primero, y su salida se inserta en el prompt:

```

---
name: pr-summary
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
---

### Pull request context
- PR diff: !`gh pr diff`
- PR comments: !`gh pr view --comments`
- Changed files: !`gh pr diff --name-only`

### Your task
Summarize this pull request...

```

Cuando se ejecuta este skill:

1. Cada `! command`` se ejecuta inmediatamente (antes de que Claude vea algo)
2. La salida reemplaza el marcador de posición en el contenido del skill
3. Claude recibe el prompt completamente renderizado con datos de PR reales

Esto es preprocesamiento, no algo que Claude ejecute. Claude solo ve el resultado final.

Tip:

Para habilitar [pensamiento extendido](#) en un skill, incluya la palabra “ultrathink” en cualquier lugar en el contenido de su skill.

Ejecutar skills en un subagent

Añada `context: fork` a su frontmatter cuando desee que un skill se ejecute en aislamiento. El contenido del skill se convierte en el prompt que impulsa el subagent. No tendrá acceso a su historial de conversación.

Warning:

context: fork solo tiene sentido para skills con instrucciones explícitas. Si su skill contiene directrices como “use estas convenciones de API” sin una tarea, el subagent recibe las directrices pero sin un prompt accionable, y regresa sin salida significativa.

Los skills y los [subagents](#) funcionan juntos en dos direcciones:

Enfoque	Prompt del sistema	Tarea	También carga
Skill con context: fork	Del tipo de agente (Explore , Plan , etc.)	Contenido de SKILL.md	CLAUDE.md
Subagent con campo skills	Cuerpo markdown del subagent	Mensaje de delegación de Claude	Skills precargados + CLAUDE.md

Con **context: fork** , escribe la tarea en tu skill y elige un tipo de agente para ejecutarla. Para lo inverso (definir un subagent personalizado que use skills como material de referencia), consulte [Subagents](#).

Ejemplo: Skill de investigación usando agente Explore

Este skill ejecuta investigación en un agente Explore bifurcado. El contenido del skill se convierte en la tarea, y el agente proporciona herramientas de solo lectura optimizadas para exploración de base de código:

```
---
name: deep-research
description: Research a topic thoroughly
context: fork
agent: Explore
---

Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyze the code
3. Summarize findings with specific file references
```

Cuando se ejecuta este skill:

1. Se crea un nuevo contexto aislado

2. El subagent recibe el contenido del skill como su prompt (“Research \$ARGUMENTS thoroughly...”)
3. El campo `agent` determina el entorno de ejecución (modelo, herramientas y permisos)
4. Los resultados se resumen y se devuelven a su conversación principal

El campo `agent` especifica qué configuración de subagent usar. Las opciones incluyen agentes integrados (`Explore` , `Plan` , `general-purpose`) o cualquier subagent personalizada de `.claude/agents/` . Si se omite, utiliza `general-purpose` .

Restringir el acceso de Claude a skills

De forma predeterminada, Claude puede invocar cualquier skill que no tenga `disable-model-invocation: true` establecido. Los skills que definen `allowed-tools` otorgan a Claude acceso a esas herramientas sin aprobación por uso cuando el skill está activo. Su [configuración de permisos](#) sigue rigiendo el comportamiento de aprobación de línea base para todas las demás herramientas. Los comandos integrados como `/compact` e `/init` no están disponibles a través de la herramienta Skill.

Tres formas de controlar qué skills puede invocar Claude:

Deshabilitar todos los skills negando la herramienta Skill en `/permissions` :

```
## Add to deny rules:
Skill
```

Permitir o denegar skills específicos usando [reglas de permisos](#):

```
## Allow only specific skills
Skill(commit)
Skill(review-pr *)

## Deny specific skills
Skill(deploy *)
```

Sintaxis de permisos: `Skill(name)` para coincidencia exacta, `Skill(name *)` para coincidencia de prefijo con cualquier argumento.

Ocultar skills individuales añadiendo `disable-model-invocation: true` a su frontmatter. Esto elimina el skill del contexto de Claude por completo.

Note:

El campo `user-invocable` solo controla la visibilidad del menú, no el acceso a la herramienta Skill. Utilice `disable-model-invocation: true` para bloquear la invocación programática.

Compartir skills

Los skills se pueden distribuir en diferentes ámbitos dependiendo de su audiencia:

- **Skills de proyecto:** Confirme `.claude/skills/` en control de versiones
- **Plugins:** Cree un directorio `skills/` en su [plugin](#)
- **Administrado:** Implemente en toda la organización a través de [configuración administrativa](#)

Generar salida visual

Los skills pueden agrupar y ejecutar scripts en cualquier idioma, dándole a Claude capacidades más allá de lo que es posible en un solo prompt. Un patrón poderoso es generar salida visual: archivos HTML interactivos que se abren en su navegador para explorar datos, depurar o crear informes.

Este ejemplo crea un explorador de base de código: una vista de árbol interactiva donde puede expandir y contraer directorios, ver tamaños de archivo de un vistazo e identificar tipos de archivo por color.

Cree el directorio Skill:

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

Cree `~/.claude/skills/codebase-visualizer/SKILL.md`. La descripción le dice a Claude cuándo activar este Skill, y las instrucciones le dicen a Claude que ejecute el script incluido:

```

---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your
codebase. Use when exploring a new repo, understanding project structure, or
identifying large files.
allowed-tools: Bash(python *)
---

## Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure
with collapsible directories.

### Usage

Run the visualization script from your project root:

```bash
python ~/.claude/skills/codebase-visualizer/scripts/visualize.py .
```text

This creates `codebase-map.html` in the current directory and opens it in your
default browser.

### What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder

```

Cree `~/.claude/skills/codebase-visualizer/scripts/visualize.py`. Este script escanea un árbol de directorios y genera un archivo HTML independiente con:

- Una **barra lateral de resumen** que muestra el recuento de archivos, recuento de directorios, tamaño total y número de tipos de archivo
- Un **gráfico de barras** que desglosa la base de código por tipo de archivo (los 8 principales por tamaño)

- Un **árbol contraíble** donde puede expandir y contraer directorios, con indicadores de tipo de archivo codificados por color

El script requiere Python pero utiliza solo bibliotecas integradas, por lo que no hay paquetes para instalar:

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

from pathlib import Path
from collections import Counter

IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) → dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size,
"ext": ext})

                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
        except PermissionError:
            pass
    return result

def generate_html(data: dict, stats: dict, output: Path) → None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[0:8]
    colors = {
```

```

        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e
    ',

        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }

    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{color
s.get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )

    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
  <meta charset="utf-8"><title>Codebase Explorer</title>
  <style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e;
color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right:
1px solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform:
uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0;
border-bottom: 1px solid #3d3d5c; }}
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}

```

```

    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius:
4px; }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
</style>
</head><body>
<div class="container">
  <div class="sidebar">
    <h1><img alt="Summary icon" data-bbox="211 356 228 371"/> Summary</h1>
    <div class="stat"><span>Files</span><span class="stat-
value">{stats["files"]:,}</span></div>
    <div class="stat"><span>Directories</span><span class="stat-value">{stats["d
irs"]:,}</span></div>
    <div class="stat"><span>Total size</span><span class="stat-
value">{fmt(data["size"])}</span></div>
    <div class="stat"><span>File types</span><span class="stat-value">{len(stats
["extensions"])}</span></div>
    <h2>By file type</h2>
    {lang_bars}
  </div>
  <div class="main">
    <h1><img alt="Folder icon" data-bbox="211 636 228 651"/> {data["name"]}</h1>
    <ul class="tree" id="root"></ul>
  </div>
</div>
<script>
  const data = {json.dumps(data)};
  const colors = {json.dumps(colors)};
  function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/
1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
  function render(node, parent) {{
    if (node.children) {{
      const det = document.createElement('details');
      det.open = parent ≡ document.getElementById('root');

```

```

    det.innerHTML = `<summary><span class="folder">📁 ${node.name}</
span><span class="size">${fmt(node.size)}</span></summary>`;
    const ul = document.createElement('ul'); ul.className = 'tree';
    node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) ||
a.name.localeCompare(b.name));
    node.children.forEach(c => render(c, ul));
    det.appendChild(ul);
    const li = document.createElement('li'); li.appendChild(det);
parent.appendChild(li);
  } else {
    const li = document.createElement('li'); li.className = 'file';
    li.innerHTML = `<span class="dot" style="background:${colors[node.ext]} '
#6b7280'`></span>${node.name}<span class="size">${fmt(node.size)}</span>`;
    parent.appendChild(li);
  }
}
data.children.forEach(c => render(c, document.getElementById('root')));
</script>
</body></html>''
output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes":
Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

Para probar, abra Claude Code en cualquier proyecto y pregunte “Visualize this codebase.” Claude ejecuta el script, genera `codebase-map.html` y lo abre en su navegador.

Este patrón funciona para cualquier salida visual: gráficos de dependencias, informes de cobertura de pruebas, documentación de API o visualizaciones de esquema de base de datos. El script incluido hace el trabajo pesado mientras Claude maneja la orquestación.

Solución de problemas

Skill no se activa

Si Claude no usa su skill cuando se espera:

1. Verifique que la descripción incluya palabras clave que los usuarios dirían naturalmente
2. Verifique que el skill aparezca en `What skills are available?`
3. Intente reformular su solicitud para que coincida más estrechamente con la descripción
4. Invóquelo directamente con `/skill-name` si el skill es invocable por el usuario

Skill se activa demasiado a menudo

Si Claude usa su skill cuando no desea:

1. Haga la descripción más específica
2. Añada `disable-model-invocation: true` si solo desea invocación manual

Claude no ve todos mis skills

Las descripciones de skills se cargan en contexto para que Claude sepa qué está disponible. Si tiene muchos skills, pueden exceder el presupuesto de caracteres. El presupuesto se escala dinámicamente al 2% de la ventana de contexto, con un respaldo de 16,000 caracteres. Ejecute `/context` para verificar una advertencia sobre skills excluidos.

Para anular el límite, establezca la variable de entorno `SLASH_COMMAND_TOOL_CHAR_BUDGET`.

Recursos relacionados

- [Subagents](#): delegue tareas a agentes especializados
- [Plugins](#): empaque y distribuya skills con otras extensiones
- [Hooks](#): automatice flujos de trabajo alrededor de eventos de herramientas
- [Memory](#): gestione archivos CLAUDE.md para contexto persistente
- [Modo interactivo](#): comandos integrados y atajos
- [Permisos](#): controle el acceso a herramientas y skills

Crear plugins

Crea plugins personalizados para extender Claude Code con skills, agentes, hooks y servidores MCP.

Los plugins te permiten extender Claude Code con funcionalidad personalizada que se puede compartir entre proyectos y equipos. Esta guía cubre la creación de tus propios plugins con skills, agentes, hooks y servidores MCP.

¿Buscas instalar plugins existentes? Consulta [Descubrir e instalar plugins](#). Para especificaciones técnicas completas, consulta [Referencia de plugins](#).

Cuándo usar plugins versus configuración independiente

Claude Code admite dos formas de agregar skills, agentes y hooks personalizados:

Enfoque	Nombres de skills	Mejor para
Independiente (directorio <code>.claude/</code>)	<code>/hello</code>	Flujos de trabajo personales, personalizaciones específicas del proyecto, experimentos rápidos
Plugins (directorios con <code>.claude-plugin/plugin.json</code>)	<code>/plugin-name:hello</code>	Compartir con compañeros de equipo, distribuir a la comunidad, lanzamientos versionados, reutilizable en múltiples proyectos

Usa configuración independiente cuando:

- Estés personalizando Claude Code para un único proyecto
- La configuración es personal y no necesita ser compartida
- Estés experimentando con skills o hooks antes de empaquetarlos
- Quieras nombres de skills cortos como `/hello` o `/deploy`

Usa plugins cuando:

- Quieras compartir funcionalidad con tu equipo o comunidad
- Necesites los mismos skills/agentes en múltiples proyectos

- Quieras control de versiones y actualizaciones fáciles para tus extensiones
- Estés distribuyendo a través de un marketplace
- Estés de acuerdo con skills con espacios de nombres como `/my-plugin:hello` (los espacios de nombres previenen conflictos entre plugins)

Tip:

Comienza con configuración independiente en `.claude/` para iteración rápida, luego [convierte a un plugin](#) cuando estés listo para compartir.

Inicio rápido

Este inicio rápido te guía a través de la creación de un plugin con un skill personalizado. Crearás un manifiesto (el archivo de configuración que define tu plugin), agregarás un skill y lo probarás localmente usando la bandera `--plugin-dir`.

Requisitos previos

- Claude Code [instalado y autenticado](#)
- Claude Code versión 1.0.33 o posterior (ejecuta `claude --version` para verificar)

Note:

Si no ves el comando `/plugin`, actualiza Claude Code a la última versión. Consulta [Troubleshooting](#) para obtener instrucciones de actualización.

Crea tu primer plugin

Step 1: Crea el directorio del plugin

Cada plugin vive en su propio directorio que contiene un manifiesto y tus skills, agentes o hooks. Crea uno ahora:

```
mkdir my-first-plugin
```

Step 2: Crea el manifiesto del plugin

El archivo de manifiesto en `.claude-plugin/plugin.json` define la identidad de tu plugin: su nombre, descripción y versión. Claude Code usa estos metadatos para mostrar tu plugin en el administrador de plugins.

Crea el directorio `.claude-plugin` dentro de tu carpeta de plugin:

```
mkdir my-first-plugin/.claude-plugin
```

Luego crea `my-first-plugin/.claude-plugin/plugin.json` con este contenido:

```
{
  "name": "my-first-plugin",
  "description": "A greeting plugin to learn the basics",
  "version": "1.0.0",
  "author": {
    "name": "Your Name"
  }
}
```

Campo	Propósito
<code>name</code>	Identificador único y espacio de nombres de skill. Los skills tienen este prefijo (por ejemplo, <code>/my-first-plugin:hello</code>).
<code>description</code>	Se muestra en el administrador de plugins al examinar o instalar plugins.
<code>version</code>	Rastrear lanzamientos usando versionado semántico .
<code>author</code>	Opcional. Útil para atribución.

Para campos adicionales como `homepage`, `repository` y `license`, consulta el [esquema de manifiesto completo](#).

Step 3: Agrega un skill

Los skills viven en el directorio `skills/`. Cada skill es una carpeta que contiene un archivo `SKILL.md`. El nombre de la carpeta se convierte en el nombre del skill, con el prefijo del espacio de nombres del plugin (`hello/` en un plugin llamado `my-first-plugin` crea `/my-first-plugin:hello`).

Crea un directorio de skill en tu carpeta de plugin:

```
mkdir -p my-first-plugin/skills/hello
```

Luego crea `my-first-plugin/skills/hello/SKILL.md` con este contenido:

```

---
description: Greet the user with a friendly message
disable-model-invocation: true
---

Greet the user warmly and ask how you can help them today.

```

Step 4: Prueba tu plugin

Ejecuta Claude Code con la bandera `--plugin-dir` para cargar tu plugin:

```
claude --plugin-dir ./my-first-plugin
```

Una vez que Claude Code se inicie, prueba tu nuevo skill:

```
/my-first-plugin:hello
```

Verás que Claude responde con un saludo. Ejecuta `/help` para ver tu skill listado bajo el espacio de nombres del plugin.

Note:

¿Por qué espacios de nombres? Los skills de plugin siempre tienen espacios de nombres (como `/greet:hello`) para prevenir conflictos cuando múltiples plugins tienen skills con el mismo nombre.

Para cambiar el prefijo del espacio de nombres, actualiza el campo `name` en `plugin.json`.

Step 5: Agrega argumentos de skill

Haz tu skill dinámico aceptando entrada del usuario. El marcador de posición `$ARGUMENTS` captura cualquier texto que el usuario proporcione después del nombre del skill.

Actualiza tu archivo `SKILL.md`:

```
---
description: Greet the user with a personalized message
---
```

```
## Hello Skill
```

```
Greet the user named "$ARGUMENTS" warmly and ask how you can help them today. Make
the greeting personal and encouraging.
```

Ejecuta `/reload-plugins` para recoger los cambios, luego prueba el skill con tu nombre:

```
/my-first-plugin:hello Alex
```

Claude te saludará por tu nombre. Para más información sobre pasar argumentos a skills, consulta [Skills](#).

Has creado y probado exitosamente un plugin con estos componentes clave:

- **Manifiesto del plugin** (`.claude-plugin/plugin.json`): describe los metadatos de tu plugin
- **Directorio de skills** (`skills/`): contiene tus skills personalizados
- **Argumentos de skill** (`$ARGUMENTS`): captura entrada del usuario para comportamiento dinámico

Tip:

La bandera `--plugin-dir` es útil para desarrollo y pruebas. Cuando estés listo para compartir tu plugin con otros, consulta [Crear y distribuir un marketplace de plugins](#).

Descripción general de la estructura del plugin

Has creado un plugin con un skill, pero los plugins pueden incluir mucho más: agentes personalizados, hooks, servidores MCP y servidores LSP.

Warning:

Error común: No pongas `commands/`, `agents/`, `skills/` o `hooks/` dentro del directorio `.claude-plugin/`. Solo `plugin.json` va dentro de `.claude-plugin/`. Todos los otros directorios deben estar en el nivel raíz del plugin.

Directorio	Ubicación	Propósito
<code>.claude-plugin/</code>	Raíz del plugin	Contiene el manifiesto <code>plugin.json</code> (opcional si los componentes usan ubicaciones predeterminadas)
<code>commands/</code>	Raíz del plugin	Skills como archivos Mark-down
<code>agents/</code>	Raíz del plugin	Definiciones de agentes personalizados
<code>skills/</code>	Raíz del plugin	Agent Skills con archivos <code>SKILL.md</code>
<code>hooks/</code>	Raíz del plugin	Manejadores de eventos en <code>hooks.json</code>
<code>.mcp.json</code>	Raíz del plugin	Configuraciones de servidor MCP
<code>.lsp.json</code>	Raíz del plugin	Configuraciones de servidor LSP para inteligencia de código
<code>settings.json</code>	Raíz del plugin	Configuraciones predeterminadas aplicadas cuando el plugin está habilitado

Note:

Próximos pasos: ¿Listo para agregar más características? Salta a [Desarrollar plugins más complejos](#) para agregar agentes, hooks, servidores MCP y servidores LSP. Para especificaciones técnicas completas de todos los componentes del plugin, consulta [Referencia de plugins](#).

Desarrollar plugins más complejos

Una vez que te sientas cómodo con plugins básicos, puedes crear extensiones más sofisticadas.

Agrega Skills a tu plugin

Los plugins pueden incluir [Agent Skills](#) para extender las capacidades de Claude. Los skills son invocados por el modelo: Claude los usa automáticamente basándose en el contexto de la tarea.

Agrega un directorio `skills/` en la raíz de tu plugin con carpetas de Skill que contengan archivos `SKILL.md`:

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

Cada `SKILL.md` necesita frontmatter con campos `name` y `description`, seguido de instrucciones:

```
---
name: code-review
description: Reviews code for best practices and potential issues. Use when
reviewing code, checking PRs, or analyzing code quality.
---

When reviewing code, check for:
1. Code organization and structure
2. Error handling
3. Security concerns
4. Test coverage
```

Después de instalar el plugin, ejecuta `/reload-plugins` para cargar los Skills. Para orientación completa sobre la autoría de Skills incluyendo divulgación progresiva y restricciones de herramientas, consulta [Agent Skills](#).

Agrega servidores LSP a tu plugin

Tip:

Para lenguajes comunes como TypeScript, Python y Rust, instala los plugins LSP precompilados desde el marketplace oficial. Crea plugins LSP personalizados solo cuando necesites soporte para lenguajes que aún no están cubiertos.

Los plugins LSP (Language Server Protocol) dan a Claude inteligencia de código en tiempo real. Si necesitas soportar un lenguaje que no tiene un plugin LSP oficial, puedes crear uno propio agregando un archivo `.lsp.json` a tu plugin:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Los usuarios que instalen tu plugin deben tener el binario del servidor de lenguaje instalado en su máquina.

Para opciones de configuración LSP completas, consulta [Servidores LSP](#).

Envía configuraciones predeterminadas con tu plugin

Los plugins pueden incluir un archivo `settings.json` en la raíz del plugin para aplicar configuración predeterminada cuando el plugin está habilitado. Actualmente, solo se admite la clave `agent`.

Establecer `agent` activa uno de los [agentes personalizados](#) del plugin como el hilo principal, aplicando su indicación del sistema, restricciones de herramientas y modelo. Esto permite que un plugin cambie cómo se comporta Claude Code por defecto cuando está habilitado.


```
{
  "agent": "security-reviewer"
}
```

Este ejemplo activa el agente `security-reviewer` definido en el directorio `agents/` del plugin. Las configuraciones de `settings.json` tienen prioridad sobre `settings` declarados en `plugin.json`. Las claves desconocidas se ignoran silenciosamente.

Organiza plugins complejos

Para plugins con muchos componentes, organiza tu estructura de directorios por funcionalidad. Para diseños de directorios completos y patrones de organización, consulta [Estructura de directorios del plugin](#).

Prueba tus plugins localmente

Usa la bandera `--plugin-dir` para probar plugins durante el desarrollo. Esto carga tu plugin directamente sin requerir instalación.

```
claude --plugin-dir ./my-plugin
```

A medida que hagas cambios en tu plugin, ejecuta `/reload-plugins` para recoger las actualizaciones sin reiniciar. Los cambios en la configuración del servidor LSP aún requieren un reinicio completo. Prueba los componentes de tu plugin:

- Prueba tus skills con `/plugin-name:skill-name`
- Verifica que los agentes aparezcan en `/agents`
- Verifica que los hooks funcionen como se espera

Tip:

Puedes cargar múltiples plugins a la vez especificando la bandera varias veces:

```
claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
```

Depura problemas del plugin

Si tu plugin no funciona como se espera:

1. **Verifica la estructura:** Asegúrate de que tus directorios estén en la raíz del plugin, no dentro de `.claude-plugin/`
2. **Prueba componentes individualmente:** Verifica cada comando, agente y hook por separado
3. **Usa herramientas de validación y depuración:** Consulta [Herramientas de depuración y desarrollo](#) para comandos CLI y técnicas de solución de problemas

Comparte tus plugins

Cuando tu plugin esté listo para compartir:

1. **Agrega documentación:** Incluye un `README.md` con instrucciones de instalación y uso
2. **Versiona tu plugin:** Usa [versionado semántico](#) en tu `plugin.json`
3. **Crea o usa un marketplace:** Distribuye a través de [marketplaces de plugins](#) para instalación
4. **Prueba con otros:** Haz que los miembros del equipo prueben el plugin antes de una distribución más amplia

Una vez que tu plugin esté en un marketplace, otros pueden instalarlo usando las instrucciones en [Descubrir e instalar plugins](#).

Envía tu plugin al marketplace oficial

Para enviar un plugin al marketplace oficial de Anthropic, usa uno de los formularios de envío en la aplicación:

- **Claude.ai:** claude.ai/settings/plugins/submit
- **Console:** platform.claude.com/plugins/submit

Note:

Para especificaciones técnicas completas, técnicas de depuración y estrategias de distribución, consulta [Referencia de plugins](#).

Convierte configuraciones existentes en plugins

Si ya tienes skills o hooks en tu directorio `.claude/`, puedes convertirlos en un plugin para compartir y distribución más fácil.

Pasos de migración

Step 1: Crea la estructura del plugin

Crea un nuevo directorio de plugin:

```
mkdir -p my-plugin/.claude-plugin
```

Crea el archivo de manifiesto en `my-plugin/.claude-plugin/plugin.json` :

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: Copia tus archivos existentes

Copia tus configuraciones existentes al directorio del plugin:

```
## Copy commands
cp -r .claude/commands my-plugin/

## Copy agents (if any)
cp -r .claude/agents my-plugin/

## Copy skills (if any)
cp -r .claude/skills my-plugin/
```

Step 3: Migra hooks

Si tienes hooks en tu configuración, crea un directorio de hooks:

```
mkdir my-plugin/hooks
```

Crea `my-plugin/hooks/hooks.json` con tu configuración de hooks. Copia el objeto `hooks` de tu `.claude/settings.json` o `settings.local.json`, ya que el formato es el mismo. El comando recibe entrada de hook como JSON en stdin, así que usa `jq` para extraer la ruta del archivo:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
"jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: Prueba tu plugin migrado

Carga tu plugin para verificar que todo funciona:

```
claude --plugin-dir ./my-plugin
```

Prueba cada componente: ejecuta tus comandos, verifica que los agentes aparezcan en `/agents` y verifica que los hooks se activen correctamente.

Qué cambia al migrar

Independiente (<code>.claude/</code>)	Plugin
Solo disponible en un proyecto	Se puede compartir a través de marketplaces
Archivos en <code>.claude/commands/</code>	Archivos en <code>plugin-name/commands/</code>
Hooks en <code>settings.json</code>	Hooks en <code>hooks/hooks.json</code>
Debe copiar manualmente para compartir	Instalar con <code>/plugin install</code>

Note:

Después de migrar, puedes eliminar los archivos originales de `.claude/` para evitar duplicados. La versión del plugin tendrá prioridad cuando se cargue.

Próximos pasos

Ahora que entiendes el sistema de plugins de Claude Code, aquí hay caminos sugeridos para diferentes objetivos:

Para usuarios de plugins

- [Descubrir e instalar plugins](#): examina marketplaces e instala plugins
- [Configura marketplaces de equipo](#): configura plugins a nivel de repositorio para tu equipo

Para desarrolladores de plugins

- [Crear y distribuir un marketplace](#): empaqueta y comparte tus plugins
- [Referencia de plugins](#): especificaciones técnicas completas
- Profundiza en componentes específicos del plugin:
 - [Skills](#): detalles de desarrollo de skills
 - [Subagents](#): configuración y capacidades del agente
 - [Hooks](#): manejo de eventos y automatización
 - [MCP](#): integración de herramientas externas

Referencia de plugins

Referencia técnica completa para el sistema de plugins de Claude Code, incluyendo esquemas, comandos CLI y especificaciones de componentes.

Tip:

¿Buscas instalar plugins? Consulta [Descubrir e instalar plugins](#). Para crear plugins, consulta [Plugins](#). Para distribuir plugins, consulta [Marketplaces de plugins](#).

Esta referencia proporciona especificaciones técnicas completas para el sistema de plugins de Claude Code, incluyendo esquemas de componentes, comandos CLI y herramientas de desarrollo.

Un **plugin** es un directorio independiente de componentes que extiende Claude Code con funcionalidad personalizada. Los componentes del plugin incluyen skills, agents, hooks, MCP servers y LSP servers.

Referencia de componentes de plugins

Skills

Los plugins añaden skills a Claude Code, creando atajos `/name` que tú o Claude pueden invocar.

Ubicación: Directorio `skills/` o `commands/` en la raíz del plugin

Formato de archivo: Los skills son directorios con `SKILL.md`; los comandos son archivos markdown simples

Estructura de skill:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (opcional)
│   └── scripts/ (opcional)
└── code-reviewer/
    └── SKILL.md
```

Comportamiento de integración:

- Los skills y comandos se descubren automáticamente cuando se instala el plugin
- Claude puede invocarlos automáticamente según el contexto de la tarea
- Los skills pueden incluir archivos de apoyo junto a SKILL.md

Para obtener detalles completos, consulta [Skills](#).

Agents

Los plugins pueden proporcionar subagents especializados para tareas específicas que Claude puede invocar automáticamente cuando sea apropiado.

Ubicación: Directorio `agents/` en la raíz del plugin

Formato de archivo: Archivos markdown que describen las capacidades del agent

Estructura del agent:

```
---
name: agent-name
description: En qué se especializa este agent y cuándo Claude debe invocarlo
---
```

Prompt del sistema detallado para el agent describiendo su rol, experiencia y comportamiento.

Puntos de integración:

- Los agents aparecen en la interfaz `/agents`
- Claude puede invocar agents automáticamente según el contexto de la tarea
- Los agents pueden ser invocados manualmente por los usuarios
- Los agents del plugin funcionan junto con los agents integrados de Claude

Para obtener detalles completos, consulta [Subagents](#).

Hooks

Los plugins pueden proporcionar manejadores de eventos que responden automáticamente a eventos de Claude Code.

Ubicación: `hooks/hooks.json` en la raíz del plugin, o en línea en `plugin.json`

Formato: Configuración JSON con coincidencias de eventos y acciones

Configuración de hook:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

Eventos disponibles:

- **PreToolUse** : Antes de que Claude use cualquier herramienta
- **PostToolUse** : Después de que Claude use exitosamente cualquier herramienta
- **PostToolUseFailure** : Después de que falle la ejecución de una herramienta de Claude
- **PermissionRequest** : Cuando se muestra un diálogo de permiso
- **UserPromptSubmit** : Cuando el usuario envía un prompt
- **Notification** : Cuando Claude Code envía notificaciones
- **Stop** : Cuando Claude intenta detenerse
- **SubagentStart** : Cuando se inicia un subagent
- **SubagentStop** : Cuando un subagent intenta detenerse

- `SessionStart` : Al comienzo de las sesiones
- `SessionEnd` : Al final de las sesiones
- `TeammateIdle` : Cuando un compañero del equipo de agents está a punto de quedarse inactivo
- `TaskCompleted` : Cuando una tarea está siendo marcada como completada
- `PreCompact` : Antes de que se compacte el historial de conversación

Tipos de hook:

- `command` : Ejecutar comandos de shell o scripts
- `prompt` : Evaluar un prompt con un LLM (usa el marcador de posición `$ARGUMENTS` para el contexto)
- `agent` : Ejecutar un verificador agentic con herramientas para tareas de verificación complejas

MCP servers

Los plugins pueden agrupar servidores Model Context Protocol (MCP) para conectar Claude Code con herramientas y servicios externos.

Ubicación: `.mcp.json` en la raíz del plugin, o en línea en `plugin.json`

Formato: Configuración estándar del servidor MCP

Configuración del servidor MCP:

```
{
  "mcpServers": {
    "plugin-database": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
      "env": {
        "DB_PATH": "${CLAUDE_PLUGIN_ROOT}/data"
      }
    },
    "plugin-api-client": {
      "command": "npx",
      "args": ["@company/mcp-server", "--plugin-mode"],
      "cwd": "${CLAUDE_PLUGIN_ROOT}"
    }
  }
}
```

Comportamiento de integración:

- Los servidores MCP del plugin se inician automáticamente cuando se habilita el plugin
- Los servidores aparecen como herramientas MCP estándar en el kit de herramientas de Claude
- Las capacidades del servidor se integran sin problemas con las herramientas existentes de Claude
- Los servidores del plugin se pueden configurar independientemente de los servidores MCP del usuario

LSP servers

Tip:

¿Buscas usar plugins LSP? Instálalos desde el marketplace oficial: busca “lsp” en la pestaña Discover de [/plugin](#). Esta sección documenta cómo crear plugins LSP para lenguajes no cubiertos por el marketplace oficial.

Los plugins pueden proporcionar servidores [Language Server Protocol](#) (LSP) para dar a Claude inteligencia de código en tiempo real mientras trabajas en tu base de código.

La integración de LSP proporciona:

- **Diagnósticos instantáneos:** Claude ve errores y advertencias inmediatamente después de cada edición
- **Navegación de código:** ir a definición, encontrar referencias e información al pasar el ratón
- **Conciencia del lenguaje:** información de tipo y documentación para símbolos de código

Ubicación: `.lsp.json` en la raíz del plugin, o en línea en `plugin.json`

Formato: Configuración JSON que asigna nombres de servidores de lenguaje a sus configuraciones

Formato del archivo `.lsp.json` :

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

En línea en `plugin.json` :

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

Campos requeridos:

Campo	Descripción
<code>command</code>	El binario LSP a ejecutar (debe estar en PATH)
<code>extensionToLanguage</code>	Asigna extensiones de archivo a identificadores de lenguaje

Campos opcionales:

Campo	Descripción
<code>args</code>	Argumentos de línea de comandos para el servidor LSP
<code>transport</code>	Transporte de comunicación: <code>stdio</code> (predeterminado) o <code>socket</code>
<code>env</code>	Variables de entorno a establecer al iniciar el servidor
<code>initializationOptions</code>	Opciones pasadas al servidor durante la inicialización
<code>settings</code>	Configuración pasada a través de <code>workspace/didChangeConfiguration</code>
<code>workspaceFolder</code>	Ruta de carpeta de espacio de trabajo para el servidor
<code>startupTimeout</code>	Tiempo máximo para esperar el inicio del servidor (milisegundos)
<code>shutdownTimeout</code>	Tiempo máximo para esperar el apagado elegante (milisegundos)
<code>restartOnCrash</code>	Si se debe reiniciar automáticamente el servidor si se bloquea

Campo	Descripción
<code>maxRestarts</code>	Número máximo de intentos de reinicio antes de rendirse

Warning:

Debes instalar el binario del servidor de lenguaje por separado. Los plugins LSP configuran cómo Claude Code se conecta a un servidor de lenguaje, pero no incluyen el servidor en sí. Si ves `Executable not found in $PATH` en la pestaña Errors de `/plugin`, instala el binario requerido para tu lenguaje.

Plugins LSP disponibles:

Plugin	Servidor de lenguaje	Comando de instalación
<code>pyright-lsp</code>	Pyright (Python)	<code>pip install pyright</code> o <code>npm install -g pyright</code>
<code>typescript-lsp</code>	TypeScript Language Server	<code>npm install -g typescript-language-server typescript</code>
<code>rust-lsp</code>	rust-analyzer	Ver instalación de rust-analyzer

Instala el servidor de lenguaje primero, luego instala el plugin desde el marketplace.

Alcances de instalación de plugins

Cuando instalas un plugin, eliges un **alcance** que determina dónde está disponible el plugin y quién más puede usarlo:

Alcance	Archivo de configuración	Caso de uso
<code>user</code>	<code>~/.claude/settings.json</code>	Plugins personales disponibles en todos los proyectos (predeterminado)
<code>project</code>	<code>.claude/settings.json</code>	Plugins de equipo compartidos a través del control de versiones

Alcance	Archivo de configuración	Caso de uso
local	.claude/settings.local.json	Plugins específicos del proyecto, ignorados por git
managed	Configuración administrada	Plugins administrados (solo lectura, solo actualizar)

Los plugins utilizan el mismo sistema de alcance que otras configuraciones de Claude Code. Para instrucciones de instalación y banderas de alcance, consulta [Instalar plugins](#). Para una explicación completa de los alcances, consulta [Alcances de configuración](#).

Esquema del manifiesto del plugin

El archivo `.claude-plugin/plugin.json` define los metadatos y la configuración de tu plugin. Esta sección documenta todos los campos y opciones soportados.

El manifiesto es opcional. Si se omite, Claude Code descubre automáticamente componentes en [ubicaciones predeterminadas](#) y deriva el nombre del plugin del nombre del directorio. Usa un manifiesto cuando necesites proporcionar metadatos o rutas de componentes personalizadas.

Esquema completo

```
{
  "name": "plugin-name",
  "version": "1.2.0",
  "description": "Brief plugin description",
  "author": {
    "name": "Author Name",
    "email": "author@example.com",
    "url": "https://github.com/author"
  },
  "homepage": "https://docs.example.com/plugin",
  "repository": "https://github.com/author/plugin",
  "license": "MIT",
  "keywords": ["keyword1", "keyword2"],
  "commands": ["/custom/commands/special.md"],
  "agents": "/custom/agents/",
  "skills": "/custom/skills/",
  "hooks": "/config/hooks.json",
  "mcpServers": "/mcp-config.json",
  "outputStyles": "/styles/",
  "lspServers": "/.lsp.json"
}
```

Campos requeridos

Si incluyes un manifiesto, `name` es el único campo requerido.

Campo	Tipo	Descripción	Ejemplo
<code>name</code>	string	Identificador único (kebab-case, sin espacios)	<code>"deployment-tools"</code>

Este nombre se utiliza para espacios de nombres de componentes. Por ejemplo, en la interfaz de usuario, el agent `agent-creator` para el plugin con nombre `plugin-dev` aparecerá como `plugin-dev:agent-creator`.

Campos de metadatos

Campo	Tipo	Descripción	Ejemplo
version	string	Versión semántica. Si también se establece en la entrada del marketplace, plugin.json tiene prioridad. Solo necesitas establecerlo en un lugar.	"2.1.0"
description	string	Explicación breve del propósito del plugin	"Deployment automation tools"
author	object	Información del autor	{"name": "Dev Team", "email": "dev@company.com" }
homepage	string	URL de documentación	"https://docs.example.com"
repository	string	URL del código fuente	"https://github.com/user/plugin"
license	string	Identificador de licencia	"MIT", "Apache-2.0"
keywords	array	Etiquetas de descubrimiento	["deployment", "ci-cd"]

Campos de ruta de componentes

Campo	Tipo	Descripción	Ejemplo
commands	string array	Archivos/directorios de comandos adicionales	". /custom/cmd.md" o [". /cmd1.md"]

Campo	Tipo	Descripción	Ejemplo
agents	string array	Archivos de agents adicionales	<code>"./custom/agents/reviewer.md"</code>
skills	string array	Directorios de skills adicionales	<code>"./custom/skills/"</code>
hooks	string array object	Rutas de configuración de hooks o configuración en línea	<code>"./my-extra-hooks.json"</code>
mcpServers	string array object	Rutas de configuración de MCP o configuración en línea	<code>"./my-extra-mcp-config.json"</code>
outputStyles	string array	Archivos/directorios de estilos de salida adicionales	<code>"./styles/"</code>
lspServers	string array object	Configuraciones de Language Server Protocol para inteligencia de código (ir a definición, encontrar referencias, etc.)	<code>"./.lsp.json"</code>

Reglas de comportamiento de rutas

Importante: Las rutas personalizadas complementan los directorios predeterminados - no los reemplazan.

- Si `commands/` existe, se carga además de las rutas de comandos personalizadas
- Todas las rutas deben ser relativas a la raíz del plugin y comenzar con `./`
- Los comandos de rutas personalizadas utilizan las mismas reglas de nomenclatura y espacios de nombres
- Se pueden especificar múltiples rutas como arrays para flexibilidad

Ejemplos de rutas:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

Variables de entorno

`${CLAUDE_PLUGIN_ROOT}` : Contiene la ruta absoluta a tu directorio de plugin. Úsala en hooks, servidores MCP y scripts para asegurar rutas correctas independientemente de la ubicación de instalación.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

Almacenamiento en caché de plugins y resolución de archivos

Los plugins se especifican de una de dos formas:

- A través de `claude --plugin-dir`, durante la duración de una sesión.

- A través de un marketplace, instalado para sesiones futuras.

Por razones de seguridad y verificación, Claude Code copia plugins del *marketplace* a la **caché de plugins** local del usuario (`~/.claude/plugins/cache`) en lugar de usarlos en su lugar. Entender este comportamiento es importante al desarrollar plugins que hacen referencia a archivos externos.

Limitaciones de traversal de rutas

Los plugins instalados no pueden hacer referencia a archivos fuera de su directorio. Las rutas que travesan fuera de la raíz del plugin (como `../shared-utils`) no funcionarán después de la instalación porque esos archivos externos no se copian a la caché.

Trabajar con dependencias externas

Si tu plugin necesita acceder a archivos fuera de su directorio, puedes crear enlaces simbólicos a archivos externos dentro de tu directorio de plugin. Los enlaces simbólicos se respetan durante el proceso de copia:

```
## Dentro de tu directorio de plugin
ln -s /path/to/shared-utils ./shared-utils
```

El contenido vinculado simbólicamente se copiará en la caché del plugin. Esto proporciona flexibilidad mientras se mantienen los beneficios de seguridad del sistema de almacenamiento en caché.

Estructura del directorio del plugin

Diseño estándar del plugin

Un plugin completo sigue esta estructura:

```

enterprise-plugin/
├── .claude-plugin/          # Directorio de metadatos (opcional)
│   └── plugin.json         # manifiesto del plugin
├── commands/               # Ubicación de comando predeterminada
│   ├── status.md
│   └── logs.md
├── agents/                 # Ubicación de agent predeterminada
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                 # Skills del Agent
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                  # Configuraciones de hooks
│   ├── hooks.json          # Configuración principal de hooks
│   └── security-hooks.json # Hooks adicionales
├── settings.json           # Configuración predeterminada para el plugin
├── .mcp.json               # Definiciones del servidor MCP
├── .lsp.json               # Configuraciones del servidor LSP
├── scripts/                # Scripts de hooks y utilidades
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                 # Archivo de licencia
└── CHANGELOG.md            # Historial de versiones

```

Warning:

El directorio `.claude-plugin/` contiene el archivo `plugin.json`. Todos los otros directorios (`commands/`, `agents/`, `skills/`, `hooks/`) deben estar en la raíz del plugin, no dentro de `.claude-plugin/`.

Referencia de ubicaciones de archivos

Componente	Ubicación predeterminada	Propósito
Manifiesto	<code>.claude-plugin/plugin.json</code>	Metadatos y configuración del plugin (opcional)
Comandos	<code>commands/</code>	Archivos Markdown de Skill (heredado; usa <code>skills/</code> para nuevos skills)
Agents	<code>agents/</code>	Archivos Markdown de Subagent
Skills	<code>skills/</code>	Skills con estructura <code><name>/SKILL.md</code>
Hooks	<code>hooks/hooks.json</code>	Configuración de hooks
Servidores MCP	<code>.mcp.json</code>	Definiciones del servidor MCP
Servidores LSP	<code>.lsp.json</code>	Configuraciones del servidor de lenguaje
Configuración	<code>settings.json</code>	Configuración predeterminada aplicada cuando se habilita el plugin. Actualmente solo se soportan configuraciones de <code>agent</code>

Referencia de comandos CLI

Claude Code proporciona comandos CLI para la gestión de plugins no interactiva, útil para scripting y automatización.

plugin install

Instala un plugin desde los marketplaces disponibles.

```
claude plugin install <plugin> [options]
```

Argumentos:

- `<plugin>` : Nombre del plugin o `plugin-name@marketplace-name` para un marketplace específico

Opciones:

Opción	Descripción	Predeterminado
<code>-s, --scope <scope></code>	Alcance de instalación: <code>user</code> , <code>project</code> , o <code>local</code>	<code>user</code>
<code>-h, --help</code>	Mostrar ayuda para el comando	

El alcance determina qué archivo de configuración se añade el plugin instalado. Por ejemplo, `--scope project` escribe en `enabledPlugins` en `.claude/settings.json`, haciendo que el plugin esté disponible para todos los que clonan el repositorio del proyecto.

Ejemplos:

```
## Instalar en alcance de usuario (predeterminado)
claude plugin install formatter@my-marketplace

## Instalar en alcance de proyecto (compartido con el equipo)
claude plugin install formatter@my-marketplace --scope project

## Instalar en alcance local (ignorado por git)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

Elimina un plugin instalado.

```
claude plugin uninstall <plugin> [options]
```

Argumentos:

- `<plugin>` : Nombre del plugin o `plugin-name@marketplace-name`

Opciones:

Opción	Descripción	Predeterminado
<code>-s, --scope <scope></code>	Desinstalar del alcance: <code>user</code> , <code>project</code> , o <code>local</code>	<code>user</code>
<code>-h, --help</code>	Mostrar ayuda para el comando	

Alias: `remove`, `rm`

plugin enable

Habilita un plugin deshabilitado.

```
claude plugin enable <plugin> [options]
```

Argumentos:

- `<plugin>`: Nombre del plugin o `plugin-name@marketplace-name`

Opciones:

Opción	Descripción	Predeterminado
<code>-s, --scope <scope></code>	Alcance a habilitar: <code>user</code> , <code>project</code> , o <code>local</code>	<code>user</code>
<code>-h, --help</code>	Mostrar ayuda para el comando	

plugin disable

Deshabilita un plugin sin desinstalarlo.

```
claude plugin disable <plugin> [options]
```

Argumentos:

- `<plugin>`: Nombre del plugin o `plugin-name@marketplace-name`

Opciones:

Opción	Descripción	Predeterminado
<code>-s, --scope <scope></code>	Alcance a deshabilitar: <code>user</code> , <code>project</code> , o <code>local</code>	<code>user</code>

Opción	Descripción	Predeterminado
<code>-h, --help</code>	Mostrar ayuda para el comando	

plugin update

Actualiza un plugin a la versión más reciente.

```
claude plugin update <plugin> [options]
```

Argumentos:

- `<plugin>` : Nombre del plugin o `plugin-name@marketplace-name`

Opciones:

Opción	Descripción	Predeterminado
<code>-s, --scope <scope></code>	Alcance a actualizar: <code>user</code> , <code>project</code> , <code>local</code> , o <code>managed</code>	<code>user</code>
<code>-h, --help</code>	Mostrar ayuda para el comando	

Herramientas de depuración y desarrollo

Comandos de depuración

Usa `claude --debug` (o `/debug` dentro de la TUI) para ver detalles de carga de plugins:

Esto muestra:

- Qué plugins se están cargando
- Cualquier error en los manifiestos del plugin
- Registro de comandos, agents y hooks
- Inicialización del servidor MCP

Problemas comunes

Problema	Causa	Solución
Plugin no se carga	<code>plugin.json</code> inválido	Valida la sintaxis JSON con <code>claude plugin validate</code> o <code>/plugin validate</code>
Los comandos no aparecen	Estructura de directorio incorrecta	Asegúrate de que <code>commands/</code> esté en la raíz, no en <code>.claude-plugin/</code>
Los hooks no se disparan	Script no ejecutable	Ejecuta <code>chmod +x script.sh</code>
El servidor MCP falla	Falta <code>\${CLAUDE_PLUGIN_ROOT}</code>	Usa la variable para todas las rutas del plugin
Errores de ruta	Se utilizan rutas absolutas	Todas las rutas deben ser relativas y comenzar con <code>.</code> /
LSP <code>Executable not found in \$PATH</code>	Servidor de lenguaje no instalado	Instala el binario (p. ej., <code>npm install -g typescript-language-server typescript</code>)

Mensajes de error de ejemplo

Errores de validación de manifiesto:

- `Invalid JSON syntax: Unexpected token }` in JSON at position 142 : busca comas faltantes, comas extra o cadenas sin comillas
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required` : falta un campo requerido
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...` : error de sintaxis JSON

Errores de carga de plugin:

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.` : la ruta del comando existe pero no contiene archivos de comando válidos

- Plugin directory not found at path: `./plugins/my-plugin`. Check that the marketplace entry has the correct path. : la ruta `source` en `marketplace.json` apunta a un directorio inexistente
- Plugin my-plugin has conflicting manifests: both `plugin.json` and marketplace entry specify components. : elimina definiciones de componentes duplicadas o elimina `strict: false` en la entrada del marketplace

Solución de problemas de hooks

El script del hook no se ejecuta:

1. Verifica que el script sea ejecutable: `chmod +x ./scripts/your-script.sh`
2. Verifica la línea shebang: La primera línea debe ser `#!/bin/bash` o `#!/usr/bin/env bash`
3. Verifica que la ruta use `${CLAUDE_PLUGIN_ROOT}`: `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. Prueba el script manualmente: `./scripts/your-script.sh`

El hook no se dispara en los eventos esperados:

1. Verifica que el nombre del evento sea correcto (sensible a mayúsculas): `PostToolUse`, no `postToolUse`
2. Verifica que el patrón del matcher coincida con tus herramientas: `"matcher": "Write|Edit"` para operaciones de archivo
3. Confirma que el tipo de hook sea válido: `command`, `prompt`, o `agent`

Solución de problemas del servidor MCP

El servidor no se inicia:

1. Verifica que el comando exista y sea ejecutable
2. Verifica que todas las rutas usen la variable `${CLAUDE_PLUGIN_ROOT}`
3. Verifica los registros del servidor MCP: `claude --debug` muestra errores de inicialización
4. Prueba el servidor manualmente fuera de Claude Code

Las herramientas del servidor no aparecen:

1. Asegúrate de que el servidor esté correctamente configurado en `.mcp.json` o `plugin.json`
2. Verifica que el servidor implemente correctamente el protocolo MCP
3. Busca tiempos de espera de conexión en la salida de depuración

Errores de estructura de directorio

Síntomas: El plugin se carga pero faltan componentes (comandos, agents, hooks).

Estructura correcta: Los componentes deben estar en la raíz del plugin, no dentro de `.claude-plugin/`. Solo `plugin.json` pertenece a `.claude-plugin/`.

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json    ← Solo el manifiesto aquí
├── commands/         ← A nivel de raíz
├── agents/            ← A nivel de raíz
└── hooks/             ← A nivel de raíz
```

Si tus componentes están dentro de `.claude-plugin/`, muévelos a la raíz del plugin.

Lista de verificación de depuración:

1. Ejecuta `claude --debug` y busca mensajes “loading plugin”
2. Verifica que cada directorio de componentes esté listado en la salida de depuración
3. Verifica que los permisos de archivo permitan leer los archivos del plugin

Referencia de distribución y versionado

Gestión de versiones

Sigue el versionado semántico para las versiones del plugin:

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

Formato de versión: `MAJOR.MINOR.PATCH`

- **MAJOR:** Cambios de ruptura (cambios de API incompatibles)
- **MINOR:** Nuevas características (adiciones compatibles hacia atrás)
- **PATCH:** Correcciones de errores (correcciones compatibles hacia atrás)

Mejores prácticas:

- Comienza en `1.0.0` para tu primer lanzamiento estable
- Actualiza la versión en `plugin.json` antes de distribuir cambios
- Documenta los cambios en un archivo `CHANGELOG.md`
- Usa versiones previas al lanzamiento como `2.0.0-beta.1` para pruebas

Warning:

Claude Code utiliza la versión para determinar si debe actualizar tu plugin. Si cambias el código de tu plugin pero no aumentas la versión en `plugin.json`, los usuarios existentes de tu plugin no verán tus cambios debido al almacenamiento en caché.

Si tu plugin está dentro de un directorio de [marketplace](#), puedes gestionar la versión a través de `marketplace.json` en su lugar y omitir el campo `version` de `plugin.json`.

Ver también

- [Plugins](#) - Tutoriales y uso práctico
- [Marketplaces de plugins](#) - Crear y gestionar marketplaces
- [Skills](#) - Detalles de desarrollo de skills
- [Subagents](#) - Configuración y capacidades del agent
- [Hooks](#) - Manejo de eventos y automatización
- [MCP](#) - Integración de herramientas externas
- [Configuración](#) - Opciones de configuración para plugins

Crear y distribuir un marketplace de plugins

Cree y aloje marketplaces de plugins para distribuir extensiones de Claude Code en equipos y comunidades.

Un **marketplace de plugins** es un catálogo que le permite distribuir plugins a otros. Los marketplaces proporcionan descubrimiento centralizado, seguimiento de versiones, actualizaciones automáticas y soporte para múltiples tipos de fuentes (repositorios git, rutas locales y más). Esta guía le muestra cómo crear su propio marketplace para compartir plugins con su equipo o comunidad.

¿Busca instalar plugins desde un marketplace existente? Consulte [Descubrir e instalar plugins precompilados](#).

Descripción general

Crear y distribuir un marketplace implica:

1. **Crear plugins:** construya uno o más plugins con comandos, agentes, hooks, MCP servers o servidores LSP. Esta guía asume que ya tiene plugins para distribuir; consulte [Crear plugins](#) para obtener detalles sobre cómo crearlos.
2. **Crear un archivo de marketplace:** defina un `marketplace.json` que enumere sus plugins y dónde encontrarlos (consulte [Crear el archivo de marketplace](#)).
3. **Alojar el marketplace:** envíe a GitHub, GitLab u otro host git (consulte [Alojar y distribuir marketplaces](#)).
4. **Compartir con usuarios:** los usuarios agregan su marketplace con `/plugin marketplace add` e instalan plugins individuales (consulte [Descubrir e instalar plugins](#)).

Una vez que su marketplace esté activo, puede actualizarlo enviando cambios a su repositorio. Los usuarios actualizan su copia local con `/plugin marketplace update`.

Tutorial: crear un marketplace local

Este ejemplo crea un marketplace con un plugin: una skill `/quality-review` para revisiones de código. Creará la estructura de directorios, agregará una skill, creará el manifiesto del plugin y el catálogo del marketplace, luego lo instalará y probará.

Step 1: Crear la estructura de directorios

```
mkdir -p my-marketplace/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/skills/quality-review
```

Step 2: Crear la skill

Cree un archivo `SKILL.md` que defina qué hace la skill `/quality-review`.

```
---
description: Review code for bugs, security, and performance
disable-model-invocation: true
---
```

Review the code I've selected or the recent changes for:

- Potential bugs or edge cases
- Security concerns
- Performance issues
- Readability improvements

Be concise and actionable.

Step 3: Crear el manifiesto del plugin

Cree un archivo `plugin.json` que describa el plugin. El manifiesto va en el directorio `.claude-plugin/`.

```
{
  "name": "quality-review-plugin",
  "description": "Adds a /quality-review skill for quick code reviews",
  "version": "1.0.0"
}
```

Step 4: Crear el archivo de marketplace

Cree el catálogo de marketplace que enumera su plugin.

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "Adds a /quality-review skill for quick code reviews"
    }
  ]
}
```

Step 5: Agregar e instalar

Agregue el marketplace e instale el plugin.

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: Pruébalo

Seleccione algo de código en su editor y ejecute su nuevo comando.

```
/review
```

Para obtener más información sobre lo que los plugins pueden hacer, incluidos hooks, agentes, MCP servers y servidores LSP, consulte [Plugins](#).

Note:

Cómo se instalan los plugins: Cuando los usuarios instalan un plugin, Claude Code copia el directorio del plugin a una ubicación de caché. Esto significa que los plugins no pueden hacer referencia a archivos fuera de su directorio usando rutas como `../shared-utils`, porque esos archivos no se copiarán.

Si necesita compartir archivos entre plugins, use enlaces simbólicos (que se siguen durante la copia). Consulte [Plugin caching and file resolution](#) para obtener detalles.

Crear el archivo de marketplace

Cree `.claude-plugin/marketplace.json` en la raíz de su repositorio. Este archivo define el nombre de su marketplace, información del propietario y una lista de plugins con sus fuentes.

Cada entrada de plugin necesita como mínimo un `name` y `source` (dónde obtenerlo). Consulte el [esquema completo](#) a continuación para todos los campos disponibles.

```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "Automatic code formatting on save",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "Deployment automation tools"
    }
  ]
}
```


Esquema de marketplace

Campos requeridos

Campo	Tipo	Descripción	Ejemplo
<code>name</code>	string	Identificador de marketplace (kebab-case, sin espacios). Esto es público: los usuarios lo ven al instalar plugins (por ejemplo, <code>/plugin install my-tool@your-marketplace</code>).	<code>"acme-tools"</code>
<code>owner</code>	object	Información del mantenedor del marketplace (consulte los campos a continuación)	
<code>plugins</code>	array	Lista de plugins disponibles	Ver a continuación

Note:

Nombres reservados: Los siguientes nombres de marketplace están reservados para uso oficial de Anthropic y no pueden ser utilizados por marketplaces de terceros: `claude-code-marketplace`, `claude-code-plugins`, `claude-plugins-official`, `anthropic-marketplace`, `anthropic-plugins`, `agent-skills`, `life-sciences`. Los nombres que se hacen pasar por marketplaces oficiales (como `official-claude-plugins` o `anthropic-tools-v2`) también están bloqueados.

Campos del propietario

Campo	Tipo	Requerido	Descripción
<code>name</code>	string	Sí	Nombre del mantenedor o equipo

Campo	Tipo	Requerido	Descripción
<code>email</code>	string	No	Correo electrónico de contacto del mantenedor

Metadatos opcionales

Campo	Tipo	Descripción
<code>metadata.description</code>	string	Descripción breve del marketplace
<code>metadata.version</code>	string	Versión del marketplace
<code>metadata.pluginRoot</code>	string	Directorio base antepuesto a rutas de fuente de plugin relativas (por ejemplo, <code>"/plugins"</code> le permite escribir <code>"source": "formatter"</code> en lugar de <code>"source": "/plugins/formatter"</code>)

Entradas de plugins

Cada entrada de plugin en el array `plugins` describe un plugin y dónde encontrarlo. Puede incluir cualquier campo del [esquema de manifiesto de plugin](#) (como `description` , `version` , `author` , `commands` , `hooks` , etc.), más estos campos específicos del marketplace: `source` , `category` , `tags` y `strict` .

Campos requeridos

Campo	Tipo	Descripción
<code>name</code>	string	Identificador de plugin (kebab-case, sin espacios). Esto es público: los usuarios lo ven al instalar (por ejemplo, <code>/plugin install my-plugin@marketplace</code>).

Campo	Tipo	Descripción
<code>source</code>	string object	Dónde obtener el plugin (consulte Fuentes de plugins a continuación)

Campos de plugin opcionales

Campos de metadatos estándar:

Campo	Tipo	Descripción
<code>description</code>	string	Descripción breve del plugin
<code>version</code>	string	Versión del plugin
<code>author</code>	object	Información del autor del plugin (<code>name</code> requerido, <code>email</code> opcional)
<code>homepage</code>	string	URL de página de inicio o documentación del plugin
<code>repository</code>	string	URL del repositorio de código fuente
<code>license</code>	string	Identificador de licencia SPDX (por ejemplo, MIT, Apache-2.0)
<code>keywords</code>	array	Etiquetas para descubrimiento y categorización de plugins
<code>category</code>	string	Categoría del plugin para organización
<code>tags</code>	array	Etiquetas para búsqueda
<code>strict</code>	boolean	Controla si <code>plugin.json</code> es la autoridad para definiciones de componentes (predeterminado: true). Consulte Modo estricto a continuación.

Campos de configuración de componentes:

Campo	Tipo	Descripción
commands	string array	Rutas personalizadas a archivos o directorios de comandos
agents	string array	Rutas personalizadas a archivos de agentes
hooks	string object	Configuración de hooks personalizada o ruta a archivo de hooks
mcpServers	string object	Configuraciones de MCP server o ruta a configuración de MCP
lspServers	string object	Configuraciones de servidor LSP o ruta a configuración de LSP

Fuentes de plugins

Las fuentes de plugins le indican a Claude Code dónde obtener cada plugin individual listado en su marketplace. Estos se establecen en el campo `source` de cada entrada de plugin en `marketplace.json`.

Una vez que un plugin se clona o copia en la máquina local, se copia en el caché de plugins versionado local en `~/.claude/plugins/cache`.

Fuente	Tipo	Campos	Notas
Ruta relativa	<code>string</code> (p. ej. <code>"/my-plugin"</code>)	—	Directorio local dentro del repositorio de marketplace. Debe comenzar con <code>./</code>
github	object	<code>repo</code> , <code>ref?</code> , <code>sha?</code>	
url	object	<code>url</code> (debe terminar en <code>.git</code>), <code>ref?</code> , <code>sha?</code>	Fuente de URL de Git

Fuente	Tipo	Campos	Notas
<code>git-subdir</code>	object	<code>url</code> , <code>path</code> , <code>ref?</code> , <code>sha?</code>	Subdirectorio dentro de un repositorio git. Clona escasamente para minimizar el ancho de banda para monorepos
<code>npm</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Instalado vía <code>npm install</code>
<code>pip</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Instalado vía pip

Note:

Fuentes de marketplace vs fuentes de plugins: Estos son conceptos diferentes que controlan cosas diferentes.

- **Fuente de marketplace** — dónde obtener el catálogo `marketplace.json` en sí. Se establece cuando los usuarios ejecutan `/plugin marketplace add` o en la configuración `extraKnownMarketplaces`. Soporta `ref` (rama/etiqueta) pero no `sha`.
- **Fuente de plugin** — dónde obtener un plugin individual listado en el marketplace. Se establece en el campo `source` de cada entrada de plugin dentro de `marketplace.json`. Soporta tanto `ref` (rama/etiqueta) como `sha` (commit exacto).

Por ejemplo, un marketplace alojado en `acme-corp/plugin-catalog` (fuente de marketplace) puede listar un plugin obtenido de `acme-corp/code-formatter` (fuente de plugin). La fuente de marketplace y la fuente de plugin apuntan a diferentes repositorios y se fijan independientemente.

Rutas relativas

Para plugins en el mismo repositorio:

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

Note:

Las rutas relativas solo funcionan cuando los usuarios agregan su marketplace a través de Git (GitHub, GitLab o URL de git). Si los usuarios agregan su marketplace a través de una URL directa al archivo `marketplace.json`, las rutas relativas no se resolverán correctamente. Para distribución basada en URL, use fuentes de GitHub, npm o URL de git en su lugar. Consulte [Solución de problemas](#) para obtener detalles.

Repositorios de GitHub

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

Puede fijar a una rama, etiqueta o commit específico:

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Campo	Tipo	Descripción
<code>repo</code>	string	Requerido. Repositorio de GitHub en formato <code>owner/repo</code>

Campo	Tipo	Descripción
<code>ref</code>	string	Opcional. Rama o etiqueta de Git (por defecto es la rama predeterminada del repositorio)
<code>sha</code>	string	Opcional. SHA de commit de git completo de 40 caracteres para fijar a una versión exacta

Repositorios de Git

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

Puede fijar a una rama, etiqueta o commit específico:

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Campo	Tipo	Descripción
<code>url</code>	string	Requerido. URL completa del repositorio de git (debe terminar con <code>.git</code>)

Campo	Tipo	Descripción
<code>ref</code>	string	Opcional. Rama o etiqueta de Git (por defecto es la rama predeterminada del repositorio)
<code>sha</code>	string	Opcional. SHA de commit de git completo de 40 caracteres para fijar a una versión exacta

Subdirectorios de Git

Use `git-subdir` para apuntar a un plugin que vive dentro de un subdirectorio de un repositorio de git. Claude Code usa un clon parcial y escaso para obtener solo el subdirectorio, minimizando el ancho de banda para monorepos grandes.

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

Puede fijar a una rama, etiqueta o commit específico:

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```


El campo `url` también acepta una abreviatura de GitHub (`owner/repo`) o URLs SSH (`git@github.com:owner/repo.git`).

Campo	Tipo	Descripción
<code>url</code>	string	Requerido. URL del repositorio de Git, abreviatura de GitHub <code>owner/repo</code> o URL SSH
<code>path</code>	string	Requerido. Ruta del subdirectorio dentro del repositorio que contiene el plugin (por ejemplo, <code>"tools/claude-plugin"</code>)
<code>ref</code>	string	Opcional. Rama o etiqueta de Git (por defecto es la rama predeterminada del repositorio)
<code>sha</code>	string	Opcional. SHA de commit de git completo de 40 caracteres para fijar a una versión exacta

Paquetes npm

Los plugins distribuidos como paquetes npm se instalan usando `npm install` . Esto funciona con cualquier paquete en el registro npm público o un registro privado que su equipo aloje.

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

Para fijar a una versión específica, agregue el campo `version` :

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

Para instalar desde un registro privado o interno, agregue el campo `registry` :

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

Campo	Tipo	Descripción
<code>package</code>	string	Requerido. Nombre del paquete o paquete con alcance (por ejemplo, <code>@org/plugin</code>)
<code>version</code>	string	Opcional. Versión o rango de versión (por ejemplo, <code>2.1.0</code> , <code>^2.0.0</code> , <code>~1.5.0</code>)
<code>registry</code>	string	Opcional. URL de registro npm personalizado. Por defecto es el registro npm del sistema (típicamente <code>npmjs.org</code>)

Entradas de plugins avanzadas

Este ejemplo muestra una entrada de plugin usando muchos de los campos opcionales, incluidas rutas personalizadas para comandos, agentes, hooks y MCP servers:

```
{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "Enterprise workflow automation tools",
  "version": "2.1.0",
  "author": {
    "name": "Enterprise Team",
    "email": "enterprise@example.com"
  },
  "homepage": "https://docs.example.com/plugins/enterprise-tools",
  "repository": "https://github.com/company/enterprise-plugin",
  "license": "MIT",
  "keywords": ["enterprise", "workflow", "automation"],
  "category": "productivity",
  "commands": [
    "./commands/core/",
    "./commands/enterprise/",
    "./commands/experimental/preview.md"
  ],
  "agents": ["/agents/security-reviewer.md", "/agents/compliance-checker.md"],
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/validate.sh"
          }
        ]
      }
    ]
  },
  "mcpServers": {
    "enterprise-db": {
```

```
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"]
  },
  "strict": false
}
```

Cosas clave a notar:

- **commands y agents** : Puede especificar múltiples directorios o archivos individuales. Las rutas son relativas a la raíz del plugin.
- **\${CLAUDE_PLUGIN_ROOT}** : Use esta variable en hooks y configuraciones de MCP server para hacer referencia a archivos dentro del directorio de instalación del plugin. Esto es necesario porque los plugins se copian a una ubicación de caché cuando se instalan.
- **strict: false** : Dado que esto se establece en false, el plugin no necesita su propio **plugin.json** . La entrada del marketplace define todo. Consulte [Modo estricto](#) a continuación.

Modo estricto

El campo **strict** controla si **plugin.json** es la autoridad para definiciones de componentes (comandos, agentes, hooks, skills, MCP servers, estilos de salida).

Valor	Comportamiento
true (predeterminado)	plugin.json es la autoridad. La entrada del marketplace puede complementarla con componentes adicionales, y ambas fuentes se fusionan.
false	La entrada del marketplace es la definición completa. Si el plugin también tiene un plugin.json que declara componentes, eso es un conflicto y el plugin falla al cargar.

Cuándo usar cada modo:

- **strict: true** : el plugin tiene su propio **plugin.json** y gestiona sus propios componentes. La entrada del marketplace puede agregar comandos o hooks adicionales encima. Este es el predeterminado y funciona para la mayoría de los plugins.
- **strict: false** : el operador del marketplace quiere control total. El repositorio del plugin proporciona archivos sin procesar, y la entrada del marketplace define cuáles de esos archivos se exponen como comandos, agentes, hooks, etc. Útil cuando el ma-

marketplace reestructura o cura los componentes de un plugin de manera diferente a la que el autor del plugin pretendía.

Alojar y distribuir marketplaces

Alojar en GitHub (recomendado)

GitHub proporciona el método de distribución más fácil:

1. **Crear un repositorio:** Configure un nuevo repositorio para su marketplace
2. **Agregar archivo de marketplace:** Cree `.claude-plugin/marketplace.json` con sus definiciones de plugins
3. **Compartir con equipos:** Los usuarios agregan su marketplace con `/plugin marketplace add owner/repo`

Beneficios: Control de versiones integrado, seguimiento de problemas y características de colaboración en equipo.

Alojar en otros servicios de git

Cualquier servicio de alojamiento de git funciona, como GitLab, Bitbucket y servidores autohospedados. Los usuarios agregan con la URL completa del repositorio:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Repositorios privados

Claude Code soporta instalar plugins desde repositorios privados. Para instalación manual y actualizaciones, Claude Code usa sus ayudantes de credenciales de git existentes. Si `git clone` funciona para un repositorio privado en su terminal, funciona en Claude Code también. Los ayudantes de credenciales comunes incluyen `gh auth login` para GitHub, Keychain de macOS y `git-credential-store`.

Las actualizaciones automáticas en segundo plano se ejecutan al inicio sin ayudantes de credenciales, ya que los mensajes interactivos bloquearían que Claude Code se inicie. Para habilitar actualizaciones automáticas para marketplaces privados, establezca el token de autenticación apropiado en su entorno:

Proveedor	Variables de entorno	Notas
GitHub	<code>GITHUB_TOKEN</code> o <code>GH_TOKEN</code>	Token de acceso personal o token de GitHub App

Proveedor	Variables de entorno	Notas
GitLab	<code>GITLAB_TOKEN</code> o <code>GL_TOKEN</code>	Token de acceso personal o token de proyecto
Bitbucket	<code>BITBUCKET_TOKEN</code>	Contraseña de aplicación o token de acceso al repositorio

Establezca el token en su configuración de shell (por ejemplo, `.bashrc`, `.zshrc`) o páselo al ejecutar Claude Code:

```
export GITHUB_TOKEN=ghp_XXXXXXXXXXXXXXXXXXXX
```

Note:

Para entornos de CI/CD, configure el token como una variable de entorno secreta. GitHub Actions proporciona automáticamente `GITHUB_TOKEN` para repositorios en la misma organización.

Probar localmente antes de la distribución

Pruebe su marketplace localmente antes de compartirlo:

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

Para el rango completo de comandos add (GitHub, URLs de Git, rutas locales, URLs remotas), consulte [Agregar marketplaces](#).

Requerir marketplaces para su equipo

Puede configurar su repositorio para que los miembros del equipo sean automáticamente solicitados para instalar su marketplace cuando confíen en la carpeta del proyecto.

Agregue su marketplace a `.claude/settings.json`:

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

También puede especificar qué plugins deben estar habilitados de forma predeterminada:

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

Para opciones de configuración completas, consulte [Configuración de plugins](#).

Restricciones de marketplace administrado

Para organizaciones que requieren control estricto sobre las fuentes de plugins, los administradores pueden restringir qué marketplaces de plugins se permite a los usuarios agregar usando la configuración `strictKnownMarketplaces` en configuración administrada.

Cuando `strictKnownMarketplaces` se configura en configuración administrada, el comportamiento de restricción depende del valor:

Valor	Comportamiento
Indefinido (predeterminado)	Sin restricciones. Los usuarios pueden agregar cualquier marketplace
Array vacío <code>[]</code>	Bloqueo completo. Los usuarios no pueden agregar nuevos marketplaces

Valor	Comportamiento
Lista de fuentes	Los usuarios solo pueden agregar marketplaces que coincidan exactamente con la lista de permitidos

Configuraciones comunes

Deshabilitar todas las adiciones de marketplace:

```
{
  "strictKnownMarketplaces": []
}
```

Permitir solo marketplaces específicos:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    }
  ]
}
```

Permitir todos los marketplaces desde un servidor git interno usando coincidencia de patrón regex en el host:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Permitir marketplaces basados en sistema de archivos desde un directorio específico usando coincidencia de patrón regex en la ruta:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

Use `"."` como `pathPattern` para permitir cualquier ruta del sistema de archivos mientras aún controla fuentes de red con `hostPattern`.

Cómo funcionan las restricciones

Las restricciones se validan temprano en el proceso de instalación de plugins, antes de cualquier solicitud de red u operación del sistema de archivos. Esto previene intentos de acceso no autorizado a marketplaces.

La lista de permitidos usa coincidencia exacta para la mayoría de tipos de fuente. Para que un marketplace sea permitido, todos los campos especificados deben coincidir exactamente:

- Para fuentes de GitHub: `repo` es requerido, y `ref` o `path` también deben coincidir si se especifican en la lista de permitidos
- Para fuentes de URL: la URL completa debe coincidir exactamente
- Para fuentes `hostPattern`: el host del marketplace se compara contra el patrón regex

- Para fuentes `pathPattern` : la ruta del sistema de archivos del marketplace se compara contra el patrón regex

Debido a que `strictKnownMarketplaces` se establece en [configuración administrada](#), los usuarios individuales y las configuraciones del proyecto no pueden anular estas restricciones.

Para detalles de configuración completos incluyendo todos los tipos de fuente soportados y comparación con `extraKnownMarketplaces` , consulte la [referencia de strictKnownMarketplaces](#).

Resolución de versiones y canales de lanzamiento

Las versiones de plugins determinan rutas de caché y detección de actualizaciones. Puede especificar la versión en el manifiesto del plugin (`plugin.json`) o en la entrada del marketplace (`marketplace.json`).

Warning:

Cuando sea posible, evite establecer la versión en ambos lugares. El manifiesto del plugin siempre gana silenciosamente, lo que puede causar que la versión del marketplace sea ignorada. Para plugins de ruta relativa, establezca la versión en la entrada del marketplace. Para todas las otras fuentes de plugins, establézcala en el manifiesto del plugin.

Configurar canales de lanzamiento

Para soportar canales de lanzamiento “estable” y “último” para sus plugins, puede configurar dos marketplaces que apunten a diferentes refs o SHAs del mismo repositorio. Luego puede asignar los dos marketplaces a diferentes grupos de usuarios a través de [configuración administrada](#).

Warning:

El `plugin.json` del plugin debe declarar una `version` diferente en cada ref o commit fijado. Si dos refs o commits tienen la misma versión de manifiesto, Claude Code los trata como idénticos y omite la actualización.

Ejemplo

```
{
  "name": "stable-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "stable"
      }
    }
  ]
}
```

```
{
  "name": "latest-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "latest"
      }
    }
  ]
}
```

Asignar canales a grupos de usuarios

Asigne cada marketplace al grupo de usuarios apropiado a través de configuración administrada. Por ejemplo, el grupo estable recibe:

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

El grupo de acceso temprano recibe `latest-tools` en su lugar:

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

Validación y pruebas

Pruebe su marketplace antes de compartirlo.

Valide la sintaxis JSON de su marketplace:

```
claude plugin validate .
```

O desde dentro de Claude Code:

```
/plugin validate .
```

Agregue el marketplace para pruebas:

```
/plugin marketplace add ./path/to/marketplace
```

Instale un plugin de prueba para verificar que todo funciona:

```
/plugin install test-plugin@marketplace-name
```

Para flujos de trabajo completos de prueba de plugins, consulte [Pruebe sus plugins localmente](#). Para solución de problemas técnicos, consulte [Referencia de plugins](#).

Solución de problemas

Marketplace no se carga

Síntomas: No puede agregar marketplace o ver plugins de él

Soluciones:

- Verifique que la URL del marketplace sea accesible
- Compruebe que `.claude-plugin/marketplace.json` existe en la ruta especificada
- Asegúrese de que la sintaxis JSON sea válida usando `claude plugin validate` o `/plugin validate`
- Para repositorios privados, confirme que tiene permisos de acceso

Errores de validación de marketplace

Ejecute `claude plugin validate .` o `/plugin validate .` desde su directorio de marketplace para verificar problemas. Errores comunes:

Error	Causa	Solución
<code>File not found: .claude-plugin/marketplace.json</code>	Manifiesto faltante	Cree <code>.claude-plugin/marketplace.json</code> con campos requeridos
<code>Invalid JSON syntax: Unexpected token...</code>	Error de sintaxis JSON	Verifique comas faltantes, comas extra o cadenas sin comillas
<code>Duplicate plugin name "x" found in marketplace</code>	Dos plugins comparten el mismo nombre	Dé a cada plugin un valor <code>name</code> único

Error	Causa	Solución
<code>plugins[0].source: Path traversal not allowed</code>	La ruta de fuente contiene <code>..</code>	Use rutas relativas a la raíz del marketplace sin <code>..</code>

Advertencias (no bloqueantes):

- `Marketplace has no plugins defined` : agregue al menos un plugin al array `plugins`
- `No marketplace description provided` : agregue `metadata.description` para ayudar a los usuarios a entender su marketplace

Fallos de instalación de plugins

Síntomas: El marketplace aparece pero la instalación del plugin falla

Soluciones:

- Verifique que las URLs de fuente del plugin sean accesibles
- Compruebe que los directorios de plugins contengan archivos requeridos
- Para fuentes de GitHub, asegúrese de que los repositorios sean públicos o tenga acceso
- Pruebe las fuentes de plugins manualmente clonando/descargando

La autenticación del repositorio privado falla

Síntomas: Errores de autenticación al instalar plugins desde repositorios privados

Soluciones:

Para instalación manual y actualizaciones:

- Verifique que esté autenticado con su proveedor de git (por ejemplo, ejecute `gh auth status` para GitHub)
- Compruebe que su ayudante de credenciales esté configurado correctamente: `git config --global credential.helper`
- Intente clonar el repositorio manualmente para verificar que sus credenciales funcionan

Para actualizaciones automáticas en segundo plano:

- Establezca el token apropiado en su entorno: `echo $GITHUB_TOKEN`
- Compruebe que el token tiene los permisos requeridos (acceso de lectura al repositorio)

- Para GitHub, asegúrese de que el token tiene el alcance `repo` para repositorios privados
- Para GitLab, asegúrese de que el token tiene al menos alcance `read_repository`
- Verifique que el token no haya expirado

Las operaciones de Git agotan el tiempo de espera

Síntomas: La instalación del plugin o las actualizaciones del marketplace fallan con un error de tiempo de espera como “Git clone timed out after 120s” o “Git pull timed out after 120s”.

Causa: Claude Code usa un tiempo de espera de 120 segundos para todas las operaciones de git, incluida la clonación de repositorios de plugins y la extracción de actualizaciones de marketplace. Los repositorios grandes o las conexiones de red lentas pueden exceder este límite.

Solución: Aumente el tiempo de espera usando la variable de entorno `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`. El valor está en milisegundos:

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000 # 5 minutos
```

Los plugins con rutas relativas fallan en marketplaces basados en URL

Síntomas: Agregó un marketplace a través de URL (como `https://example.com/marketplace.json`), pero los plugins con fuentes de ruta relativa como `./plugins/my-plugin` fallan al instalar con errores “path not found”.

Causa: Los marketplaces basados en URL solo descargan el archivo `marketplace.json` en sí. No descargan archivos de plugins del servidor. Las rutas relativas en la entrada del marketplace hacen referencia a archivos en el servidor remoto que no fueron descargados.

Soluciones:

- **Use fuentes externas:** Cambie las entradas de plugins para usar fuentes de GitHub, npm o URL de git en lugar de rutas relativas:

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Use un marketplace basado en Git:** Aloje su marketplace en un repositorio de Git y agréguelo con la URL de git. Los marketplaces basados en Git clonan el repositorio completo, haciendo que las rutas relativas funcionen correctamente.

Archivos no encontrados después de la instalación

Síntomas: El plugin se instala pero las referencias a archivos fallan, especialmente archivos fuera del directorio del plugin

Causa: Los plugins se copian a un directorio de caché en lugar de usarse en el lugar. Las rutas que hacen referencia a archivos fuera del directorio del plugin (como `../shared-utils`) no funcionarán porque esos archivos no se copian.

Soluciones: Consulte [Plugin caching and file resolution](#) para soluciones alternativas incluyendo enlaces simbólicos y reestructuración de directorios.

Para herramientas de depuración adicionales y problemas comunes, consulte [Debugging and development tools](#).

Ver también

- [Descubrir e instalar plugins precompilados](#) - Instalación de plugins desde marketplaces existentes
- [Plugins](#) - Creación de sus propios plugins
- [Referencia de plugins](#) - Especificaciones técnicas completas y esquemas
- [Configuración de plugins](#) - Opciones de configuración de plugins
- [Referencia de strictKnownMarketplaces](#) - Restricciones de marketplace administrado

Descubra e instale plugins pregenerados a través de mercados

Encuentre e instale plugins de mercados para extender Claude Code con nuevos comandos, agentes y capacidades.

Los plugins extienden Claude Code con skills, agentes, hooks y servidores MCP. Los mercados de plugins son catálogos que le ayudan a descubrir e instalar estas extensiones sin construirlas usted mismo.

¿Busca crear y distribuir su propio mercado? Consulte [Crear y distribuir un mercado de plugins](#).

Cómo funcionan los mercados

Un mercado es un catálogo de plugins que alguien más ha creado y compartido. Usar un mercado es un proceso de dos pasos:

Step 1: Agregar el mercado

Esto registra el catálogo con Claude Code para que pueda explorar lo que está disponible. Aún no se instalan plugins.

Step 2: Instalar plugins individuales

Explore el catálogo e instale los plugins que desee.

Piénselo como agregar una tienda de aplicaciones: agregar la tienda le da acceso para explorar su colección, pero usted sigue eligiendo qué aplicaciones descargar individualmente.

Mercado oficial de Anthropic

El mercado oficial de Anthropic (`claude-plugins-official`) está disponible automáticamente cuando inicia Claude Code. Ejecute `/plugin` y vaya a la pestaña **Discover** para explorar lo que está disponible.

Para instalar un plugin del mercado oficial:

```
/plugin install plugin-name@claude-plugins-official
```

Note:

El mercado oficial es mantenido por Anthropic. Para enviar un plugin al mercado oficial, use uno de los formularios de envío en la aplicación:

- **Claude.ai:** claude.ai/settings/plugins/submit
- **Console:** platform.claude.com/plugins/submit

Para distribuir plugins de forma independiente, [cree su propio mercado](#) y compártalo con los usuarios.

El mercado oficial incluye varias categorías de plugins:

Inteligencia de código

Los plugins de inteligencia de código habilitan la herramienta LSP integrada de Claude Code, dándole a Claude la capacidad de saltar a definiciones, encontrar referencias y ver errores de tipo inmediatamente después de ediciones. Estos plugins configuran conexiones de [Language Server Protocol](#), la misma tecnología que potencia la inteligencia de código de VS Code.

Estos plugins requieren que el binario del servidor de lenguaje esté instalado en su sistema. Si ya tiene un servidor de lenguaje instalado, Claude puede solicitarle que instale el plugin correspondiente cuando abra un proyecto.

Lenguaje	Plugin	Binario requerido
C/C++	clangd-lsp	clangd
C#	csharp-lsp	csharp-ls
Go	gopls-lsp	gopls
Java	jdtls-lsp	jdtls
Kotlin	kotlin-lsp	kotlin-language-server
Lua	lua-lsp	lua-language-server
PHP	php-lsp	intelephense
Python	pyright-lsp	pyright-langserver

Lenguaje	Plugin	Binario requerido
Rust	<code>rust-analyzer-lsp</code>	<code>rust-analyzer</code>
Swift	<code>swift-lsp</code>	<code>sourcekit-lsp</code>
TypeScript	<code>typescript-lsp</code>	<code>typescript-language-server</code>

También puede [crear su propio plugin LSP](#) para otros lenguajes.

Note:

Si ve `Executable not found in $PATH` en la pestaña Errors de `/plugin` después de instalar un plugin, instale el binario requerido de la tabla anterior.

Lo que Claude gana con los plugins de inteligencia de código

Una vez que se instala un plugin de inteligencia de código y su binario de servidor de lenguaje está disponible, Claude gana dos capacidades:

- **Diagnósticos automáticos:** después de cada edición de archivo que Claude realiza, el servidor de lenguaje analiza los cambios e informa errores y advertencias automáticamente. Claude ve errores de tipo, importaciones faltantes y problemas de sintaxis sin necesidad de ejecutar un compilador o linter. Si Claude introduce un error, lo nota y corrige el problema en el mismo turno. Esto no requiere configuración más allá de instalar el plugin. Puede ver diagnósticos en línea presionando **Ctrl+O** cuando aparece el indicador “diagnostics found”.
- **Navegación de código:** Claude puede usar el servidor de lenguaje para saltar a definiciones, encontrar referencias, obtener información de tipo al pasar el ratón, listar símbolos, encontrar implementaciones y rastrear jerarquías de llamadas. Estas operaciones dan a Claude una navegación más precisa que la búsqueda basada en grep, aunque la disponibilidad puede variar según el lenguaje y el entorno.

Si encuentra problemas, consulte [Solución de problemas de inteligencia de código](#).

Integraciones externas

Estos plugins incluyen [servidores MCP](#) preconfigurados para que pueda conectar Claude a servicios externos sin configuración manual:

- **Control de fuente:** `github` , `gitlab`
- **Gestión de proyectos:** `atlassian` (Jira/Confluence), `asana` , `linear` , `notion`
- **Diseño:** `figma`

- **Infraestructura:** `vercel`, `firebase`, `supabase`
- **Comunicación:** `slack`
- **Monitoreo:** `sentry`

Flujos de trabajo de desarrollo

Plugins que agregan comandos y agentes para tareas de desarrollo comunes:

- **commit-commands:** Flujos de trabajo de confirmación de Git incluyendo confirmación, push y creación de PR
- **pr-review-toolkit:** Agentes especializados para revisar solicitudes de extracción
- **agent-sdk-dev:** Herramientas para construir con el Claude Agent SDK
- **plugin-dev:** Kit de herramientas para crear sus propios plugins

Estilos de salida

Personalice cómo responde Claude:

- **explanatory-output-style:** Información educativa sobre opciones de implementación
- **learning-output-style:** Modo de aprendizaje interactivo para construcción de habilidades

Pruébelo: agregue el mercado de demostración

Anthropic también mantiene un [mercado de plugins de demostración](#) (`claude-code-plugins`) con plugins de ejemplo que muestran lo que es posible con el sistema de plugins. A diferencia del mercado oficial, debe agregar este manualmente.

Step 1: Agregar el mercado

Desde dentro de Claude Code, ejecute el comando `plugin marketplace add` para el mercado `anthropics/claude-code`:

```
/plugin marketplace add anthropics/claude-code
```

Esto descarga el catálogo del mercado y pone sus plugins a su disposición.

Step 2: Explorar plugins disponibles

Ejecute `/plugin` para abrir el administrador de plugins. Esto abre una interfaz con pestañas con cuatro pestañas por las que puede ciclar usando **Tab** (o **Shift+Tab** para ir hacia atrás):

- **Discover:** explore plugins disponibles de todos sus mercados
- **Installed:** vea y administre sus plugins instalados
- **Marketplaces:** agregue, elimine o actualice sus mercados agregados
- **Errors:** vea cualquier error de carga de plugins

Vaya a la pestaña **Discover** para ver plugins del mercado que acaba de agregar.

Step 3: Instalar un plugin

Seleccione un plugin para ver sus detalles, luego elija un alcance de instalación:

- **User scope:** instale para usted en todos los proyectos
- **Project scope:** instale para todos los colaboradores en este repositorio
- **Local scope:** instale para usted en este repositorio solamente

Por ejemplo, seleccione **commit-commands** (un plugin que agrega comandos de flujo de trabajo de git) e instálelo en su alcance de usuario.

También puede instalar directamente desde la línea de comandos:

```
/plugin install commit-commands@anthropics-claude-code
```

Consulte [Alcances de configuración](#) para obtener más información sobre alcances.

Step 4: Usar su nuevo plugin

Después de instalar, los comandos del plugin están disponibles inmediatamente. Los comandos de plugin tienen espacios de nombres por el nombre del plugin, por lo que **commit-commands** proporciona comandos como `/commit-commands:commit`.

Pruébelo haciendo un cambio en un archivo y ejecutando:

```
/commit-commands:commit
```

Esto prepara sus cambios, genera un mensaje de confirmación y crea la confirmación.

Cada plugin funciona de manera diferente. Consulte la descripción del plugin en la pestaña **Discover** o su página de inicio para aprender qué comandos y capacidades proporciona.

El resto de esta guía cubre todas las formas en que puede agregar mercados, instalar plugins y administrar su configuración.

Agregar mercados

Use el comando `/plugin marketplace add` para agregar mercados de diferentes fuentes.

Tip:

Atajos: Puede usar `/plugin market` en lugar de `/plugin marketplace`, y `rm` en lugar de `remove`.

- **Repositorios de GitHub:** formato `owner/repo` (por ejemplo, `anthropics/claude-code`)
- **URLs de Git:** cualquier URL de repositorio de git (GitLab, Bitbucket, auto-hospedado)
- **Rutas locales:** directorios o rutas directas a archivos `marketplace.json`
- **URLs remotas:** URLs directas a archivos `marketplace.json` hospedados

Agregar desde GitHub

Agregue un repositorio de GitHub que contenga un archivo `.claude-plugin/marketplace.json` usando el formato `owner/repo` —donde `owner` es el nombre de usuario o la organización de GitHub y `repo` es el nombre del repositorio.

Por ejemplo, `anthropics/claude-code` se refiere al repositorio `claude-code` propiedad de `anthropics`:

```
/plugin marketplace add anthropics/claude-code
```

Agregar desde otros hosts de Git

Agregue cualquier repositorio de git proporcionando la URL completa. Esto funciona con cualquier host de Git, incluyendo GitLab, Bitbucket y servidores auto-hospedados:

Usando HTTPS:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Usando SSH:

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

Para agregar una rama o etiqueta específica, agregue `#` seguido de la ref:

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

Agregar desde rutas locales

Agregue un directorio local que contenga un archivo `.claude-plugin/marketplace.json` :

```
/plugin marketplace add ./my-marketplace
```

También puede agregar una ruta directa a un archivo `marketplace.json` :

```
/plugin marketplace add ./path/to/marketplace.json
```

Agregar desde URLs remotas

Agregue un archivo `marketplace.json` remoto a través de URL:

```
/plugin marketplace add https://example.com/marketplace.json
```

Note:

Los mercados basados en URL tienen algunas limitaciones en comparación con los mercados basados en Git. Si encuentra errores “path not found” al instalar plugins, consulte [Solución de problemas](#).

Instalar plugins

Una vez que haya agregado mercados, puede instalar plugins directamente (se instala en alcance de usuario por defecto):

```
/plugin install plugin-name@marketplace-name
```


Para elegir un [alcance de instalación](#) diferente, use la interfaz interactiva: ejecute `/plugin`, vaya a la pestaña **Discover** y presione **Enter** en un plugin. Verá opciones para:

- **User scope** (predeterminado): instale para usted en todos los proyectos
- **Project scope**: instale para todos los colaboradores en este repositorio (agrega a `.claude/settings.json`)
- **Local scope**: instale para usted en este repositorio solamente (no compartido con colaboradores)

También puede ver plugins con alcance **managed**—estos son instalados por administradores a través de [configuración administrada](#) y no pueden ser modificados.

Ejecute `/plugin` y vaya a la pestaña **Installed** para ver sus plugins agrupados por alcance.

Warning:

Asegúrese de confiar en un plugin antes de instalarlo. Anthropic no controla qué servidores MCP, archivos u otro software se incluyen en los plugins y no puede verificar que funcionen como se pretende. Consulte la página de inicio de cada plugin para obtener más información.

Administrar plugins instalados

Ejecute `/plugin` y vaya a la pestaña **Installed** para ver, habilitar, deshabilitar o desinstalar sus plugins. Escriba para filtrar la lista por nombre o descripción del plugin.

También puede administrar plugins con comandos directos.

Deshabilite un plugin sin desinstalarlo:

```
/plugin disable plugin-name@marketplace-name
```

Vuelva a habilitar un plugin deshabilitado:

```
/plugin enable plugin-name@marketplace-name
```

Elimine completamente un plugin:

```
/plugin uninstall plugin-name@marketplace-name
```

La opción `--scope` le permite dirigirse a un alcance específico con comandos CLI:

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

Aplicar cambios de plugins sin reiniciar

Cuando instala, habilita o deshabilita plugins durante una sesión, algunos cambios (como nuevos comandos y hooks) tienen efecto inmediatamente. Otros, incluyendo actualizaciones del servidor LSP, requieren un reinicio.

Para activar todos los cambios de plugins pendientes sin reiniciar, ejecute:

```
/reload-plugins
```

Claude Code recarga todos los plugins activos e informa qué se cargó. Si se agregaron o actualizaron servidores LSP, le informará que esos requieren un reinicio para tener efecto.

Administrar mercados

Puede administrar mercados a través de la interfaz interactiva `/plugin` o con comandos CLI.

Usar la interfaz interactiva

Ejecute `/plugin` y vaya a la pestaña **Marketplaces** para:

- Ver todos sus mercados agregados con sus fuentes y estado
- Agregar nuevos mercados
- Actualizar listados de mercados para obtener los últimos plugins
- Eliminar mercados que ya no necesita

Usar comandos CLI

También puede administrar mercados con comandos directos.

Enumere todos los mercados configurados:

```
/plugin marketplace list
```

Actualice listados de plugins de un mercado:

```
/plugin marketplace update marketplace-name
```

Elimine un mercado:

```
/plugin marketplace remove marketplace-name
```

Warning:

Eliminar un mercado desinstalará cualquier plugin que haya instalado desde él.

Configurar actualizaciones automáticas

Claude Code puede actualizar automáticamente mercados y sus plugins instalados al inicio. Cuando la actualización automática está habilitada para un mercado, Claude Code actualiza los datos del mercado e actualiza los plugins instalados a sus versiones más recientes. Si se actualizaron plugins, verá una notificación pidiéndole que ejecute

```
/reload-plugins .
```

Alterne la actualización automática para mercados individuales a través de la interfaz:

1. Ejecute `/plugin` para abrir el administrador de plugins
2. Seleccione **Marketplaces**
3. Elija un mercado de la lista
4. Seleccione **Enable auto-update** o **Disable auto-update**

Los mercados oficiales de Anthropic tienen la actualización automática habilitada por defecto. Los mercados de terceros y de desarrollo local tienen la actualización automática deshabilitada por defecto.

Para deshabilitar todas las actualizaciones automáticas completamente tanto para Claude Code como para todos los plugins, establezca la variable de entorno

`DISABLE_AUTOUPDATER`. Consulte [Actualizaciones automáticas](#) para obtener detalles.

Para mantener las actualizaciones automáticas de plugins habilitadas mientras se deshabilitan las actualizaciones automáticas de Claude Code, establezca

`FORCE_AUTOUPDATE_PLUGINS=true` junto con `DISABLE_AUTOUPDATER` :

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

Esto es útil cuando desea administrar las actualizaciones de Claude Code manualmente pero aún recibir actualizaciones automáticas de plugins.

Configurar mercados de equipo

Los administradores de equipo pueden configurar la instalación automática de mercados para proyectos agregando configuración de mercado a `.claude/settings.json`. Cuando los miembros del equipo confían en la carpeta del repositorio, Claude Code les solicita que instalen estos mercados y plugins.

Agregue `extraKnownMarketplaces` a su `.claude/settings.json` del proyecto:

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Para opciones de configuración completas incluyendo `extraKnownMarketplaces` y `enabledPlugins`, consulte [Configuración de plugins](#).

Seguridad

Los plugins y mercados son componentes altamente confiables que pueden ejecutar código arbitrario en su máquina con sus privilegios de usuario. Solo instale plugins y agregue mercados de fuentes en las que confíe. Las organizaciones pueden restringir qué mercados se permite a los usuarios agregar usando [restricciones de mercado administradas](#).

Solución de problemas

Comando /plugin no reconocido

Si ve “unknown command” o el comando `/plugin` no aparece:

1. **Verifique su versión:** Ejecute `claude --version`. Los plugins requieren la versión 1.0.33 o posterior.
2. **Actualice Claude Code:**
 - **Homebrew:** `brew upgrade claude-code`
 - **npm:** `npm update -g @anthropic-ai/claude-code`
 - **Instalador nativo:** Vuelva a ejecutar el comando de instalación desde [Setup](#)
3. **Reinicie Claude Code:** Después de actualizar, reinicie su terminal y ejecute `claude` nuevamente.

Problemas comunes

- **Mercado no cargando:** Verifique que la URL sea accesible y que `.claude-plugin/marketplace.json` exista en la ruta
- **Fallos de instalación de plugins:** Verifique que las URLs de fuente de plugins sean accesibles y que los repositorios sean públicos (o tenga acceso)
- **Archivos no encontrados después de la instalación:** Los plugins se copian a un caché, por lo que las rutas que hacen referencia a archivos fuera del directorio del plugin no funcionarán
- **Habilidades de plugins no apareciendo:** Limpie el caché con `rm -rf ~/.claude/plugins/cache`, reinicie Claude Code y reinstale el plugin.

Para solución de problemas detallada con soluciones, consulte [Solución de problemas](#) en la guía de mercados. Para herramientas de depuración, consulte [Herramientas de depuración y desarrollo](#).

Problemas de inteligencia de código

- **Servidor de lenguaje no iniciando:** verifique que el binario esté instalado y disponible en su `$PATH`. Consulte la pestaña Errors de `/plugin` para obtener detalles.
- **Alto uso de memoria:** los servidores de lenguaje como `rust-analyzer` y `pyright` pueden consumir memoria significativa en proyectos grandes. Si experimenta problemas de memoria, deshabilite el plugin con `/plugin disable <plugin-name>` y confíe en las herramientas de búsqueda integradas de Claude en su lugar.
- **Diagnósticos falsos positivos en monorepos:** los servidores de lenguaje pueden reportar errores de importación no resuelta para paquetes internos si el espacio de

trabajo no está configurado correctamente. Estos no afectan la capacidad de Claude para editar código.

Próximos pasos

- **Construya sus propios plugins:** Consulte [Plugins](#) para crear skills, agentes y hooks
- **Cree un mercado:** Consulte [Crear un mercado de plugins](#) para distribuir plugins a su equipo o comunidad
- **Referencia técnica:** Consulte [Referencia de plugins](#) para especificaciones completas

Part 6: Advanced & Automation

Crear subagentes personalizados

Cree y utilice subagentes de IA especializados en Claude Code para flujos de trabajo específicos de tareas y una mejor gestión del contexto.

Los subagentes son asistentes de IA especializados que manejan tipos específicos de tareas. Cada subagente se ejecuta en su propia ventana de contexto con un mensaje del sistema personalizado, acceso a herramientas específicas y permisos independientes. Cuando Claude encuentra una tarea que coincide con la descripción de un subagente, delega en ese subagente, que trabaja de forma independiente y devuelve resultados.

Note:

Si necesita múltiples agentes trabajando en paralelo y comunicándose entre sí, consulte [equipos de agentes](#) en su lugar. Los subagentes funcionan dentro de una única sesión; los equipos de agentes se coordinan entre sesiones separadas.

Los subagentes le ayudan a:

- **Preservar contexto** manteniendo la exploración e implementación fuera de su conversación principal
- **Aplicar restricciones** limitando qué herramientas puede usar un subagente
- **Reutilizar configuraciones** en proyectos con subagentes a nivel de usuario
- **Especializar el comportamiento** con mensajes del sistema enfocados para dominios específicos
- **Controlar costos** enrutando tareas a modelos más rápidos y económicos como Haiku

Claude utiliza la descripción de cada subagente para decidir cuándo delegar tareas. Cuando crea un subagente, escriba una descripción clara para que Claude sepa cuándo usarlo.

Claude Code incluye varios subagentes integrados como **Explore**, **Plan** y **general-purpose**. También puede crear subagentes personalizados para manejar tareas específicas. Esta página cubre los [subagentes integrados](#), [cómo crear los suyos](#), [opciones de configuración completas](#), [patrones para trabajar con subagentes](#) y [subagentes de ejemplo](#).

Subagentes integrados

Claude Code incluye subagentes integrados que Claude utiliza automáticamente cuando es apropiado. Cada uno hereda los permisos de la conversación principal con restricciones de herramientas adicionales.

Explore

Un agente rápido y de solo lectura optimizado para buscar y analizar bases de código.

- **Modelo:** Haiku (rápido, baja latencia)
- **Herramientas:** Herramientas de solo lectura (acceso denegado a herramientas Write y Edit)
- **Propósito:** Descubrimiento de archivos, búsqueda de código, exploración de bases de código

Claude delega en Explore cuando necesita buscar o entender una base de código sin hacer cambios. Esto mantiene los resultados de exploración fuera del contexto de su conversación principal.

Al invocar Explore, Claude especifica un nivel de minuciosidad: **quick** para búsquedas dirigidas, **medium** para exploración equilibrada, o **very thorough** para análisis exhaustivo.

Plan

Un agente de investigación utilizado durante [plan mode](#) para recopilar contexto antes de presentar un plan.

- **Modelo:** Hereda de la conversación principal
- **Herramientas:** Herramientas de solo lectura (acceso denegado a herramientas Write y Edit)
- **Propósito:** Investigación de base de código para planificación

Cuando está en plan mode y Claude necesita entender su base de código, delega la investigación al subagente Plan. Esto evita el anidamiento infinito (los subagentes no pueden generar otros subagentes) mientras sigue recopilando el contexto necesario.

General-purpose

Un agente capaz para tareas complejas de múltiples pasos que requieren tanto exploración como acción.

- **Modelo:** Hereda de la conversación principal
- **Herramientas:** Todas las herramientas
- **Propósito:** Investigación compleja, operaciones de múltiples pasos, modificaciones de código

Claude delega en general-purpose cuando la tarea requiere tanto exploración como modificación, razonamiento complejo para interpretar resultados, o múltiples pasos dependientes.

Other

Claude Code incluye agentes auxiliares adicionales para tareas específicas. Estos se invocan típicamente automáticamente, por lo que no necesita usarlos directamente.

Agente	Modelo	Cuándo Claude lo usa
Bash	Hereda	Ejecutar comandos de terminal en un contexto separado
statusline-setup	Sonnet	Cuando ejecuta <code>/statusline</code> para configurar su línea de estado
Claude Code Guide	Haiku	Cuando hace preguntas sobre características de Claude Code

Más allá de estos subagentes integrados, puede crear los suyos propios con mensajes personalizados, restricciones de herramientas, modos de permiso, hooks y skills. Las siguientes secciones muestran cómo comenzar y personalizar subagentes.

Inicio rápido: crear su primer subagente

Los subagentes se definen en archivos Markdown con frontmatter YAML. Puede [crearlos manualmente](#) o usar el comando `/agents`.

Este tutorial lo guía a través de la creación de un subagente a nivel de usuario con el comando `/agent`. El subagente revisa código y sugiere mejoras para la base de código.

Step 1: Abrir la interfaz de subagentes

En Claude Code, ejecute:

```
/agents
```

Step 2: Crear un nuevo agente a nivel de usuario

Seleccione **Create new agent**, luego elija **User-level**. Esto guarda el subagente en `~/.claude/agents/` para que esté disponible en todos sus proyectos.

Step 3: Generar con Claude

Seleccione **Generate with Claude**. Cuando se le solicite, describa el subagente:

```
A code improvement agent that scans files and suggests improvements
for readability, performance, and best practices. It should explain
each issue, show the current code, and provide an improved version.
```

Claude genera el mensaje del sistema y la configuración. Presione **e** para abrirlo en su editor si desea personalizarlo.

Step 4: Seleccionar herramientas

Para un revisor de solo lectura, deselectione todo excepto **Read-only tools**. Si mantiene todas las herramientas seleccionadas, el subagente hereda todas las herramientas disponibles para la conversación principal.

Step 5: Seleccionar modelo

Elija qué modelo usa el subagente. Para este agente de ejemplo, seleccione **Sonnet**, que equilibra capacidad y velocidad para analizar patrones de código.

Step 6: Elegir un color

Elija un color de fondo para el subagente. Esto le ayuda a identificar qué subagente se está ejecutando en la interfaz de usuario.

Step 7: Guardar e intentarlo

Guarde el subagente. Está disponible inmediatamente (no se requiere reinicio). Intente:

```
Use the code-improver agent to suggest improvements in this project
```

Claude delega en su nuevo subagente, que escanea la base de código y devuelve sugerencias de mejora.

Ahora tiene un subagente que puede usar en cualquier proyecto en su máquina para analizar bases de código y sugerir mejoras.

También puede crear subagentes manualmente como archivos Markdown, definirlos a través de banderas CLI, o distribuirlos a través de plugins. Las siguientes secciones cubren todas las opciones de configuración.

Configurar subagentes

Usar el comando `/agents`

El comando `/agents` proporciona una interfaz interactiva para administrar subagentes. Ejecute `/agents` para:

- Ver todos los subagentes disponibles (integrados, usuario, proyecto y plugin)
- Crear nuevos subagentes con configuración guiada o generación de Claude
- Editar la configuración de subagentes existentes y el acceso a herramientas
- Eliminar subagentes personalizados
- Ver qué subagentes están activos cuando existen duplicados

Esta es la forma recomendada de crear y administrar subagentes. Para creación manual o automatización, también puede agregar archivos de subagentes directamente.

Para enumerar todos los subagentes configurados desde la línea de comandos sin iniciar una sesión interactiva, ejecute `claude agents`. Esto muestra agentes agrupados por fuente e indica cuáles se anulan por definiciones de mayor prioridad.

Elegir el alcance del subagente

Los subagentes son archivos Markdown con frontmatter YAML. Guárdelos en diferentes ubicaciones según el alcance. Cuando múltiples subagentes comparten el mismo nombre, la ubicación de mayor prioridad gana.

Ubicación	Alcance	Prioridad	Cómo crear
Bandera CLI <code>--agents</code>	Sesión actual	1 (más alta)	Pasar JSON al lanzar Claude Code

Ubicación	Alcance	Prioridad	Cómo crear
<code>.claude/agents/</code>	Proyecto actual	2	Interactivo o manual
<code>~/.claude/agents/</code>	Todos sus proyectos	3	Interactivo o manual
Directorio <code>agents/</code> del plugin	Donde el plugin está habilitado	4 (más baja)	Instalado con plugins

Los subagentes de proyecto (`.claude/agents/`) son ideales para subagentes específicos de una base de código. Verifíquelos en control de versiones para que su equipo pueda usarlos y mejorarlos colaborativamente.

Los subagentes de usuario (`~/.claude/agents/`) son subagentes personales disponibles en todos sus proyectos.

Los subagentes definidos por CLI se pasan como JSON al lanzar Claude Code. Existen solo para esa sesión y no se guardan en disco, lo que los hace útiles para pruebas rápidas o scripts de automatización:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  }
}'
```

La bandera `--agents` acepta JSON con los mismos campos de [frontmatter](#) que los subagentes basados en archivos: `description`, `prompt`, `tools`, `disallowedTools`, `model`, `permissionMode`, `mcpServers`, `hooks`, `maxTurns`, `skills` y `memory`. Use `prompt` para el mensaje del sistema, equivalente al cuerpo markdown en subagentes basados en archivos. Consulte la [referencia CLI](#) para el formato JSON completo.

Los subagentes de **plugin** provienen de [plugins](#) que ha instalado. Aparecen en `/agents` junto a sus subagentes personalizados. Consulte la [referencia de componentes de plugin](#) para obtener detalles sobre la creación de subagentes de plugin.

Escribir archivos de subagentes

Los archivos de subagentes usan frontmatter YAML para configuración, seguido del mensaje del sistema en Markdown:

Note:

Los subagentes se cargan al inicio de la sesión. Si crea un subagente agregando manualmente un archivo, reinicie su sesión o use `/agents` para cargarlo inmediatamente.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
tools: Read, Glob, Grep
model: sonnet
---
```

```
You are a code reviewer. When invoked, analyze the code and provide
specific, actionable feedback on quality, security, and best practices.
```

El frontmatter define los metadatos y la configuración del subagente. El cuerpo se convierte en el mensaje del sistema que guía el comportamiento del subagente. Los subagentes reciben solo este mensaje del sistema (más detalles básicos del entorno como el directorio de trabajo), no el mensaje del sistema completo de Claude Code.

Campos de frontmatter soportados

Los siguientes campos se pueden usar en el frontmatter YAML. Solo `name` y `description` son requeridos.

Campo	Requerido	Descripción
<code>name</code>	Sí	Identificador único usando letras minúsculas y guiones

Campo	Requerido	Descripción
<code>description</code>	Sí	Cuándo Claude debe delegar en este subagente
<code>tools</code>	No	Herramientas que el subagente puede usar. Hereda todas las herramientas si se omite
<code>disallowedTools</code>	No	Herramientas a denegar, eliminadas de la lista heredada o especificada
<code>model</code>	No	Modelo a usar: <code>sonnet</code> , <code>opus</code> , <code>haiku</code> , o <code>inherit</code> . Por defecto <code>inherit</code>
<code>permissionMode</code>	No	Modo de permiso : <code>default</code> , <code>acceptEdits</code> , <code>dontAsk</code> , <code>bypassPermissions</code> , o <code>plan</code>
<code>maxTurns</code>	No	Número máximo de turnos agentivos antes de que el subagente se detenga
<code>skills</code>	No	Skills a cargar en el contexto del subagente al inicio. El contenido completo de la skill se inyecta, no solo se pone disponible para invocación. Los subagentes no heredan skills de la conversación principal

Campo	Requerido	Descripción
<code>mcpServers</code>	No	Servidores MCP disponibles para este subagente. Cada entrada es un nombre de servidor que hace referencia a un servidor ya configurado (por ejemplo, <code>"slack"</code>) o una definición en línea con el nombre del servidor como clave y una configuración completa del servidor MCP como valor
<code>hooks</code>	No	Hooks de ciclo de vida limitados a este subagente
<code>memory</code>	No	Alcance de memoria persistente : <code>user</code> , <code>project</code> , o <code>local</code> . Habilita aprendizaje entre sesiones
<code>background</code>	No	Establecer en <code>true</code> para ejecutar siempre este subagente como una tarea de fondo . Por defecto: <code>false</code>
<code>isolation</code>	No	Establecer en <code>worktree</code> para ejecutar el subagente en un git worktree temporal, dándole una copia aislada del repositorio. El <code>worktree</code> se limpia automáticamente si el subagente no realiza cambios

Elegir un modelo

El campo `model` controla qué [modelo de IA](#) usa el subagente:

- **Alias de modelo:** Use uno de los alias disponibles: `sonnet`, `opus`, o `haiku`
- **inherit:** Use el mismo modelo que la conversación principal

- **Omitido:** Si no se especifica, por defecto es `inherit` (usa el mismo modelo que la conversación principal)

Controlar capacidades del subagente

Puede controlar qué pueden hacer los subagentes a través del acceso a herramientas, modos de permiso y reglas condicionales.

Herramientas disponibles

Los subagentes pueden usar cualquiera de las [herramientas internas](#) de Claude Code. Por defecto, los subagentes heredan todas las herramientas de la conversación principal, incluidas las herramientas MCP.

Para restringir herramientas, use el campo `tools` (lista blanca) o el campo `disallowedTools` (lista negra):

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

Restringir qué subagentes pueden ser generados

Cuando un agente se ejecuta como el hilo principal con `claude --agent`, puede generar subagentes usando la herramienta Agent. Para restringir qué tipos de subagentes puede generar, use la sintaxis `Agent(agent_type)` en el campo `tools`.

Note:

En la versión 2.1.63, la herramienta Task fue renombrada a Agent. Las referencias existentes a `Task(...)` en configuraciones y definiciones de agentes aún funcionan como alias.

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```


Esta es una lista blanca: solo los subagentes `worker` y `researcher` pueden ser generados. Si el agente intenta generar cualquier otro tipo, la solicitud falla y el agente solo ve los tipos permitidos en su mensaje. Para bloquear agentes específicos mientras se permiten todos los demás, use `permissions.deny` en su lugar.

Para permitir generar cualquier subagente sin restricciones, use `Agent` sin paréntesis:

```
tools: Agent, Read, Bash
```

Si `Agent` se omite completamente de la lista `tools`, el agente no puede generar ningún subagente. Esta restricción solo se aplica a agentes que se ejecutan como el hilo principal con `claude --agent`. Los subagentes no pueden generar otros subagentes, por lo que `Agent(agent_type)` no tiene efecto en definiciones de subagentes.

Modos de permiso

El campo `permissionMode` controla cómo el subagente maneja solicitudes de permiso. Los subagentes heredan el contexto de permiso de la conversación principal pero pueden anular el modo.

Modo	Comportamiento
<code>default</code>	Verificación de permiso estándar con solicitudes
<code>acceptEdits</code>	Auto-aceptar ediciones de archivo
<code>dontAsk</code>	Auto-denegar solicitudes de permiso (las herramientas explícitamente permitidas aún funcionan)
<code>bypassPermissions</code>	Omitir todas las verificaciones de permiso
<code>plan</code>	Plan mode (exploración de solo lectura)

Warning:

Use `bypassPermissions` con cuidado. Omite todas las verificaciones de permiso, permitiendo que el subagente ejecute cualquier operación sin aprobación.

Si el principal usa `bypassPermissions`, esto tiene precedencia y no puede ser anulado.

Precargar skills en subagentes

Use el campo `skills` para inyectar contenido de skill en el contexto de un subagente al inicio. Esto le da al subagente conocimiento de dominio sin requerir que descubra y cargue skills durante la ejecución.

```
---
name: api-developer
description: Implement API endpoints following team conventions
skills:
  - api-conventions
  - error-handling-patterns
---

Implement API endpoints. Follow the conventions and patterns from the preloaded skills.
```

El contenido completo de cada skill se inyecta en el contexto del subagente, no solo se pone disponible para invocación. Los subagentes no heredan skills de la conversación principal; debe enumerarlas explícitamente.

Note:

Esto es lo inverso de [ejecutar una skill en un subagente](#). Con `skills` en un subagente, el subagente controla el mensaje del sistema y carga el contenido de la skill. Con `context: fork` en una skill, el contenido de la skill se inyecta en el agente que especifique. Ambos usan el mismo sistema subyacente.

Habilitar memoria persistente

El campo `memory` le da al subagente un directorio persistente que sobrevive entre conversaciones. El subagente usa este directorio para acumular conocimiento a lo largo del tiempo, como patrones de base de código, insights de depuración y decisiones arquitectónicas.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
memory: user
---
```

You are a code reviewer. As you review code, update your agent memory with patterns, conventions, and recurring issues you discover.

Elija un alcance basado en qué tan ampliamente debe aplicarse la memoria:

Alcance	Ubicación	Usar cuando
user	<code>~/.claude/agent-memory/<name-of-agent>/</code>	el subagente debe recordar aprendizajes en todos los proyectos
project	<code>.claude/agent-memory/<name-of-agent>/</code>	el conocimiento del subagente es específico del proyecto y compartible a través de control de versiones
local	<code>.claude/agent-memory-local/<name-of-agent>/</code>	el conocimiento del subagente es específico del proyecto pero no debe ser verificado en control de versiones

Cuando la memoria está habilitada:

- El mensaje del sistema del subagente incluye instrucciones para leer y escribir en el directorio de memoria.
- El mensaje del sistema del subagente también incluye las primeras 200 líneas de `MEMORY.md` en el directorio de memoria, con instrucciones para curar `MEMORY.md` si excede 200 líneas.
- Las herramientas Read, Write y Edit se habilitan automáticamente para que el subagente pueda administrar sus archivos de memoria.

Consejos de memoria persistente

- `user` es el alcance predeterminado recomendado. Use `project` o `local` cuando el conocimiento del subagente es solo relevante para una base de código específica.

- Pida al subagente que consulte su memoria antes de comenzar el trabajo: “Review this PR, and check your memory for patterns you’ve seen before.”
- Pida al subagente que actualice su memoria después de completar una tarea: “Now that you’re done, save what you learned to your memory.” Con el tiempo, esto construye una base de conocimiento que hace que el subagente sea más efectivo.
- Incluya instrucciones de memoria directamente en el archivo markdown del subagente para que mantenga proactivamente su propia base de conocimiento:

Update your agent memory as you discover codepaths, patterns, library locations, and key architectural decisions. This builds up institutional knowledge across conversations. Write concise notes about what you found and where.

Reglas condicionales con hooks

Para un control más dinámico sobre el uso de herramientas, use hooks `PreToolUse` para validar operaciones antes de que se ejecuten. Esto es útil cuando necesita permitir algunas operaciones de una herramienta mientras bloquea otras.

Este ejemplo crea un subagente que solo permite consultas de base de datos de solo lectura. El hook `PreToolUse` ejecuta el script especificado en `command` antes de que se ejecute cada comando Bash:

```
---
name: db-reader
description: Execute read-only database queries
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---
```

Claude Code [pasa la entrada del hook como JSON](#) a través de stdin a comandos de hook. El script de validación lee este JSON, extrae el comando Bash y [sale con código 2](#) para bloquear operaciones de escritura:

```
#!/bin/bash
## ./scripts/validate-readonly-query.sh

INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

## Block SQL write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE)
\b' > /dev/null; then
    echo "Blocked: Only SELECT queries are allowed" >&2
    exit 2
fi

exit 0
```

Consulte [Hook input](#) para el esquema de entrada completo y [códigos de salida](#) para cómo los códigos de salida afectan el comportamiento.

Deshabilitar subagentes específicos

Puede evitar que Claude use subagentes específicos agregándolos a la matriz `deny` en su [configuración](#). Use el formato `Agent(subagent-name)` donde `subagent-name` coincide con el campo `name` del subagente.

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

Esto funciona tanto para subagentes integrados como personalizados. También puede usar la bandera CLI `--disallowedTools` :

```
claude --disallowedTools "Agent(Explore)"
```

Consulte la [documentación de Permisos](#) para más detalles sobre reglas de permiso.

Definir hooks para subagentes

Los subagentes pueden definir [hooks](#) que se ejecutan durante el ciclo de vida del subagente. Hay dos formas de configurar hooks:

1. **En el frontmatter del subagente:** Defina hooks que se ejecuten solo mientras ese subagente está activo
2. **En `settings.json` :** Defina hooks que se ejecuten en la sesión principal cuando los subagentes comienzan o se detienen

Hooks en frontmatter de subagentes

Defina hooks directamente en el archivo markdown del subagente. Estos hooks solo se ejecutan mientras ese subagente específico está activo y se limpian cuando termina.

Todos los [eventos de hook](#) son soportados. Los eventos más comunes para subagentes son:

Evento	Entrada del matcher	Cuándo se dispara
<code>PreToolUse</code>	Nombre de herramienta	Antes de que el subagente use una herramienta
<code>PostToolUse</code>	Nombre de herramienta	Después de que el subagente usa una herramienta
<code>Stop</code>	(ninguno)	Cuando el subagente termina (convertido a <code>SubagentStop</code> en tiempo de ejecución)

Este ejemplo valida comandos Bash con el hook `PreToolUse` y ejecuta un linter después de ediciones de archivo con `PostToolUse` :

```
---
name: code-reviewer
description: Review code changes with automatic linting
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-command.sh $TOOL_INPUT"
  PostToolUse:
    - matcher: "Edit|Write"
      hooks:
        - type: command
          command: "./scripts/run-linter.sh"
---
```

Los hooks `Stop` en frontmatter se convierten automáticamente a eventos `SubagentStop`.

Hooks a nivel de proyecto para eventos de subagentes

Configure hooks en `settings.json` que respondan a eventos de ciclo de vida de subagentes en la sesión principal.

Evento	Entrada del matcher	Cuándo se dispara
<code>SubagentStart</code>	Nombre de tipo de agente	Cuando un subagente comienza la ejecución
<code>SubagentStop</code>	Nombre de tipo de agente	Cuando un subagente se completa

Ambos eventos soportan matchers para dirigirse a tipos de agentes específicos por nombre. Este ejemplo ejecuta un script de configuración solo cuando el subagente `db-agent` comienza, y un script de limpieza cuando cualquier subagente se detiene:

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

Consulte [Hooks](#) para el formato de configuración de hook completo.

Trabajar con subagentes

Entender la delegación automática

Claude delega automáticamente tareas basadas en la descripción de la tarea en su solicitud, el campo `description` en configuraciones de subagentes y el contexto actual. Para fomentar la delegación proactiva, incluya frases como “use proactively” en el campo `description` de su subagente.

También puede solicitar un subagente específico explícitamente:

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```


Ejecutar subagentes en primer plano o fondo

Los subagentes pueden ejecutarse en primer plano (bloqueante) o fondo (concurrente):

- **Los subagentes en primer plano** bloquean la conversación principal hasta completarse. Las solicitudes de permiso y preguntas aclaratorias (como `AskUserQuestion`) se le pasan a usted.
- **Los subagentes en fondo** se ejecutan concurrentemente mientras continúa trabajando. Antes de lanzar, Claude Code solicita cualquier permiso de herramienta que el subagente necesitará, asegurando que tenga las aprobaciones necesarias por adelantado. Una vez en ejecución, el subagente hereda estos permisos y auto-deniega cualquier cosa no preaprobada. Si un subagente en fondo necesita hacer preguntas aclaratorias, esa llamada de herramienta falla pero el subagente continúa.

Si un subagente en fondo falla debido a permisos faltantes, puede [reanudarlo](#) en primer plano para reintentar con solicitudes interactivas.

Claude decide si ejecutar subagentes en primer plano o fondo basado en la tarea. También puede:

- Pedir a Claude que “run this in the background”
- Presionar **Ctrl+B** para poner una tarea en ejecución en fondo

Para deshabilitar toda la funcionalidad de tarea de fondo, establezca la variable de entorno `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` en `1`. Consulte [Variables de entorno](#).

Patrones comunes

Aislar operaciones de alto volumen

Uno de los usos más efectivos para subagentes es aislar operaciones que producen grandes cantidades de salida. Ejecutar pruebas, obtener documentación o procesar archivos de registro puede consumir contexto significativo. Al delegar estos a un subagente, la salida detallada permanece en el contexto del subagente mientras solo el resumen relevante regresa a su conversación principal.

```
Use a subagent to run the test suite and report only the failing tests with their error messages
```

Ejecutar investigación en paralelo

Para investigaciones independientes, genere múltiples subagentes para trabajar simultáneamente:

Research the authentication, database, and API modules in parallel using separate subagents

Cada subagente explora su área de forma independiente, luego Claude sintetiza los hallazgos. Esto funciona mejor cuando las rutas de investigación no dependen una de la otra.

Warning:

Cuando los subagentes se completan, sus resultados regresan a su conversación principal. Ejecutar muchos subagentes que cada uno devuelve resultados detallados puede consumir contexto significativo.

Para tareas que necesitan paralelismo sostenido o exceden su ventana de contexto, [equipos de agentes](#) le dan a cada trabajador su propio contexto independiente.

Encadenar subagentes

Para flujos de trabajo de múltiples pasos, pida a Claude que use subagentes en secuencia. Cada subagente completa su tarea y devuelve resultados a Claude, que luego pasa contexto relevante al siguiente subagente.

Use the code-reviewer subagent to find performance issues, then use the optimizer subagent to fix them

Elegir entre subagentes y conversación principal

Use la **conversación principal** cuando:

- La tarea necesita ida y vuelta frecuente o refinamiento iterativo
- Múltiples fases comparten contexto significativo (planificación → implementación → prueba)
- Está haciendo un cambio rápido y dirigido
- La latencia importa. Los subagentes comienzan de nuevo y pueden necesitar tiempo para recopilar contexto

Use **subagentes** cuando:

- La tarea produce salida detallada que no necesita en su contexto principal
- Desea aplicar restricciones de herramientas o permisos específicos
- El trabajo es autónomo y puede devolver un resumen

Considere [Skills](#) en su lugar cuando desee mensajes reutilizables o flujos de trabajo que se ejecuten en el contexto de la conversación principal en lugar de contexto de subagente aislado.

Para una pregunta rápida sobre algo ya en su conversación, use `/btw` en lugar de un subagente. Ve su contexto completo pero no tiene acceso a herramientas, y la respuesta se descarta en lugar de agregarse al historial.

Note:

Los subagentes no pueden generar otros subagentes. Si su flujo de trabajo requiere delegación anidada, use [Skills](#) o [encadene subagentes](#) desde la conversación principal.

Administrar contexto de subagentes

Reanudar subagentes

Cada invocación de subagente crea una nueva instancia con contexto fresco. Para continuar el trabajo de un subagente existente en lugar de comenzar de nuevo, pida a Claude que lo reanude.

Los subagentes reanudados retienen su historial de conversación completo, incluidas todas las llamadas de herramienta anteriores, resultados y razonamiento. El subagente continúa exactamente donde se detuvo en lugar de comenzar de nuevo.

Cuando un subagente se completa, Claude recibe su ID de agente. Para reanudar un subagente, pida a Claude que continúe el trabajo anterior:

```
Use the code-reviewer subagent to review the authentication module
[Agent completes]

Continue that code review and now analyze the authorization logic
[Claude resumes the subagent with full context from previous conversation]
```

También puede pedir a Claude el ID del agente si desea referenciarlo explícitamente, o encontrar IDs en los archivos de transcripción en `~/.claude/projects/{project}/{sessionId}/subagents/`. Cada transcripción se almacena como `agent-{agentId}.jsonl`.

Las transcripciones de subagentes persisten independientemente de la conversación principal:

- **Compactación de conversación principal:** Cuando la conversación principal se compacta, las transcripciones de subagentes no se ven afectadas. Se almacenan en archivos separados.
- **Persistencia de sesión:** Las transcripciones de subagentes persisten dentro de su sesión. Puede [reanudar un subagente](#) después de reiniciar Claude Code reanudando la misma sesión.
- **Limpieza automática:** Las transcripciones se limpian basadas en la configuración `cleanupPeriodDays` (por defecto: 30 días).

Auto-compactación

Los subagentes soportan compactación automática usando la misma lógica que la conversación principal. Por defecto, la compactación automática se dispara aproximadamente al 95% de capacidad. Para disparar la compactación antes, establezca

`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` en un porcentaje más bajo (por ejemplo, `50`). Consulte [variables de entorno](#) para detalles.

Los eventos de compactación se registran en archivos de transcripción de subagentes:

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

El valor `preTokens` muestra cuántos tokens se usaron antes de que ocurriera la compactación.

Subagentes de ejemplo

Estos ejemplos demuestran patrones efectivos para construir subagentes. Úselos como puntos de partida, o genere una versión personalizada con Claude.

Tip:

Mejores prácticas:

- **Diseñe subagentes enfocados:** cada subagente debe sobresalir en una tarea específica
- **Escriba descripciones detalladas:** Claude usa la descripción para decidir cuándo delegar
- **Limite el acceso a herramientas:** otorgue solo permisos necesarios para seguridad y enfoque
- **Verifique en control de versiones:** comparta subagentes de proyecto con su equipo

Revisor de código

Un subagente de solo lectura que revisa código sin modificarlo. Este ejemplo muestra cómo diseñar un subagente enfocado con acceso limitado a herramientas (sin herramientas Edit o Write) y un mensaje detallado que especifica exactamente qué buscar y cómo formatear la salida.

```
---  
name: code-reviewer  
description: Expert code review specialist. Proactively reviews code for quality,  
security, and maintainability. Use immediately after writing or modifying code.  
tools: Read, Grep, Glob, Bash  
model: inherit  
---
```

You are a senior code reviewer ensuring high standards of code quality and security.

When invoked:

1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:

- Code is clear and readable
- Functions and variables are well-named
- No duplicated code
- Proper error handling
- No exposed secrets or API keys
- Input validation implemented
- Good test coverage
- Performance considerations addressed

Provide feedback organized by priority:

- Critical issues (must fix)
- Warnings (should fix)
- Suggestions (consider improving)

Include specific examples of how to fix issues.

Depurador

Un subagente que puede tanto analizar como corregir problemas. A diferencia del revisor de código, este incluye Edit porque corregir errores requiere modificar código. El mensaje proporciona un flujo de trabajo claro desde diagnóstico hasta verificación.

```
---  
name: debugger  
description: Debugging specialist for errors, test failures, and unexpected behavior. Use proactively when encountering any issues.  
tools: Read, Edit, Bash, Grep, Glob  
---
```

You are an expert debugger specializing in root cause analysis.

When invoked:

1. Capture error message and stack trace
2. Identify reproduction steps
3. Isolate the failure location
4. Implement minimal fix
5. Verify solution works

Debugging process:

- Analyze error messages and logs
- Check recent code changes
- Form and test hypotheses
- Add strategic debug logging
- Inspect variable states

For each issue, provide:

- Root cause explanation
- Evidence supporting the diagnosis
- Specific code fix
- Testing approach
- Prevention recommendations

Focus on fixing the underlying issue, not the symptoms.

Científico de datos

Un subagente específico de dominio para trabajo de análisis de datos. Este ejemplo muestra cómo crear subagentes para flujos de trabajo especializados fuera de tareas de codificación típicas. Establece explícitamente `model: sonnet` para análisis más capaz.

```
---  
name: data-scientist  
description: Data analysis expert for SQL queries, BigQuery operations, and data  
insights. Use proactively for data analysis tasks and queries.  
tools: Bash, Read, Write  
model: sonnet  
---
```

You are a data scientist specializing in SQL and BigQuery analysis.

When invoked:

1. Understand the data analysis requirement
2. Write efficient SQL queries
3. Use BigQuery command line tools (bq) when appropriate
4. Analyze and summarize results
5. Present findings clearly

Key practices:

- Write optimized SQL queries with proper filters
- Use appropriate aggregations and joins
- Include comments explaining complex logic
- Format results for readability
- Provide data-driven recommendations

For each analysis:

- Explain the query approach
- Document any assumptions
- Highlight key findings
- Suggest next steps based on data

Always ensure queries are efficient and cost-effective.

Validador de consultas de base de datos

Un subagente que permite acceso a Bash pero valida comandos para permitir solo consultas SQL de solo lectura. Este ejemplo muestra cómo usar hooks `PreToolUse` para validación condicional cuando necesita control más fino que el campo `tools` proporciona.


```

---
name: db-reader
description: Execute read-only database queries. Use when analyzing data or
generating reports.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---

```

You are a database analyst with read-only access. Execute SELECT queries to answer questions about the data.

When asked to analyze data:

1. Identify which tables contain the relevant data
2. Write efficient SELECT queries with appropriate filters
3. Present results clearly with context

You cannot modify data. If asked to INSERT, UPDATE, DELETE, or modify schema, explain that you only have read access.

Claude Code [pasa la entrada del hook como JSON](#) a través de stdin a comandos de hook. El script de validación lee este JSON, extrae el comando que se está ejecutando y lo verifica contra una lista de operaciones de escritura SQL. Si se detecta una operación de escritura, el script [sale con código 2](#) para bloquear la ejecución y devuelve un mensaje de error a Claude a través de stderr.

Cree el script de validación en cualquier lugar de su proyecto. La ruta debe coincidir con el campo `command` en su configuración de hook:

```
#!/bin/bash
## Blocks SQL write operations, allows SELECT queries

## Read JSON input from stdin
INPUT=$(cat)

## Extract the command field from tool_input using jq
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

if [ -z "$COMMAND" ]; then
    exit 0
fi

## Block write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE|REPLACE|MERGE)\b' > /dev/null; then
    echo "Blocked: Write operations not allowed. Use SELECT queries only." >&2
    exit 2
fi

exit 0
```

Haga el script ejecutable:

```
chmod +x ./scripts/validate-readonly-query.sh
```

El hook recibe JSON a través de stdin con el comando Bash en `tool_input.command`. El código de salida 2 bloquea la operación y alimenta el mensaje de error de vuelta a Claude. Consulte [Hooks](#) para detalles sobre códigos de salida y [Hook input](#) para el esquema de entrada completo.

Próximos pasos

Ahora que entiende los subagentes, explore estas características relacionadas:

- [Distribuir subagentes con plugins](#) para compartir subagentes entre equipos o proyectos

- [Ejecutar Claude Code programáticamente](#) con el Agent SDK para CI/CD y automatización
- [Usar servidores MCP](#) para dar a los subagentes acceso a herramientas y datos externos

Orquestar equipos de sesiones de Claude Code

Coordine múltiples instancias de Claude Code trabajando juntas como un equipo, con tareas compartidas, mensajería entre agentes y gestión centralizada.

Warning:

Los equipos de agentes son experimentales y están deshabilitados por defecto. Habilítelos agregando `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` a su [settings.json](#) o entorno. Los equipos de agentes tienen [limitaciones conocidas](#) alrededor de la reanudación de sesiones, coordinación de tareas y comportamiento de apagado.

Los equipos de agentes le permiten coordinar múltiples instancias de Claude Code trabajando juntas. Una sesión actúa como el líder del equipo, coordinando el trabajo, asignando tareas y sintetizando resultados. Los compañeros de equipo trabajan de forma independiente, cada uno en su propia ventana de contexto, y se comunican directamente entre sí.

A diferencia de los [subagents](#), que se ejecutan dentro de una única sesión y solo pueden reportar al agente principal, también puede interactuar directamente con compañeros de equipo individuales sin pasar por el líder.

Note:

Los equipos de agentes requieren Claude Code v2.1.32 o posterior. Verifique su versión con `claude --version`.

Esta página cubre:

- [Cuándo usar equipos de agentes](#), incluyendo los mejores casos de uso y cómo se comparan con los subagents
- [Iniciando un equipo](#)
- [Controlando compañeros de equipo](#), incluyendo modos de visualización, asignación de tareas y delegación
- [Mejores prácticas para trabajo paralelo](#)

Cuándo usar equipos de agentes

Los equipos de agentes son más efectivos para tareas donde la exploración paralela agrega valor real. Vea [ejemplos de casos de uso](#) para escenarios completos. Los casos de uso más sólidos son:

- **Investigación y revisión:** múltiples compañeros de equipo pueden investigar diferentes aspectos de un problema simultáneamente, luego compartir y desafiar los hallazgos de los demás
- **Nuevos módulos o características:** los compañeros de equipo pueden poseer cada una una pieza separada sin pisarse mutuamente
- **Depuración con hipótesis competidoras:** los compañeros de equipo prueban diferentes teorías en paralelo y convergen en la respuesta más rápidamente
- **Coordinación entre capas:** cambios que abarcan frontend, backend y pruebas, cada uno propiedad de un compañero de equipo diferente

Los equipos de agentes agregan sobrecarga de coordinación y usan significativamente más tokens que una única sesión. Funcionan mejor cuando los compañeros de equipo pueden operar de forma independiente. Para tareas secuenciales, ediciones del mismo archivo o trabajo con muchas dependencias, una única sesión o [subagents](#) son más efectivos.

Comparar con subagents

Tanto los equipos de agentes como los [subagents](#) le permiten paralelizar el trabajo, pero operan de manera diferente. Elija según si sus trabajadores necesitan comunicarse entre sí:

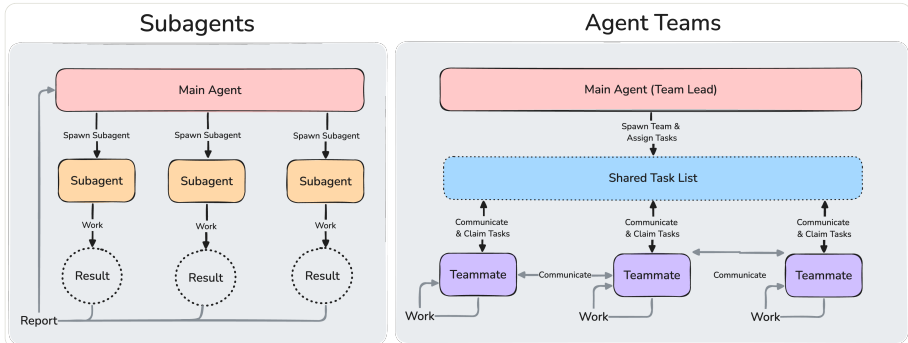


Diagrama comparando arquitecturas de subagent y equipo de agentes. Los subagents son generados por el agente principal, hacen trabajo y reportan resultados. Los equipos de agentes se coordinan a través de una lista de tareas compartida, con compañeros de equipo comunicándose directamente entre sí.

	Subagents	Equipos de agentes
Contexto	Ventana de contexto propia; los resultados regresan al llamador	Ventana de contexto propia; completamente independiente
Comunicación	Reportan resultados solo al agente principal	Los compañeros de equipo se envían mensajes directamente
Coordinación	El agente principal gestiona todo el trabajo	Lista de tareas compartida con auto-coordinación
Mejor para	Tareas enfocadas donde solo importa el resultado	Trabajo complejo que requiere discusión y colaboración
Costo de tokens	Menor: resultados resumidos de vuelta al contexto principal	Mayor: cada compañero de equipo es una instancia Claude separada

Use subagents cuando necesite trabajadores rápidos y enfocados que reporten. Use equipos de agentes cuando los compañeros de equipo necesiten compartir hallazgos, desafiarse mutuamente y coordinarse por su cuenta.

Habilitar equipos de agentes

Los equipos de agentes están deshabilitados por defecto. Habilítelos configurando la variable de entorno `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` a `1`, ya sea en su entorno de shell o a través de [settings.json](#):

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

Inicie su primer equipo de agentes

Después de habilitar los equipos de agentes, dígame a Claude que cree un equipo de agentes y describa la tarea y la estructura del equipo que desea en lenguaje natural. Claude crea el equipo, genera compañeros de equipo y coordina el trabajo según su indicación.

Este ejemplo funciona bien porque los tres roles son independientes y pueden explorar el problema sin esperar el uno al otro:

Estoy diseñando una herramienta CLI que ayuda a los desarrolladores a rastrear comentarios TODO en su base de código. Crea un equipo de agentes para explorar esto desde diferentes ángulos: un compañero de equipo en UX, uno en arquitectura técnica, uno jugando al abogado del diablo.

A partir de ahí, Claude crea un equipo con una [lista de tareas compartida](#), genera compañeros de equipo para cada perspectiva, los hace explorar el problema, sintetiza hallazgos e intenta [limpiar el equipo](#) cuando termina.

La terminal del líder enumera todos los compañeros de equipo y en qué están trabajando. Use Shift+Down para ciclar a través de compañeros de equipo y enviarles un mensaje directamente. Después del último compañero de equipo, Shift+Down vuelve al líder.

Si desea que cada compañero de equipo esté en su propio panel dividido, vea [Elegir un modo de visualización](#).

Controle su equipo de agentes

Dígale al líder lo que desea en lenguaje natural. Maneja la coordinación del equipo, asignación de tareas y delegación según sus instrucciones.

Elegir un modo de visualización

Los equipos de agentes admiten dos modos de visualización:

- **En proceso:** todos los compañeros de equipo se ejecutan dentro de su terminal principal. Use Shift+Down para ciclar a través de compañeros de equipo y escriba para enviarles un mensaje directamente. Funciona en cualquier terminal, sin configuración adicional requerida.
- **Paneles divididos:** cada compañero de equipo obtiene su propio panel. Puede ver la salida de todos a la vez y hacer clic en un panel para interactuar directamente. Requiere tmux o iTerm2.

Note:

`tmux` tiene limitaciones conocidas en ciertos sistemas operativos y tradicionalmente funciona mejor en macOS. Usar `tmux -CC` en iTerm2 es el punto de entrada sugerido en `tmux`.

El valor predeterminado es `"auto"`, que usa paneles divididos si ya está ejecutándose dentro de una sesión `tmux`, y en proceso de lo contrario. La configuración `"tmux"` habilita el modo de panel dividido y detecta automáticamente si usar `tmux` o `iTerm2` según su terminal. Para anular, configure `teammateMode` en su [settings.json](#):

```
{
  "teammateMode": "in-process"
}
```

Para forzar el modo en proceso para una única sesión, páselo como una bandera:

```
claude --teammate-mode in-process
```

El modo de panel dividido requiere `tmux` o `iTerm2` con la [CLI it2](#). Para instalar manualmente:

- **tmux:** instale a través del gestor de paquetes de su sistema. Vea la [wiki de tmux](#) para instrucciones específicas de la plataforma.
- **iTerm2:** instale la [CLI it2](#), luego habilite la API de Python en **iTerm2** → **Settings** → **General** → **Magic** → **Enable Python API**.

Especificar compañeros de equipo y modelos

Claude decide el número de compañeros de equipo a generar según su tarea, o puede especificar exactamente lo que desea:

```
Crea un equipo con 4 compañeros de equipo para refactorizar estos módulos en
paralelo.
Usa Sonnet para cada compañero de equipo.
```

Requerir aprobación de plan para compañeros de equipo

Para tareas complejas o riesgosas, puede requerir que los compañeros de equipo planifiquen antes de implementar. El compañero de equipo trabaja en modo de plan de solo lectura hasta que el líder apruebe su enfoque:

Genera un compañero de equipo arquitecto para refactorizar el módulo de autenticación.

Requiere aprobación de plan antes de que hagan cambios.

Cuando un compañero de equipo termina de planificar, envía una solicitud de aprobación de plan al líder. El líder revisa el plan y lo aprueba o lo rechaza con retroalimentación. Si se rechaza, el compañero de equipo permanece en modo de plan, revisa según la retroalimentación y reenvía. Una vez aprobado, el compañero de equipo sale del modo de plan y comienza la implementación.

El líder toma decisiones de aprobación de forma autónoma. Para influir en el juicio del líder, proporcione criterios en su indicación, como “solo aprueba planes que incluyan cobertura de pruebas” o “rechaza planes que modifiquen el esquema de la base de datos”.

Hable directamente con los compañeros de equipo

Cada compañero de equipo es una sesión completa e independiente de Claude Code. Puede enviar un mensaje a cualquier compañero de equipo directamente para dar instrucciones adicionales, hacer preguntas de seguimiento o redirigir su enfoque.

- **Modo en proceso:** use Shift+Down para ciclar a través de compañeros de equipo, luego escriba para enviarles un mensaje. Presione Enter para ver la sesión de un compañero de equipo, luego Escape para interrumpir su turno actual. Presione Ctrl+T para alternar la lista de tareas.
- **Modo de panel dividido:** haga clic en el panel de un compañero de equipo para interactuar directamente con su sesión. Cada compañero de equipo tiene una vista completa de su propio terminal.

Asignar y reclamar tareas

La lista de tareas compartida coordina el trabajo en todo el equipo. El líder crea tareas y los compañeros de equipo las trabajan. Las tareas tienen tres estados: pendiente, en progreso y completada. Las tareas también pueden depender de otras tareas: una tarea pendiente con dependencias sin resolver no puede ser reclamada hasta que esas dependencias se completen.

El líder puede asignar tareas explícitamente, o los compañeros de equipo pueden auto-reclamar:

- **El líder asigna:** dígame al líder qué tarea dar a qué compañero de equipo
- **Auto-reclamar:** después de terminar una tarea, un compañero de equipo recoge la siguiente tarea sin asignar y sin bloquear por su cuenta

El reclamo de tareas usa bloqueo de archivos para prevenir condiciones de carrera cuando múltiples compañeros de equipo intentan reclamar la misma tarea simultáneamente.

Apagar compañeros de equipo

Para terminar gracefully la sesión de un compañero de equipo:

```
Pídele al compañero de equipo investigador que se apague
```

El líder envía una solicitud de apagado. El compañero de equipo puede aprobar, saliendo gracefully, o rechazar con una explicación.

Limpiar el equipo

Cuando haya terminado, pídele al líder que limpie:

```
Limpia el equipo
```

Esto elimina los recursos compartidos del equipo. Cuando el líder ejecuta la limpieza, verifica si hay compañeros de equipo activos y falla si alguno aún se está ejecutando, así que apáguelos primero.

Warning:

Siempre use el líder para limpiar. Los compañeros de equipo no deben ejecutar la limpieza porque su contexto de equipo puede no resolverse correctamente, dejando potencialmente recursos en un estado inconsistente.

Aplicar puertas de calidad con hooks

Use [hooks](#) para aplicar reglas cuando los compañeros de equipo terminen el trabajo o las tareas se completen:

- `TeammateIdle` : se ejecuta cuando un compañero de equipo está a punto de quedarse inactivo. Salga con código 2 para enviar retroalimentación y mantener al compañero de equipo trabajando.
- `TaskCompleted` : se ejecuta cuando una tarea está siendo marcada como completada. Salga con código 2 para prevenir la finalización y enviar retroalimentación.

Cómo funcionan los equipos de agentes

Esta sección cubre la arquitectura y la mecánica detrás de los equipos de agentes. Si desea comenzar a usarlos, vea [Controle su equipo de agentes](#) arriba.

Cómo Claude inicia equipos de agentes

Hay dos formas en que los equipos de agentes se inician:

- **Usted solicita un equipo:** dé a Claude una tarea que se beneficie del trabajo paralelo y solicite explícitamente un equipo de agentes. Claude crea uno según sus instrucciones.
- **Claude propone un equipo:** si Claude determina que su tarea se beneficiaría del trabajo paralelo, puede sugerir crear un equipo. Usted confirma antes de que proceda.

En ambos casos, usted mantiene el control. Claude no creará un equipo sin su aprobación.

Arquitectura

Un equipo de agentes consiste en:

Componente	Rol
Líder del equipo	La sesión principal de Claude Code que crea el equipo, genera compañeros de equipo y coordina el trabajo
Compañeros de equipo	Instancias separadas de Claude Code que cada una trabaja en tareas asignadas
Lista de tareas	Lista compartida de elementos de trabajo que los compañeros de equipo reclaman y completan
Buzón	Sistema de mensajería para comunicación entre agentes

Vea [Elegir un modo de visualización](#) para opciones de configuración de visualización. Los mensajes de los compañeros de equipo llegan al líder automáticamente.

El sistema gestiona las dependencias de tareas automáticamente. Cuando un compañero de equipo completa una tarea de la que otras tareas dependen, las tareas bloqueadas se desbloquean sin intervención manual.

Los equipos y tareas se almacenan localmente:

- **Configuración del equipo:** `~/.claude/teams/{team-name}/config.json`
- **Lista de tareas:** `~/.claude/tasks/{team-name}/`

La configuración del equipo contiene un array `members` con el nombre de cada compañero de equipo, ID de agente y tipo de agente. Los compañeros de equipo pueden leer este archivo para descubrir otros miembros del equipo.

Permisos

Los compañeros de equipo comienzan con la configuración de permisos del líder. Si el líder se ejecuta con `--dangerously-skip-permissions`, todos los compañeros de equipo también lo hacen. Después de generar, puede cambiar los modos de compañeros de equipo individuales, pero no puede establecer modos por compañero de equipo en el momento de la generación.

Contexto y comunicación

Cada compañero de equipo tiene su propia ventana de contexto. Cuando se genera, un compañero de equipo carga el mismo contexto de proyecto que una sesión regular: CLAUDE.md, MCP servers y skills. También recibe la indicación de generación del líder. El historial de conversación del líder no se transfiere.

Cómo los compañeros de equipo comparten información:

- **Entrega automática de mensajes:** cuando los compañeros de equipo envían mensajes, se entregan automáticamente a los destinatarios. El líder no necesita sondear actualizaciones.
- **Notificaciones de inactividad:** cuando un compañero de equipo termina y se detiene, notifica automáticamente al líder.
- **Lista de tareas compartida:** todos los agentes pueden ver el estado de la tarea y reclamar trabajo disponible.

Mensajería de compañeros de equipo:

- **message:** enviar un mensaje a un compañero de equipo específico
- **broadcast:** enviar a todos los compañeros de equipo simultáneamente. Use con moderación, ya que los costos escalan con el tamaño del equipo.

Uso de tokens

Los equipos de agentes usan significativamente más tokens que una única sesión. Cada compañero de equipo tiene su propia ventana de contexto, y el uso de tokens escala con el número de compañeros de equipo activos. Para investigación, revisión y trabajo de nuevas características, los tokens adicionales generalmente valen la pena. Para tareas rutinarias, una única sesión es más rentable. Vea [costos de tokens de equipos de agentes](#) para orientación de uso.

Ejemplos de casos de uso

Estos ejemplos muestran cómo los equipos de agentes manejan tareas donde la exploración paralela agrega valor.

Ejecutar una revisión de código paralela

Un único revisor tiende a gravitar hacia un tipo de problema a la vez. Dividir criterios de revisión en dominios independientes significa que la seguridad, el rendimiento y la cobertura de pruebas reciben atención exhaustiva simultáneamente. La indicación asigna a cada compañero de equipo una lente distinta para que no se superpongan:

```
Crea un equipo de agentes para revisar la PR #142. Genera tres revisores:  
- Uno enfocado en implicaciones de seguridad  
- Uno verificando impacto de rendimiento  
- Uno validando cobertura de pruebas  
Que cada uno revise e informe hallazgos.
```

Cada revisor trabaja desde la misma PR pero aplica un filtro diferente. El líder sintetiza hallazgos en los tres después de que terminen.

Investigar con hipótesis competidoras

Cuando la causa raíz es poco clara, un único agente tiende a encontrar una explicación plausible y dejar de buscar. La indicación lucha contra esto haciendo que los compañeros de equipo sean explícitamente adversarios: el trabajo de cada uno no es solo investigar su propia teoría sino desafiar las de los demás.

```
Los usuarios reportan que la aplicación se cierra después de un mensaje en lugar  
de  
mantenerse conectada. Genera 5 compañeros de equipo de agentes para investigar  
diferentes hipótesis. Haz que se hablen entre sí para intentar refutar las teorías  
de los demás, como un debate científico. Actualiza el documento de hallazgos con  
cualquier consenso que emerja.
```

La estructura de debate es el mecanismo clave aquí. La investigación secuencial sufre de anclaje: una vez que se explora una teoría, la investigación posterior está sesgada hacia ella.

Con múltiples investigadores independientes intentando activamente refutar mutuamente, la teoría que sobrevive es mucho más probable que sea la causa raíz real.

Mejores prácticas

Dé a los compañeros de equipo suficiente contexto

Los compañeros de equipo cargan contexto de proyecto automáticamente, incluyendo CLAUDE.md, MCP servers y skills, pero no heredan el historial de conversación del líder. Vea [Contexto y comunicación](#) para detalles. Incluya detalles específicos de la tarea en la indicación de generación:

```
Genera un compañero de equipo revisor de seguridad con la indicación: "Revisa el
módulo
de autenticación en src/auth/ para vulnerabilidades de seguridad. Enfócate en
manejo
de tokens, gestión de sesiones y validación de entrada. La aplicación usa tokens
JWT
almacenados en cookies httpOnly. Reporta cualquier problema con calificaciones de
severidad."
```

Elegir un tamaño de equipo apropiado

No hay límite duro en el número de compañeros de equipo, pero se aplican restricciones prácticas:

- **Los costos de tokens escalan linealmente:** cada compañero de equipo tiene su propia ventana de contexto y consume tokens independientemente. Vea [costos de tokens de equipos de agentes](#) para detalles.
- **La sobrecarga de coordinación aumenta:** más compañeros de equipo significa más comunicación, coordinación de tareas y potencial para conflictos
- **Rendimientos decrecientes:** más allá de cierto punto, compañeros de equipo adicionales no aceleran el trabajo proporcionalmente

Comience con 3-5 compañeros de equipo para la mayoría de flujos de trabajo. Esto equilibra el trabajo paralelo con coordinación manejable. Los ejemplos en esta guía usan 3-5 compañeros de equipo porque ese rango funciona bien en diferentes tipos de tareas.

Tener 5-6 [tareas](#) por compañero de equipo mantiene a todos productivos sin cambio de contexto excesivo. Si tiene 15 tareas independientes, 3 compañeros de equipo es un buen punto de partida.

Escale solo cuando el trabajo genuinamente se beneficie de tener compañeros de equipo trabajando simultáneamente. Tres compañeros de equipo enfocados a menudo superan a cinco dispersos.

Dimensionar tareas apropiadamente

- **Demasiado pequeñas:** la sobrecarga de coordinación excede el beneficio
- **Demasiado grandes:** los compañeros de equipo trabajan demasiado tiempo sin check-ins, aumentando el riesgo de esfuerzo desperdiciado
- **Justo bien:** unidades auto-contenidas que producen un entregable claro, como una función, un archivo de prueba o una revisión

Tip:

El líder divide el trabajo en tareas y las asigna a los compañeros de equipo automáticamente. Si no está creando suficientes tareas, pídele que divida el trabajo en piezas más pequeñas. Tener 5-6 tareas por compañero de equipo mantiene a todos productivos y permite al líder reasignar trabajo si alguien se queda atrapado.

Espere a que los compañeros de equipo terminen

A veces el líder comienza a implementar tareas por sí mismo en lugar de esperar a los compañeros de equipo. Si nota esto:

Espera a que tus compañeros de equipo completen sus tareas antes de proceder

Comience con investigación y revisión

Si es nuevo en equipos de agentes, comience con tareas que tengan límites claros y no requieran escribir código: revisar una PR, investigar una biblioteca o investigar un error. Estas tareas muestran el valor de la exploración paralela sin los desafíos de coordinación que vienen con la implementación paralela.

Evitar conflictos de archivos

Dos compañeros de equipo editando el mismo archivo lleva a sobrescrituras. Divida el trabajo para que cada compañero de equipo posea un conjunto diferente de archivos.

Monitorear y dirigir

Verifique el progreso de los compañeros de equipo, redirija enfoques que no estén funcionando y sintetice hallazgos a medida que lleguen. Dejar que un equipo se ejecute desatendido durante demasiado tiempo aumenta el riesgo de esfuerzo desperdiciado.

Solución de problemas

Los compañeros de equipo no aparecen

Si los compañeros de equipo no aparecen después de que le pida a Claude que cree un equipo:

- En modo en proceso, los compañeros de equipo pueden ya estar ejecutándose pero no ser visibles. Presione Shift+Down para ciclar a través de compañeros de equipo activos.
- Verifique que la tarea que le dio a Claude fue lo suficientemente compleja para justificar un equipo. Claude decide si generar compañeros de equipo según la tarea.
- Si solicitó explícitamente paneles divididos, asegúrese de que tmux esté instalado y disponible en su PATH:

```
which tmux
```

- Para iTerm2, verifique que la CLI `it2` esté instalada y la API de Python esté habilitada en las preferencias de iTerm2.

Demasiados avisos de permisos

Las solicitudes de permisos de compañeros de equipo suben al líder, lo que puede crear fricción. Pre-apruebe operaciones comunes en su [configuración de permisos](#) antes de generar compañeros de equipo para reducir interrupciones.

Los compañeros de equipo se detienen en errores

Los compañeros de equipo pueden detenerse después de encontrar errores en lugar de recuperarse. Verifique su salida usando Shift+Down en modo en proceso o haciendo clic en el panel en modo dividido, luego:

- Dele instrucciones adicionales directamente
- Genere un compañero de equipo de reemplazo para continuar el trabajo

El líder se apaga antes de que el trabajo esté hecho

El líder puede decidir que el equipo está terminado antes de que todas las tareas estén realmente completas. Si esto sucede, dígame que continúe. También puede decirle al líder que espere a que los compañeros de equipo terminen antes de proceder si comienza a hacer trabajo en lugar de delegar.

Sesiones tmux huérfanas

Si una sesión tmux persiste después de que el equipo termina, puede no haber sido completamente limpiada. Enumere sesiones y mate la creada por el equipo:

```
tmux ls
tmux kill-session -t <session-name>
```

Limitaciones

Los equipos de agentes son experimentales. Las limitaciones actuales a tener en cuenta:

- **Sin reanudación de sesión con compañeros de equipo en proceso:** `/resume` y `/rewind` no restauran compañeros de equipo en proceso. Después de reanudar una sesión, el líder puede intentar enviar mensajes a compañeros de equipo que ya no existen. Si esto sucede, dígame al líder que genere nuevos compañeros de equipo.
- **El estado de la tarea puede retrasarse:** los compañeros de equipo a veces no marcan las tareas como completadas, lo que bloquea tareas dependientes. Si una tarea parece atrapada, verifique si el trabajo está realmente hecho y actualice el estado de la tarea manualmente o dígame al líder que empuje al compañero de equipo.
- **El apagado puede ser lento:** los compañeros de equipo terminan su solicitud actual o llamada de herramienta antes de apagarse, lo que puede tomar tiempo.
- **Un equipo por sesión:** un líder solo puede gestionar un equipo a la vez. Limpie el equipo actual antes de iniciar uno nuevo.
- **Sin equipos anidados:** los compañeros de equipo no pueden generar sus propios equipos o compañeros de equipo. Solo el líder puede gestionar el equipo.
- **El líder es fijo:** la sesión que crea el equipo es el líder de por vida. No puede promover un compañero de equipo a líder o transferir liderazgo.
- **Permisos establecidos en la generación:** todos los compañeros de equipo comienzan con el modo de permiso del líder. Puede cambiar modos de compañeros de equipo individuales después de generar, pero no puede establecer modos por compañero de equipo en el momento de la generación.
- **Los paneles divididos requieren tmux o iTerm2:** el modo en proceso predeterminado funciona en cualquier terminal. El modo de panel dividido no es compatible con la terminal integrada de VS Code, Windows Terminal o Ghostty.

Tip:

CLAUDE.md funciona normalmente: los compañeros de equipo leen archivos `CLAUDE.md` de su directorio de trabajo. Use esto para proporcionar orientación específica del proyecto a todos los compañeros de equipo.

Próximos pasos

Explore enfoques relacionados para trabajo paralelo y delegación:

- **Delegación ligera:** [subagents](#) generan agentes auxiliares para investigación o verificación dentro de su sesión, mejor para tareas que no necesitan coordinación entre agentes
- **Sesiones paralelas manuales:** [Git worktrees](#) le permiten ejecutar múltiples sesiones de Claude Code usted mismo sin coordinación de equipo automatizada
- **Comparar enfoques:** vea la comparación [subagent vs agent team](#) para un desglose lado a lado

Ejecutar Claude Code mediante programación

Utilice el Agent SDK para ejecutar Claude Code mediante programación desde la CLI, Python o TypeScript.

El [Agent SDK](#) le proporciona las mismas herramientas, bucle de agente y gestión de contexto que potencian Claude Code. Está disponible como CLI para scripts e CI/CD, o como paquetes de [Python](#) y [TypeScript](#) para control programático completo.

Note:

La CLI se llamaba anteriormente “modo sin interfaz”. La bandera `-p` y todas las opciones de CLI funcionan de la misma manera.

Para ejecutar Claude Code mediante programación desde la CLI, pase `-p` con su indicación y cualquier [opción de CLI](#):

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

Esta página cubre el uso del Agent SDK a través de la CLI (`claude -p`). Para los paquetes SDK de Python y TypeScript con salidas estructuradas, devoluciones de llamada de aprobación de herramientas y objetos de mensaje nativos, consulte la [documentación completa del Agent SDK](#).

Uso básico

Agregue la bandera `-p` (o `--print`) a cualquier comando `claude` para ejecutarlo de forma no interactiva. Todas las [opciones de CLI](#) funcionan con `-p`, incluyendo:

- `--continue` para [continuar conversaciones](#)
- `--allowedTools` para [aprobar herramientas automáticamente](#)
- `--output-format` para [obtener salida estructurada](#)

Este ejemplo le pregunta a Claude sobre su base de código e imprime la respuesta:

```
claude -p "What does the auth module do?"
```

Ejemplos

Estos ejemplos destacan patrones comunes de CLI.

Obtener salida estructurada

Utilice `--output-format` para controlar cómo se devuelven las respuestas:

- `text` (predeterminado): salida de texto sin formato
- `json` : JSON estructurado con resultado, ID de sesión y metadatos
- `stream-json` : JSON delimitado por saltos de línea para transmisión en tiempo real

Este ejemplo devuelve un resumen del proyecto como JSON con metadatos de sesión, con el resultado de texto en el campo `result` :

```
claude -p "Summarize this project" --output-format json
```

Para obtener una salida que se ajuste a un esquema específico, utilice `--output-format json` con `--json-schema` y una definición de [JSON Schema](#). La respuesta incluye metadatos sobre la solicitud (ID de sesión, uso, etc.) con la salida estructurada en el campo `structured_output`.

Este ejemplo extrae nombres de funciones y los devuelve como una matriz de cadenas:

```
claude -p "Extract the main function names from auth.py" \  
  --output-format json \  
  --json-schema '{"type":"object","properties":{"functions":  
{"type":"array","items":{"type":"string"}},"required":["functions"]}'
```

Tip:

Utilice una herramienta como `jq` para analizar la respuesta y extraer campos específicos:

```
## Extract the text result
claude -p "Summarize this project" --output-format json | jq -r '.result'

## Extract structured output
claude -p "Extract function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":\
{"type":"array","items":{"type":"string"}},"required":["functions"]}}' \
  | jq '.structured_output'
```

Transmitir respuestas

Utilice `--output-format stream-json` con `--verbose` e `--include-partial-messages` para recibir tokens a medida que se generan. Cada línea es un objeto JSON que representa un evento:

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

El siguiente ejemplo utiliza `jq` para filtrar deltas de texto y mostrar solo el texto transmitido. La bandera `-r` genera cadenas sin formato (sin comillas) y `-j` se une sin saltos de línea para que los tokens se transmitan continuamente:

```
claude -p "Write a poem" --output-format stream-json --verbose --include-partial-
messages | \
  jq -rj 'select(.type == "stream_event" and .event.delta.type? == "text_delta")
| .event.delta.text'
```

Para transmisión programática con devoluciones de llamada y objetos de mensaje, consulte [Transmitir respuestas en tiempo real](#) en la documentación del Agent SDK.

Aprobar herramientas automáticamente

Utilice `--allowedTools` para permitir que Claude use ciertas herramientas sin solicitar confirmación. Este ejemplo ejecuta un conjunto de pruebas y corrige fallos, permitiendo que Claude ejecute comandos Bash y lea/edite archivos sin pedir permiso:

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

Crear una confirmación

Este ejemplo revisa los cambios preparados y crea una confirmación con un mensaje apropiado:

```
claude -p "Look at my staged changes and create an appropriate commit" \
  --allowedTools "Bash(git diff *),Bash(git log *),Bash(git status *),Bash(git
  commit *)"
```

La bandera `--allowedTools` utiliza [sintaxis de regla de permiso](#). El `*` final habilita la coincidencia de prefijo, por lo que `Bash(git diff *)` permite cualquier comando que comience con `git diff`. El espacio antes de `*` es importante: sin él, `Bash(git diff*)` también coincidiría con `git diff-index`.

Note:

Las [skills](#) invocadas por el usuario como `/commit` y los [comandos integrados](#) solo están disponibles en modo interactivo. En modo `-p`, describa la tarea que desea realizar en su lugar.

Personalizar el indicador del sistema

Utilice `--append-system-prompt` para agregar instrucciones mientras mantiene el comportamiento predeterminado de Claude Code. Este ejemplo canaliza un diff de PR a Claude e le indica que revise las vulnerabilidades de seguridad:

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for
  vulnerabilities." \
  --output-format json
```

Consulte [banderas de indicador del sistema](#) para más opciones, incluyendo `--system-prompt` para reemplazar completamente el indicador predeterminado.

Continuar conversaciones

Utilice `--continue` para continuar la conversación más reciente, o `--resume` con un ID de sesión para continuar una conversación específica. Este ejemplo ejecuta una revisión y luego envía indicaciones de seguimiento:

```
## First request
claude -p "Review this codebase for performance issues"

## Continue the most recent conversation
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

Si está ejecutando múltiples conversaciones, capture el ID de sesión para reanudar una específica:

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

Próximos pasos

- [Inicio rápido del Agent SDK](#): construya su primer agente con Python o TypeScript
- [Referencia de CLI](#): todas las banderas y opciones de CLI
- [GitHub Actions](#): utilice el Agent SDK en flujos de trabajo de GitHub
- [GitLab CI/CD](#): utilice el Agent SDK en canalizaciones de GitLab

Continúa sesiones locales desde cualquier dispositivo con Control Remoto

Continúa una sesión local de Claude Code desde tu teléfono, tableta o cualquier navegador usando Remote Control. Funciona con claude.ai/code y la aplicación móvil de Claude.

Note:

Remote Control está disponible en todos los planes. Los administradores de Team y Enterprise deben habilitar primero Claude Code en [configuración de administración](#).

Remote Control conecta claude.ai/code o la aplicación Claude para [iOS](#) y [Android](#) a una sesión de Claude Code que se ejecuta en tu máquina. Inicia una tarea en tu escritorio y luego continúa desde tu teléfono en el sofá o desde un navegador en otra computadora.

Cuando inicias una sesión de Remote Control en tu máquina, Claude sigue ejecutándose localmente todo el tiempo, por lo que nada se mueve a la nube. Con Remote Control puedes:

- **Usar tu entorno local completo de forma remota:** tu sistema de archivos, [MCP servers](#), herramientas y configuración del proyecto permanecen disponibles
- **Trabajar desde ambas superficies a la vez:** la conversación se mantiene sincronizada en todos los dispositivos conectados, para que puedas enviar mensajes desde tu terminal, navegador y teléfono indistintamente
- **Sobrevivir a interrupciones:** si tu portátil se duerme o tu red se cae, la sesión se reconecta automáticamente cuando tu máquina vuelve a estar en línea

A diferencia de [Claude Code en la web](#), que se ejecuta en infraestructura en la nube, las sesiones de Remote Control se ejecutan directamente en tu máquina e interactúan con tu sistema de archivos local. Las interfaces web y móvil son solo una ventana a esa sesión local.

Note:

Remote Control requiere Claude Code v2.1.51 o posterior. Verifica tu versión con `claude --version`.

Esta página cubre la configuración, cómo iniciar y conectarse a sesiones, y cómo Remote Control se compara con Claude Code en la web.

Requisitos

Antes de usar Remote Control, confirma que tu entorno cumple con estas condiciones:

- **Suscripción:** disponible en planes Pro, Max, Team y Enterprise. Los administradores de Team y Enterprise deben habilitar primero Claude Code en [configuración de administración](#). Las claves API no son compatibles.
- **Autenticación:** ejecuta `claude` y usa `/login` para iniciar sesión a través de claude.ai si aún no lo has hecho.
- **Confianza del espacio de trabajo:** ejecuta `claude` en tu directorio de proyecto al menos una vez para aceptar el diálogo de confianza del espacio de trabajo.

Inicia una sesión de Remote Control

Puedes iniciar una nueva sesión directamente en Remote Control, o conectar una sesión que ya está en ejecución.

Nueva sesión

Navega a tu directorio de proyecto y ejecuta:

```
claude remote-control
```

El proceso sigue ejecutándose en tu terminal, esperando conexiones remotas. Muestra una URL de sesión que puedes usar para [conectarte desde otro dispositivo](#), y puedes presionar la barra espaciadora para mostrar un código QR para acceso rápido desde tu teléfono. Mientras una sesión remota está activa, la terminal muestra el estado de la conexión y la actividad de las herramientas.

Este comando admite las siguientes banderas:

- `--name "My Project"` : establece un título de sesión personalizado visible en la lista de sesiones en claude.ai/code. También puedes pasar el nombre como argumento posicional: `claude remote-control "My Project"`
- `--verbose` : muestra registros detallados de conexión y sesión
- `--sandbox` / `--no-sandbox` : habilita o deshabilita [sandboxing](#) para aislamiento del sistema de archivos y red durante la sesión. El sandboxing está deshabilitado de forma predeterminada.

Desde una sesión existente

Si ya estás en una sesión de Claude Code y deseas continuarla de forma remota, usa el comando `/remote-control` (o `/rc`):

```
/remote-control
```

Pasa un nombre como argumento para establecer un título de sesión personalizado:

```
/remote-control My Project
```

Esto inicia una sesión de Remote Control que lleva tu historial de conversación actual y muestra una URL de sesión y código QR que puedes usar para [conectarte desde otro dispositivo](#). Las banderas `--verbose`, `--sandbox` y `--no-sandbox` no están disponibles con este comando.

Conectarse desde otro dispositivo

Una vez que una sesión de Remote Control está activa, tienes varias formas de conectarte desde otro dispositivo:

- **Abre la URL de sesión** en cualquier navegador para ir directamente a la sesión en claude.ai/code. Tanto `claude remote-control` como `/remote-control` muestran esta URL en la terminal.
- **Escanea el código QR** que se muestra junto a la URL de sesión para abrirlo directamente en la aplicación Claude. Con `claude remote-control`, presiona la barra espaciadora para alternar la visualización del código QR.
- **Abre claude.ai/code o la aplicación Claude** y encuentra la sesión por nombre en la lista de sesiones. Las sesiones de Remote Control muestran un icono de computadora con un punto de estado verde cuando están en línea.

La sesión remota toma su nombre del argumento `--name` (o el nombre pasado a `/remote-control`), tu último mensaje, tu valor `/rename`, o “Remote Control session” si no hay historial de conversación. Si el entorno ya tiene una sesión activa, se te preguntará si deseas continuarla o iniciar una nueva.

Si aún no tienes la aplicación Claude, usa el comando `/mobile` dentro de Claude Code para mostrar un código QR de descarga para [iOS](#) o [Android](#).

Habilita Remote Control para todas las sesiones

De forma predeterminada, Remote Control solo se activa cuando ejecutas explícitamente `claude remote-control` o `/remote-control`. Para habilitarlo automáticamente para cada sesión, ejecuta `/config` dentro de Claude Code y establece **Enable Remote Control for all sessions** en `true`. Establécelo de nuevo en `false` para deshabilitarlo.

Cada instancia de Claude Code admite una sesión remota a la vez. Si ejecutas múltiples instancias, cada una obtiene su propio entorno y sesión.

Conexión y seguridad

Tu sesión local de Claude Code realiza solo solicitudes HTTPS salientes y nunca abre puertos entrantes en tu máquina. Cuando inicias Remote Control, se registra con la API de Anthropic y sondea en busca de trabajo. Cuando te conectas desde otro dispositivo, el servidor enruta mensajes entre el cliente web o móvil y tu sesión local a través de una conexión de transmisión.

Todo el tráfico viaja a través de la API de Anthropic sobre TLS, el mismo transporte de seguridad que cualquier sesión de Claude Code. La conexión utiliza múltiples credenciales de corta duración, cada una limitada a un propósito único y expirando de forma independiente.

Remote Control vs Claude Code en la web

Remote Control y [Claude Code en la web](#) ambos usan la interfaz `claude.ai/code`. La diferencia clave es dónde se ejecuta la sesión: Remote Control se ejecuta en tu máquina, por lo que tus MCP servers locales, herramientas y configuración del proyecto permanecen disponibles. Claude Code en la web se ejecuta en infraestructura en la nube administrada por Anthropic.

Usa Remote Control cuando estés en medio de trabajo local y desees continuar desde otro dispositivo. Usa Claude Code en la web cuando desees iniciar una tarea sin ninguna configuración local, trabajar en un repositorio que no tienes clonado, o ejecutar múltiples tareas en paralelo.

Limitaciones

- **Una sesión remota a la vez:** cada sesión de Claude Code admite una conexión remota.

- **La terminal debe permanecer abierta:** Remote Control se ejecuta como un proceso local. Si cierras la terminal o detienes el proceso `claude`, la sesión termina. Ejecuta `claude remote-control` de nuevo para iniciar una nueva.
- **Interrupción de red extendida:** si tu máquina está despierta pero no puede alcanzar la red durante más de aproximadamente 10 minutos, la sesión agota el tiempo de espera y el proceso se cierra. Ejecuta `claude remote-control` de nuevo para iniciar una nueva sesión.

Recursos relacionados

- [Claude Code en la web](#): ejecuta sesiones en entornos en la nube administrados por Anthropic en lugar de en tu máquina
- [Autenticación](#): configura `/login` y gestiona credenciales para claude.ai
- [Referencia de CLI](#): lista completa de banderas y comandos incluyendo `claude remote-control`
- [Seguridad](#): cómo las sesiones de Remote Control se ajustan al modelo de seguridad de Claude Code
- [Uso de datos](#): qué datos fluyen a través de la API de Anthropic durante sesiones locales y remotas

Ejecutar prompts en un horario

Utilice `/loop` y las herramientas de programación cron para ejecutar prompts repetidamente, sondear el estado o establecer recordatorios únicos dentro de una sesión de Claude Code.

Note:

Las tareas programadas requieren Claude Code v2.1.72 o posterior. Verifique su versión con `claude --version`.

Las tareas programadas permiten que Claude vuelva a ejecutar un prompt automáticamente en un intervalo. Úselas para sondear una implementación, supervisar un PR, verificar una compilación de larga duración o recordarse a sí mismo que debe hacer algo más adelante en la sesión.

Las tareas tienen alcance de sesión: viven en el proceso actual de Claude Code y desaparecen cuando sale. Para la programación duradera que sobrevive a los reinicios y se ejecuta sin una sesión de terminal activa, consulte [tareas programadas de Desktop](#) o [GitHub Actions](#).

Programar un prompt recurrente con `/loop`

La [skill agrupada](#) `/loop` es la forma más rápida de programar un prompt recurrente. Pase un intervalo opcional y un prompt, y Claude configura un trabajo cron que se dispara en segundo plano mientras la sesión permanece abierta.

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude analiza el intervalo, lo convierte en una expresión cron, programa el trabajo y confirma la cadencia y el ID del trabajo.

Sintaxis de intervalo

Los intervalos son opcionales. Puede colocarlos al principio, al final o dejarlos fuera completamente.

Forma	Ejemplo	Intervalo analizado
Token inicial	<code>/loop 30m check the build</code>	cada 30 minutos
Cláusula <code>every</code> final	<code>/loop check the build every 2 hours</code>	cada 2 horas
Sin intervalo	<code>/loop check the build</code>	por defecto cada 10 minutos

Las unidades admitidas son `s` para segundos, `m` para minutos, `h` para horas y `d` para días. Los segundos se redondean al minuto más cercano ya que cron tiene una granularidad de un minuto. Los intervalos que no se dividen uniformemente en su unidad, como `7m` o `90m`, se redondean al intervalo más limpio y Claude le dice cuál eligió.

Bucle sobre otro comando

El prompt programado puede ser en sí mismo una invocación de comando o skill. Esto es útil para volver a ejecutar un flujo de trabajo que ya ha empaquetado.

```
/loop 20m /review-pr 1234
```

Cada vez que se dispara el trabajo, Claude ejecuta `/review-pr 1234` como si lo hubiera escrito.

Establecer un recordatorio único

Para recordatorios únicos, describa lo que desea en lenguaje natural en lugar de usar `/loop`. Claude programa una tarea de un solo disparo que se elimina a sí misma después de ejecutarse.

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude fija la hora de disparo a un minuto y hora específicos usando una expresión cron y confirma cuándo se disparará.

Gestionar tareas programadas

Pida a Claude en lenguaje natural que enumere o cancele tareas, o haga referencia directamente a las herramientas subyacentes.

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

Bajo el capó, Claude utiliza estas herramientas:

Herra- mienta	Propósito
<code>CronCreate</code>	Programar una nueva tarea. Acepta una expresión cron de 5 campos, el prompt a ejecutar y si se repite o se dispara una sola vez.
<code>CronList</code>	Enumerar todas las tareas programadas con sus IDs, horarios y prompts.
<code>CronDelete</code>	Cancelar una tarea por ID.

Cada tarea programada tiene un ID de 8 caracteres que puede pasar a `CronDelete`. Una sesión puede contener hasta 50 tareas programadas a la vez.

Cómo se ejecutan las tareas programadas

El programador verifica cada segundo si hay tareas vencidas y las encola con baja prioridad. Un prompt programado se dispara entre sus turnos, no mientras Claude está en medio de una respuesta. Si Claude está ocupado cuando vence una tarea, el prompt espera hasta que termina el turno actual.

Todos los tiempos se interpretan en su zona horaria local. Una expresión cron como `0 9 * * *` significa las 9am donde está ejecutando Claude Code, no UTC.

Jitter

Para evitar que cada sesión golpee la API en el mismo momento de reloj de pared, el programador agrega un pequeño desplazamiento determinista a los tiempos de disparo:

- Las tareas recurrentes se disparan hasta un 10% de su período tarde, limitado a 15 minutos. Un trabajo por hora podría dispararse en cualquier momento desde `:00` hasta `:06`.

- Las tareas únicas programadas para la parte superior o inferior de la hora se disparan hasta 90 segundos antes.

El desplazamiento se deriva del ID de la tarea, por lo que la misma tarea siempre obtiene el mismo desplazamiento. Si el tiempo exacto es importante, elija un minuto que no sea `:00` o `:30`, por ejemplo `3 9 * * *` en lugar de `0 9 * * *`, y el jitter único no se aplicará.

Vencimiento de tres días

Las tareas recurrentes expiran automáticamente 3 días después de su creación. La tarea se dispara una última vez y luego se elimina a sí misma. Esto limita cuánto tiempo puede ejecutarse un bucle olvidado. Si necesita que una tarea recurrente dure más tiempo, cancele y recree antes de que expire, o use [tareas programadas de Desktop](#) para programación duradera.

Referencia de expresión cron

CronCreate acepta expresiones cron estándar de 5 campos: `minute hour day-of-month month day-of-week`. Todos los campos admiten comodines (`*`), valores únicos (`5`), pasos (`*/15`), rangos (`1-5`) y listas separadas por comas (`1,15,30`).

Ejemplo	Significado
<code>* / 5 * * * *</code>	Cada 5 minutos
<code>0 * * * *</code>	Cada hora en punto
<code>7 * * * *</code>	Cada hora a los 7 minutos
<code>0 9 * * *</code>	Todos los días a las 9am local
<code>0 9 * * 1-5</code>	Días de semana a las 9am local
<code>30 14 15 3 *</code>	15 de marzo a las 2:30pm local

El día de la semana usa `0` o `7` para domingo hasta `6` para sábado. La sintaxis extendida como `L`, `W`, `?` y alias de nombres como `MON` o `JAN` no se admiten.

Cuando tanto el día del mes como el día de la semana están restringidos, una fecha coincide si cualquiera de los campos coincide. Esto sigue la semántica estándar de vixie-cron.

Deshabilitar tareas programadas

Establezca `CLAUDE_CODE_DISABLE_CRON=1` en su entorno para deshabilitar completamente el programador. Las herramientas cron y `/Loop` dejan de estar disponibles, y cualquier tarea ya programada deja de dispararse. Consulte [Variables de entorno](#) para la lista completa de banderas de deshabilitación.

Limitaciones

La programación con alcance de sesión tiene limitaciones inherentes:

- Las tareas solo se disparan mientras Claude Code se está ejecutando e inactivo. Cerrar la terminal o dejar que la sesión salga cancela todo.
- Sin recuperación de disparos perdidos. Si el tiempo programado de una tarea pasa mientras Claude está ocupado en una solicitud de larga duración, se dispara una vez cuando Claude queda inactivo, no una vez por intervalo perdido.
- Sin persistencia entre reinicios. Reiniciar Claude Code borra todas las tareas con alcance de sesión.

Para la automatización impulsada por cron que necesita ejecutarse sin supervisión, use un [flujo de trabajo de GitHub Actions](#) con un disparador `schedule`, o [tareas programadas de Desktop](#) si desea un flujo de configuración gráfico.

Code Review

Configure revisiones automatizadas de PR que detecten errores lógicos, vulnerabilidades de seguridad y regresiones mediante análisis multiagente de su base de código completa

Note:

Code Review está en vista previa de investigación, disponible para suscripciones de [Teams y Enterprise](#). No está disponible para organizaciones con [Zero Data Retention](#) habilitado.

Code Review analiza sus solicitudes de extracción de GitHub y publica hallazgos como comentarios en línea en las líneas de código donde encontró problemas. Una flota de agentes especializados examina los cambios de código en el contexto de su base de código completa, buscando errores lógicos, vulnerabilidades de seguridad, casos límite rotos y regresiones sutiles.

Los hallazgos se etiquetan por severidad y no aprueban ni bloquean su PR, por lo que los flujos de trabajo de revisión existentes permanecen intactos. Puede ajustar lo que Claude marca agregando un archivo `CLAUDE.md` o `REVIEW.md` a su repositorio.

Para ejecutar Claude en su propia infraestructura de CI en lugar de este servicio administrado, consulte [GitHub Actions](#) o [GitLab CI/CD](#).

Esta página cubre:

- [Cómo funcionan las revisiones](#)
- [Configuración](#)
- [Personalización de revisiones](#) con `CLAUDE.md` y `REVIEW.md`
- [Precios](#)

Cómo funcionan las revisiones

Una vez que un administrador [habilita Code Review](#) para su organización, las revisiones se ejecutan automáticamente cuando se abre o actualiza una solicitud de extracción. Múltiples agentes analizan el diff y el código circundante en paralelo en la infraestructura de Anthropic. Cada agente busca una clase diferente de problema, luego un paso de verificación verifica los candidatos contra el comportamiento real del código para filtrar falsos

positivos. Los resultados se deduplican, se clasifican por severidad y se publican como comentarios en línea en las líneas específicas donde se encontraron problemas. Si no se encuentran problemas, Claude publica un breve comentario de confirmación en la PR.

Las revisiones se escalan en costo con el tamaño y la complejidad de la PR, completándose en un promedio de 20 minutos. Los administradores pueden monitorear la actividad de revisión y el gasto a través del [panel de análisis](#).

Niveles de severidad

Cada hallazgo se etiqueta con un nivel de severidad:

Marcador	Severidad	Significado
	Normal	Un error que debe corregirse antes de fusionar
	Nit	Un problema menor, vale la pena corregir pero no bloqueante
	Preexistente	Un error que existe en la base de código pero no fue introducido por esta PR

Los hallazgos incluyen una sección de razonamiento extendido plegable que puede expandir para entender por qué Claude marcó el problema y cómo verificó el problema.

Qué verifica Code Review

De forma predeterminada, Code Review se enfoca en la corrección: errores que romperían la producción, no preferencias de formato o cobertura de pruebas faltante. Puede expandir lo que verifica [agregando archivos de orientación](#) a su repositorio.

Configurar Code Review

Un administrador habilita Code Review una vez para la organización y selecciona qué repositorios incluir.

Step 1: Abrir configuración de administrador de Claude Code

Vaya a [claude.ai/admin-settings/claude-code](#) y encuentre la sección Code Review. Necesita acceso de administrador a su organización de Claude y permiso para instalar GitHub Apps en su organización de GitHub.

Step 2: Iniciar configuración

Haga clic en **Setup**. Esto inicia el flujo de instalación de GitHub App.

Step 3: Instalar la Claude GitHub App

Siga las indicaciones para instalar la Claude GitHub App en su organización de GitHub. La aplicación solicita estos permisos de repositorio:

- **Contents:** lectura y escritura
- **Issues:** lectura y escritura
- **Pull requests:** lectura y escritura

Code Review utiliza acceso de lectura a contenidos y acceso de escritura a solicitudes de extracción. El conjunto de permisos más amplio también admite [GitHub Actions](#) si lo habilita más adelante.

Step 4: Seleccionar repositorios

Elija qué repositorios habilitar para Code Review. Si no ve un repositorio, asegúrese de haber dado a la Claude GitHub App acceso a él durante la instalación. Puede agregar más repositorios más adelante.

Step 5: Establecer disparadores de revisión por repositorio

Después de que se complete la configuración, la sección Code Review muestra sus repositorios en una tabla. Para cada repositorio, use el menú desplegable para elegir cuándo se ejecutan las revisiones:

- **After PR creation only:** la revisión se ejecuta una vez cuando se abre una PR o se marca como lista para revisión
- **After every push to PR branch:** la revisión se ejecuta en cada push, detectando nuevos problemas a medida que la PR evoluciona y resolviendo automáticamente los hilos cuando corrige problemas marcados

Revisar en cada push ejecuta más revisiones y cuesta más. Comience con la creación de PR solamente y cambie a on-push para repositorios donde desea cobertura continua y limpieza automática de hilos.

La tabla de repositorios también muestra el costo promedio por revisión para cada repositorio basado en la actividad reciente. Use el menú de acciones de fila para activar o desactivar Code Review por repositorio, o para eliminar un repositorio por completo.

Para verificar la configuración, abra una PR de prueba. Una ejecución de verificación llamada **Claude Code Review** aparece dentro de unos minutos. Si no aparece, confirme que el repositorio está listado en su configuración de administrador y que la Claude GitHub App tiene acceso a él.

Personalizar revisiones

Code Review lee dos archivos de su repositorio para guiar lo que marca. Ambos son aditivos además de las verificaciones de corrección predeterminadas:

- **CLAUDE.md** : instrucciones de proyecto compartidas que Claude Code utiliza para todas las tareas, no solo revisiones. Úselo cuando la orientación también se aplique a sesiones interactivas de Claude Code.
- **REVIEW.md** : orientación solo de revisión, leída exclusivamente durante revisiones de código. Úselo para reglas que son estrictamente sobre qué marcar u omitir durante la revisión y que abarrotarían su **CLAUDE.md** general.

CLAUDE.md

Code Review lee los archivos **CLAUDE.md** de su repositorio y trata las violaciones recién introducidas como hallazgos de nivel nit. Esto funciona bidireccionalmente: si su PR cambia el código de una manera que hace que una declaración **CLAUDE.md** sea obsoleta, Claude marca que los documentos necesitan actualización también.

Claude lee archivos **CLAUDE.md** en cada nivel de su jerarquía de directorios, por lo que las reglas en el **CLAUDE.md** de un subdirectorio se aplican solo a archivos bajo esa ruta. Consulte la [documentación de memoria](#) para obtener más información sobre cómo funciona **CLAUDE.md**.

Para orientación específica de revisión que no desea aplicar a sesiones generales de Claude Code, use **REVIEW.md** en su lugar.

REVIEW.md

Agregue un archivo **REVIEW.md** a la raíz de su repositorio para reglas específicas de revisión. Úselo para codificar:

- Directrices de estilo de empresa o equipo: “preferir retornos tempranos sobre condicionales anidados”
- Convenciones específicas de lenguaje o marco no cubiertas por linters
- Cosas que Claude siempre debe marcar: “cualquier nueva ruta de API debe tener una prueba de integración”
- Cosas que Claude debe omitir: “no comentar sobre formato en código generado bajo `/gen/`”

Ejemplo **REVIEW.md** :

```
## Directrices de Code Review

### Siempre verificar
- Los nuevos puntos finales de API tienen pruebas de integración correspondientes
- Las migraciones de base de datos son compatibles hacia atrás
- Los mensajes de error no filtran detalles internos a los usuarios

### Estilo
- Preferir declaraciones `match` sobre verificaciones `isinstance` encadenadas
- Usar registro estructurado, no interpolación de f-string en llamadas de registro

### Omitir
- Archivos generados bajo `src/gen/`
- Cambios solo de formato en archivos `*.lock`
```

Claude descubre automáticamente `REVIEW.md` en la raíz del repositorio. No se necesita configuración.

Ver uso

Vaya a claude.ai/analytics/code-review para ver la actividad de Code Review en toda su organización. El panel muestra:

Sección	Lo que muestra
PRs re-viewed	Conteo diario de solicitudes de extracción revisadas durante el rango de tiempo seleccionado
Cost weekly	Gasto semanal en Code Review
Feedback	Conteo de comentarios de revisión que se resolvieron automáticamente porque un desarrollador abordó el problema
Repository breakdown	Conteos por repositorio de PRs revisadas y comentarios resueltos

La tabla de repositorios en la configuración de administrador también muestra el costo promedio por revisión para cada repositorio.

Precios

Code Review se factura según el uso de tokens. Las revisiones promedian \$15-25, escalando con el tamaño de la PR, la complejidad de la base de código y cuántos problemas requieren verificación. El uso de Code Review se factura por separado a través de [extra usage](#) y no cuenta contra el uso incluido en su plan.

El disparador de revisión que elija afecta el costo total:

- **After PR creation only:** se ejecuta una vez por PR
- **After every push:** se ejecuta en cada commit, multiplicando el costo por el número de pushes

Los costos aparecen en su factura de Anthropic independientemente de si su organización usa AWS Bedrock o Google Vertex AI para otras características de Claude Code. Para establecer un límite de gasto mensual para Code Review, vaya a [claude.ai/admin-settings/usage](#) y configure el límite para el servicio Claude Code Review.

Monitoree el gasto a través del gráfico de costo semanal en [analytics](#) o la columna de costo promedio por repositorio en la configuración de administrador.

Recursos relacionados

Code Review está diseñado para funcionar junto con el resto de Claude Code. Si desea ejecutar revisiones localmente antes de abrir una PR, necesita una configuración autohospedada o desea profundizar en cómo `CLAUDE.md` da forma al comportamiento de Claude en todas las herramientas, estas páginas son buenos siguientes pasos:

- [Plugins](#): explore el mercado de plugins, incluido un plugin `code-review` para ejecutar revisiones bajo demanda localmente antes de hacer push
- [GitHub Actions](#): ejecute Claude en sus propios flujos de trabajo de GitHub Actions para automatización personalizada más allá de la revisión de código
- [GitLab CI/CD](#): integración de Claude autohospedada para tuberías de GitLab
- [Memory](#): cómo funcionan los archivos `CLAUDE.md` en Claude Code
- [Analytics](#): rastrear el uso de Claude Code más allá de la revisión de código

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

Aprenda a integrar Claude Code en su flujo de trabajo de desarrollo con Claude Code GitHub Actions

Claude Code GitHub Actions trae automatización impulsada por IA a su flujo de trabajo de GitHub. Con una simple mención `@claude` en cualquier PR o problema, Claude puede analizar su código, crear solicitudes de extracción, implementar características y corregir errores, todo mientras sigue los estándares de su proyecto. Para revisiones automáticas publicadas en cada PR sin un disparador, consulte [GitHub Code Review](#).

Note:

Claude Code GitHub Actions se construye sobre el [Claude Agent SDK](#), que permite la integración programática de Claude Code en sus aplicaciones. Puede usar el SDK para crear flujos de trabajo de automatización personalizados más allá de GitHub Actions.

Info:

Claude Opus 4.6 ya está disponible. Claude Code GitHub Actions utiliza Sonnet de forma predeterminada. Para usar Opus 4.6, configure el [parámetro de modelo](#) para usar `claude-opus-4-6`.

¿Por qué usar Claude Code GitHub Actions?

- **Creación instantánea de PR:** Describa lo que necesita y Claude crea un PR completo con todos los cambios necesarios
- **Implementación de código automatizada:** Convierta problemas en código funcional con un único comando
- **Sigue sus estándares:** Claude respeta sus directrices `CLAUDE.md` y patrones de código existentes
- **Configuración simple:** Comience en minutos con nuestro instalador y clave API
- **Seguro por defecto:** Su código permanece en los ejecutores de Github

¿Qué puede hacer Claude?

Claude Code proporciona una poderosa GitHub Action que transforma la forma en que trabaja con código:

Claude Code Action

Esta GitHub Action le permite ejecutar Claude Code dentro de sus flujos de trabajo de GitHub Actions. Puede usar esto para crear cualquier flujo de trabajo personalizado sobre Claude Code.

[Ver repositorio →](#)

Configuración

Configuración rápida

La forma más fácil de configurar esta acción es a través de Claude Code en la terminal. Solo abra claude y ejecute `/install-github-app`.

Este comando lo guiará a través de la configuración de la aplicación de GitHub y los secretos requeridos.

Note:

- Debe ser administrador del repositorio para instalar la aplicación de GitHub y agregar secretos
- La aplicación de GitHub solicitará permisos de lectura y escritura para Contenidos, Problemas y Solicitudes de extracción
- Este método de inicio rápido solo está disponible para usuarios directos de Claude API. Si está usando AWS Bedrock o Google Vertex AI, consulte la sección [Usar con AWS Bedrock y Google Vertex AI](#).

Configuración manual

Si el comando `/install-github-app` falla o prefiere la configuración manual, siga estas instrucciones de configuración manual:

1. **Instale la aplicación de GitHub de Claude** en su repositorio: <https://github.com/apps/claude>

La aplicación de GitHub de Claude requiere los siguientes permisos de repositorio:

- **Contenidos:** Lectura y escritura (para modificar archivos del repositorio)
- **Problemas:** Lectura y escritura (para responder a problemas)

- **Solicitudes de extracción:** Lectura y escritura (para crear PR e insertar cambios)

Para más detalles sobre seguridad y permisos, consulte la [documentación de seguridad](#).

2. **Agregue ANTHROPIC_API_KEY** a sus secretos del repositorio ([Aprenda cómo usar secretos en GitHub Actions](#))
3. **Copie el archivo de flujo de trabajo** de [examples/claude.yml](#) en el directorio `.github/workflows/` de su repositorio

Tip:

Después de completar la configuración rápida o manual, pruebe la acción etiquetando `@claude` en un comentario de problema o PR.

Actualización desde Beta

Warning:

Claude Code GitHub Actions v1.0 introduce cambios importantes que requieren actualizar sus archivos de flujo de trabajo para actualizar a v1.0 desde la versión beta.

Si actualmente está usando la versión beta de Claude Code GitHub Actions, le recomendamos que actualice sus flujos de trabajo para usar la versión GA. La nueva versión simplifica la configuración mientras agrega características poderosas como la detección automática de modo.

Cambios esenciales

Todos los usuarios de beta deben hacer estos cambios en sus archivos de flujo de trabajo para actualizar:

1. **Actualice la versión de la acción:** Cambie `@beta` a `@v1`
2. **Elimine la configuración de modo:** Elimine `mode: "tag"` o `mode: "agent"` (ahora se detecta automáticamente)
3. **Actualice las entradas de solicitud:** Reemplace `direct_prompt` con `prompt`
4. **Mueva opciones de CLI:** Convierta `max_turns`, `model`, `custom_instructions`, etc. a `claude_args`

Referencia de cambios importantes

Entrada Beta antigua	Nueva entrada v1.0
mode	(Eliminado - se detecta automáticamente)
direct_prompt	prompt
override_prompt	prompt con variables de GitHub
custom_instructions	claude_args: --append-system-prompt
max_turns	claude_args: --max-turns
model	claude_args: --model
allowed_tools	claude_args: --allowedTools
disallowed_tools	claude_args: --disallowedTools
claude_env	settings formato JSON

Ejemplo antes y después

Versión beta:

```
- uses: anthropics/claude-code-action@beta
  with:
    mode: "tag"
    direct_prompt: "Review this PR for security issues"
    anthropic_api_key: "${{ secrets.ANTHROPIC_API_KEY }}"
    custom_instructions: "Follow our coding standards"
    max_turns: "10"
    model: "claude-sonnet-4-6"
```

Versión GA (v1.0):

```
- uses: anthropics/claude-code-action@v1
  with:
    prompt: "Review this PR for security issues"
    anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
    claude_args: |
      --append-system-prompt "Follow our coding standards"
      --max-turns 10
      --model claude-sonnet-4-6
```

Tip:

La acción ahora detecta automáticamente si ejecutar en modo interactivo (responde a menciones `@claude`) o modo de automatización (se ejecuta inmediatamente con un solicitud) según su configuración.

Casos de uso de ejemplo

Claude Code GitHub Actions puede ayudarle con una variedad de tareas. El [directorio de ejemplos](#) contiene flujos de trabajo listos para usar para diferentes escenarios.

Flujo de trabajo básico

```
name: Claude Code
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          # Responds to @claude mentions in comments
```

Usar skills

```
name: Code Review
on:
  pull_request:
    types: [opened, synchronize]
jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Review this pull request for code quality, correctness, and
security. Analyze the diff, then post your findings as review comments."
          claude_args: "--max-turns 5"
```

Automatización personalizada con solicitudes

```
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Generate a summary of yesterday's commits and open issues"
          claude_args: "--model opus"
```

Casos de uso comunes

En comentarios de problema o PR:

```
@claude implement this feature based on the issue description
@claude how should I implement user authentication for this endpoint?
@claude fix the TypeError in the user dashboard component
```

Claude analizará automáticamente el contexto y responderá apropiadamente.

Mejores prácticas

Configuración de CLAUDE.md

Cree un archivo `CLAUDE.md` en la raíz de su repositorio para definir directrices de estilo de código, criterios de revisión, reglas específicas del proyecto y patrones preferidos. Este archivo guía la comprensión de Claude de los estándares de su proyecto.

Consideraciones de seguridad

Warning:

Nunca confirme claves API directamente en su repositorio.

Para una guía de seguridad completa que incluya permisos, autenticación y mejores prácticas, consulte la [documentación de seguridad de Claude Code Action](#).

Siempre use GitHub Secrets para claves API:

- Agregue su clave API como un secreto del repositorio llamado `ANTHROPIC_API_KEY`
- Haga referencia a él en flujos de trabajo: `anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`
- Limite los permisos de acción solo a lo necesario
- Revise las sugerencias de Claude antes de fusionar

Siempre use GitHub Secrets (por ejemplo, `${{ secrets.ANTHROPIC_API_KEY }}`) en lugar de codificar claves API directamente en sus archivos de flujo de trabajo.

Optimización del rendimiento

Use plantillas de problemas para proporcionar contexto, mantenga su `CLAUDE.md` conciso y enfocado, y configure tiempos de espera apropiados para sus flujos de trabajo.

Costos de CI

Al usar Claude Code GitHub Actions, tenga en cuenta los costos asociados:

Costos de GitHub Actions:

- Claude Code se ejecuta en ejecutores alojados en GitHub, que consumen sus minutos de GitHub Actions
- Consulte la [documentación de facturación de GitHub](#) para obtener detalles de precios y límites de minutos

Costos de API:

- Cada interacción de Claude consume tokens de API según la longitud de solicitudes y respuestas
- El uso de tokens varía según la complejidad de la tarea y el tamaño de la base de código
- Consulte la [página de precios de Claude](#) para obtener las tasas de tokens actuales

Consejos de optimización de costos:

- Use comandos específicos `@claude` para reducir llamadas API innecesarias
- Configure `--max-turns` apropiado en `claude_args` para evitar iteraciones excesivas
- Establezca tiempos de espera a nivel de flujo de trabajo para evitar trabajos descontrolados
- Considere usar controles de concurrencia de GitHub para limitar ejecuciones paralelas

Ejemplos de configuración

Claude Code Action v1 simplifica la configuración con parámetros unificados:

```
- uses: anthropics/claude-code-action@v1
  with:
    anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
    prompt: "Your instructions here" # Optional
    claude_args: "--max-turns 5" # Optional CLI arguments
```

Características clave:

- **Interfaz de solicitud unificada** - Use `prompt` para todas las instrucciones
- **Skills** - Invoque [skills](#) instalados directamente desde la solicitud
- **Paso de CLI** - Cualquier argumento de CLI de Claude Code a través de `claude_args`
- **Disparadores flexibles** - Funciona con cualquier evento de GitHub

Visite el [directorio de ejemplos](#) para archivos de flujo de trabajo completos.

Tip:

Al responder a comentarios de problema o PR, Claude responde automáticamente a menciones @claude. Para otros eventos, use el parámetro `prompt` para proporcionar instrucciones.

Usar con AWS Bedrock y Google Vertex AI

Para entornos empresariales, puede usar Claude Code GitHub Actions con su propia infraestructura en la nube. Este enfoque le da control sobre la residencia de datos y la facturación mientras mantiene la misma funcionalidad.

Requisitos previos

Antes de configurar Claude Code GitHub Actions con proveedores en la nube, necesita:

Para Google Cloud Vertex AI:

1. Un proyecto de Google Cloud con Vertex AI habilitado
2. Federación de identidad de carga de trabajo configurada para GitHub Actions
3. Una cuenta de servicio con los permisos requeridos
4. Una aplicación de GitHub (recomendado) o use el GITHUB_TOKEN predeterminado

Para AWS Bedrock:

1. Una cuenta de AWS con Amazon Bedrock habilitado
2. Proveedor de identidad OIDC de GitHub configurado en AWS
3. Un rol de IAM con permisos de Bedrock
4. Una aplicación de GitHub (recomendado) o use el GITHUB_TOKEN predeterminado

Step 1: Crear una aplicación de GitHub personalizada (Recomendado para proveedores de terceros)

Para el mejor control y seguridad al usar proveedores de terceros como Vertex AI o Bedrock, le recomendamos crear su propia aplicación de GitHub:

1. Vaya a <https://github.com/settings/apps/new>
2. Complete la información básica:

- **Nombre de la aplicación de GitHub:** Elija un nombre único (por ejemplo, "YourOrg Claude Assistant")

- **URL de inicio:** El sitio web de su organización o la URL del repositorio

1. Configure los ajustes de la aplicación:

- **Webhooks:** Desmarque “Activo” (no es necesario para esta integración)

1. Establezca los permisos requeridos:

- **Permisos del repositorio:**

- Contenidos: Lectura y escritura
- Problemas: Lectura y escritura
- Solicitudes de extracción: Lectura y escritura

1. Haga clic en “Crear aplicación de GitHub”

2. Después de la creación, haga clic en “Generar una clave privada” y guarde el archivo `.pem` descargado

3. Anote su ID de aplicación en la página de configuración de la aplicación

4. Instale la aplicación en su repositorio:

- Desde la página de configuración de su aplicación, haga clic en “Instalar aplicación” en la barra lateral izquierda
- Seleccione su cuenta u organización
- Elija “Solo repositorios seleccionados” y seleccione el repositorio específico
- Haga clic en “Instalar”

1. Agregue la clave privada como un secreto a su repositorio:

- Vaya a Configuración de su repositorio → Secretos y variables → Acciones
- Cree un nuevo secreto llamado `APP_PRIVATE_KEY` con el contenido del archivo `.pem`

1. Agregue el ID de la aplicación como un secreto:

- Cree un nuevo secreto llamado `APP_ID` con el ID de su aplicación de GitHub

Note:

Esta aplicación se usará con la acción [actions/create-github-app-token](#) para generar tokens de autenticación en sus flujos de trabajo.

Alternativa para Claude API o si no desea configurar su propia aplicación de Github: Use la aplicación oficial de Anthropic:

1. Instale desde: <https://github.com/apps/claude>

2. No se requiere configuración adicional para autenticación

Step 2: Configurar autenticación del proveedor en la nube

Elija su proveedor en la nube y configure autenticación segura:

Configure AWS para permitir que GitHub Actions se autentique de forma segura sin almacenar credenciales.

***Nota de seguridad:** Use configuraciones específicas del repositorio y otorgue solo los permisos mínimos requeridos.*

Configuración requerida:

1. Habilitar Amazon Bedrock:

- Solicite acceso a modelos de Claude en Amazon Bedrock
- Para modelos entre regiones, solicite acceso en todas las regiones requeridas

1. Configurar proveedor de identidad OIDC de GitHub:

- URL del proveedor: `https://token.actions.githubusercontent.com`
- Audiencia: `sts.amazonaws.com`

1. Crear rol de IAM para GitHub Actions:

- Tipo de entidad de confianza: Identidad web
- Proveedor de identidad: `token.actions.githubusercontent.com`
- Permisos: política `AmazonBedrockFullAccess`
- Configurar política de confianza para su repositorio específico

Valores requeridos:

Después de la configuración, necesitará:

- **AWS_ROLE_TO_ASSUME:** El ARN del rol de IAM que creó

Tip:

OIDC es más seguro que usar claves de acceso estáticas de AWS porque las credenciales son temporales y se rotan automáticamente.

Consulte la [documentación de AWS](#) para obtener instrucciones detalladas de configuración de OIDC.

Configure Google Cloud para permitir que GitHub Actions se autentique de forma segura sin almacenar credenciales.

Nota de seguridad: Use configuraciones específicas del repositorio y otorgue solo los permisos mínimos requeridos.

Configuración requerida:

1. Habilitar APIs en su proyecto de Google Cloud:

- API de credenciales de IAM
- API de servicio de token de seguridad (STS)
- API de Vertex AI

1. Crear recursos de Federación de identidad de carga de trabajo:

- Crear un grupo de identidad de carga de trabajo
- Agregar un proveedor OIDC de GitHub con:
 - Emisor: `https://token.actions.githubusercontent.com`
- Asignaciones de atributos para repositorio y propietario
- **Recomendación de seguridad:** Use condiciones de atributo específicas del repositorio

1. Crear una cuenta de servicio:

- Otorgue solo el rol `Vertex AI User`
- **Recomendación de seguridad:** Cree una cuenta de servicio dedicada por repositorio

1. Configurar enlaces de IAM:

- Permitir que el grupo de identidad de carga de trabajo suplante la cuenta de servicio
- **Recomendación de seguridad:** Use conjuntos de principios específicos del repositorio

Valores requeridos:

Después de la configuración, necesitará:

- **GCP_WORKLOAD_IDENTITY_PROVIDER:** El nombre completo del recurso del proveedor
- **GCP_SERVICE_ACCOUNT:** La dirección de correo electrónico de la cuenta de servicio

Tip:

Workload Identity Federation elimina la necesidad de claves de cuenta de servicio descargables, mejorando la seguridad.

Para obtener instrucciones de configuración detalladas, consulte la [documentación de Federación de identidad de carga de trabajo de Google Cloud](#).

Step 3: Agregar secretos requeridos

Agregue los siguientes secretos a su repositorio (Configuración → Secretos y variables → Acciones):

Para Claude API (Directo):**1. Para autenticación de API:**

- `ANTHROPIC_API_KEY` : Su clave de API de Claude de console.anthropic.com

1. Para aplicación de GitHub (si usa su propia aplicación):

- `APP_ID` : El ID de su aplicación de GitHub
- `APP_PRIVATE_KEY` : El contenido de la clave privada (.pem)

Para Google Cloud Vertex AI**1. Para autenticación de GCP:**

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. Para aplicación de GitHub (si usa su propia aplicación):

- `APP_ID` : El ID de su aplicación de GitHub
- `APP_PRIVATE_KEY` : El contenido de la clave privada (.pem)

Para AWS Bedrock**1. Para autenticación de AWS:**

- `AWS_ROLE_TO_ASSUME`

1. Para aplicación de GitHub (si usa su propia aplicación):

- `APP_ID` : El ID de su aplicación de GitHub
- `APP_PRIVATE_KEY` : El contenido de la clave privada (.pem)

Step 4: Crear archivos de flujo de trabajo

Cree archivos de flujo de trabajo de GitHub Actions que se integren con su proveedor en la nube. Los ejemplos a continuación muestran configuraciones completas tanto para AWS Bedrock como para Google Vertex AI:

Requisitos previos:

- Acceso a AWS Bedrock habilitado con permisos de modelo de Claude
- GitHub configurado como proveedor de identidad OIDC en AWS
- Rol de IAM con permisos de Bedrock que confía en GitHub Actions

Secretos de GitHub requeridos:

Nombre del secreto	Descripción
<code>AWS_ROLE_TO_ASSUME</code>	ARN del rol de IAM para acceso a Bedrock
<code>APP_ID</code>	Su ID de aplicación de GitHub (de la configuración de la aplicación)
<code>APP_PRIVATE_KEY</code>	La clave privada que generó para su aplicación de GitHub

```

name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
        '@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
        contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
        '@claude'))
    runs-on: ubuntu-latest
    env:
      AWS_REGION: us-west-2
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${ secrets.APP_ID }
          private-key: ${ secrets.APP_PRIVATE_KEY }

```

```
- name: Configure AWS Credentials (OIDC)
  uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
    aws-region: us-west-2

- uses: anthropics/claude-code-action@v1
  with:
    github_token: ${{ steps.app-token.outputs.token }}
    use_bedrock: "true"
    claude_args: '--model us.anthropic.claude-sonnet-4-6 --max-turns 10'
```

Tip:

El formato de ID de modelo para Bedrock incluye un prefijo de región (por ejemplo, `us.anthropic.claude-sonnet-4-6`).

Requisitos previos:

- API de Vertex AI habilitada en su proyecto de GCP
- Federación de identidad de carga de trabajo configurada para GitHub
- Cuenta de servicio con permisos de Vertex AI

Secretos de GitHub requeridos:

Nombre del secreto	Descripción
<code>GCP_WORKLOAD_IDENTITY_PROVIDER</code>	Nombre del recurso del proveedor de identidad de carga de trabajo
<code>GCP_SERVICE_ACCOUNT</code>	Correo electrónico de la cuenta de servicio con acceso a Vertex AI
<code>APP_ID</code>	Su ID de aplicación de GitHub (de la configuración de la aplicación)
<code>APP_PRIVATE_KEY</code>	La clave privada que generó para su aplicación de GitHub

```

name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
        '@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
        contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
        '@claude'))
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${ secrets.APP_ID }
          private-key: ${ secrets.APP_PRIVATE_KEY }

      - name: Authenticate to Google Cloud

```



```

    id: auth
    uses: google-github-actions/auth@v2
    with:
      workload_identity_provider: $
    {{ secrets.GCP_WORKLOAD_IDENTITY_PROVIDER }}
      service_account: ${{ secrets.GCP_SERVICE_ACCOUNT }}

- uses: anthropics/claude-code-action@v1
  with:
    github_token: ${{ steps.app-token.outputs.token }}
    trigger_phrase: "@claude"
    use_vertex: "true"
    claude_args: '--model claude-sonnet-4@20250514 --max-turns 10'
  env:
    ANTHROPIC_VERTEX_PROJECT_ID: ${{ steps.auth.outputs.project_id }}
    CLOUD_ML_REGION: us-east5
    VERTEX_REGION_CLAUDE_3_7_SONNET: us-east5

```

Tip:

El ID del proyecto se recupera automáticamente del paso de autenticación de Google Cloud, por lo que no necesita codificarlo.

Solución de problemas

Claude no responde a comandos @claude

Verifique que la aplicación de GitHub esté instalada correctamente, compruebe que los flujos de trabajo estén habilitados, asegúrese de que la clave API esté configurada en los secretos del repositorio y confirme que el comentario contenga `@claude` (no `/claude`).

CI no se ejecuta en los commits de Claude

Asegúrese de estar usando la aplicación de GitHub o una aplicación personalizada (no el usuario de Acciones), verifique que los disparadores de flujo de trabajo incluyan los eventos necesarios y confirme que los permisos de la aplicación incluyan disparadores de CI.

Errores de autenticación

Confirme que la clave API sea válida y tenga permisos suficientes. Para Bedrock/Vertex, verifique la configuración de credenciales y asegúrese de que los secretos tengan los nombres correctos en los flujos de trabajo.

Configuración avanzada

Parámetros de acción

Claude Code Action v1 utiliza una configuración simplificada:

Parámetro	Descripción	Requerido
<code>prompt</code>	Instrucciones para Claude (texto sin formato o un nombre de skill)	No*
<code>claude_args</code>	Argumentos de CLI pasados a Claude Code	No
<code>anthropic_api_key</code>	Clave API de Claude	Sí**
<code>github_token</code>	Token de GitHub para acceso a API	No
<code>trigger_phrase</code>	Frase de disparo personalizada (pre-determinado: “@claude”)	No
<code>use_bedrock</code>	Usar AWS Bedrock en lugar de Claude API	No
<code>use_vertex</code>	Usar Google Vertex AI en lugar de Claude API	No

*Prompt es opcional - cuando se omite para comentarios de problema/PR, Claude responde a la frase de disparo

**Requerido para Claude API directo, no para Bedrock/Vertex

Pasar argumentos de CLI

El parámetro `claude_args` acepta cualquier argumento de CLI de Claude Code:

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/config.json"
```

Argumentos comunes:

- `--max-turns` : Máximo de turnos de conversación (predeterminado: 10)
- `--model` : Modelo a usar (por ejemplo, `claude-sonnet-4-6`)
- `--mcp-config` : Ruta a la configuración de MCP
- `--allowed-tools` : Lista separada por comas de herramientas permitidas
- `--debug` : Habilitar salida de depuración

Métodos de integración alternativos

Aunque el comando `/install-github-app` es el enfoque recomendado, también puede:

- **Aplicación de GitHub personalizada:** Para organizaciones que necesitan nombres de usuario personalizados o flujos de autenticación personalizados. Cree su propia aplicación de GitHub con permisos requeridos (contenidos, problemas, solicitudes de extracción) y use la acción `actions/create-github-app-token` para generar tokens en sus flujos de trabajo.
- **GitHub Actions manual:** Configuración de flujo de trabajo directo para máxima flexibilidad
- **Configuración de MCP:** Carga dinámica de servidores del Protocolo de contexto del modelo

Consulte la [documentación de Claude Code Action](#) para obtener guías detalladas sobre autenticación, seguridad y configuración avanzada.

Personalizar el comportamiento de Claude

Puede configurar el comportamiento de Claude de dos formas:

1. **CLAUDE.md:** Defina estándares de codificación, criterios de revisión y reglas específicas del proyecto en un archivo `CLAUDE.md` en la raíz de su repositorio. Claude seguirá estas directrices al crear PR y responder a solicitudes. Consulte nuestra [documentación de Memory](#) para más detalles.
2. **Solicitudes personalizadas:** Use el parámetro `prompt` en el archivo de flujo de trabajo para proporcionar instrucciones específicas del flujo de trabajo. Esto le permite personalizar el comportamiento de Claude para diferentes flujos de trabajo o tareas.

Claude seguirá estas directrices al crear PR y responder a solicitudes.

Claude Code GitLab CI/CD

Aprenda a integrar Claude Code en su flujo de trabajo de desarrollo con GitLab CI/CD

Info:

Claude Code para GitLab CI/CD se encuentra actualmente en beta. Las características y funcionalidades pueden evolucionar a medida que refinamos la experiencia.

Esta integración es mantenida por GitLab. Para obtener soporte, consulte el siguiente [problema de GitLab](#).

Note:

Esta integración se basa en el [Claude Code CLI y Agent SDK](#), lo que permite el uso programático de Claude en sus trabajos de CI/CD y flujos de trabajo de automatización personalizados.

¿Por qué usar Claude Code con GitLab?

- **Creación instantánea de MR:** Describa lo que necesita, y Claude propone un MR completo con cambios y explicación
- **Implementación automatizada:** Convierta problemas en código funcional con un único comando o mención
- **Consciente del proyecto:** Claude sigue sus directrices `CLAUDE.md` y patrones de código existentes
- **Configuración simple:** Agregue un trabajo a `.gitlab-ci.yml` y una variable de CI/CD enmascarada
- **Listo para empresas:** Elija Claude API, AWS Bedrock o Google Vertex AI para cumplir con los requisitos de residencia de datos y adquisición
- **Seguro por defecto:** Se ejecuta en sus ejecutores de GitLab con su protección de rama y aprobaciones

Cómo funciona

Claude Code utiliza GitLab CI/CD para ejecutar tareas de IA en trabajos aislados y confirmar resultados a través de MRs:

1. **Orquestación impulsada por eventos:** GitLab escucha los desencadenantes elegidos (por ejemplo, un comentario que menciona `@claude` en un problema, MR o hilo de revisión). El trabajo recopila contexto del hilo y repositorio, construye indicaciones a partir de esa entrada y ejecuta Claude Code.
2. **Abstracción de proveedores:** Utilice el proveedor que se ajuste a su entorno:
 - Claude API (SaaS)
 - AWS Bedrock (acceso basado en IAM, opciones entre regiones)
 - Google Vertex AI (nativo de GCP, Federación de Identidad de Carga de Trabajo)
3. **Ejecución en sandbox:** Cada interacción se ejecuta en un contenedor con reglas estrictas de red y sistema de archivos. Claude Code aplica permisos con alcance de espacio de trabajo para restringir escrituras. Cada cambio fluye a través de un MR para que los revisores vean el diff y las aprobaciones sigan siendo aplicables.

Elija puntos finales regionales para reducir la latencia y cumplir con los requisitos de soberanía de datos mientras utiliza acuerdos en la nube existentes.

¿Qué puede hacer Claude?

Claude Code habilita flujos de trabajo de CI/CD poderosos que transforman la forma en que trabaja con código:

- Crear y actualizar MRs a partir de descripciones o comentarios de problemas
- Analizar regresiones de rendimiento y proponer optimizaciones
- Implementar características directamente en una rama, luego abrir un MR
- Corregir errores y regresiones identificados por pruebas o comentarios
- Responder a comentarios de seguimiento para iterar sobre cambios solicitados

Configuración

Configuración rápida

La forma más rápida de comenzar es agregar un trabajo mínimo a su `.gitlab-ci.yml` y establecer su clave de API como una variable enmascarada.

1. **Agregue una variable de CI/CD enmascarada**
 - Vaya a **Configuración** → **CI/CD** → **Variables**

- Agregue `ANTHROPIC_API_KEY` (enmascarada, protegida según sea necesario)

2. Agregue un trabajo de Claude a `.gitlab-ci.yml`

```
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # Ajuste las reglas para que se adapten a cómo desea desencadenar el trabajo:
  # - ejecuciones manuales
  # - eventos de solicitud de fusión
  # - desencadenadores web/API cuando un comentario contiene '@claude'
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    # Opcional: inicie un servidor MCP de GitLab si su configuración proporciona
    uno
    - /bin/gitlab-mcp-server || true
    # Utilice variables AI_FLOW_* cuando invoque a través de desencadenadores web/
    API con cargas de contexto
    - echo "$AI_FLOW_INPUT for $AI_FLOW_CONTEXT on $AI_FLOW_EVENT"
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Review this MR and implement the requested changes'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
```

Después de agregar el trabajo y su variable `ANTHROPIC_API_KEY`, pruebe ejecutando el trabajo manualmente desde **CI/CD** → **Pipelines**, o desencadénelo desde un MR para permitir que Claude proponga actualizaciones en una rama y abra un MR si es necesario.

Note:

Para ejecutar en AWS Bedrock o Google Vertex AI en lugar de Claude API, consulte la sección [Usar con AWS Bedrock y Google Vertex AI](#) a continuación para obtener instrucciones de autenticación y configuración del entorno.

Configuración manual (recomendada para producción)

Si prefiere una configuración más controlada o necesita proveedores empresariales:

1. Configure el acceso del proveedor:

- **Claude API:** Cree y almacene `ANTHROPIC_API_KEY` como una variable de CI/CD enmascarada
- **AWS Bedrock:** Configure GitLab → AWS OIDC y cree un rol de IAM para Bedrock
- **Google Vertex AI:** Configure la Federación de Identidad de Carga de Trabajo para GitLab → GCP

2. Agregue credenciales de proyecto para operaciones de API de GitLab:

- Utilice `CI_JOB_TOKEN` de forma predeterminada, o cree un Token de Acceso de Proyecto con alcance `api`
- Almacene como `GITLAB_ACCESS_TOKEN` (enmascarado) si utiliza un PAT

3. Agregue el trabajo de Claude a `.gitlab-ci.yml` (consulte los ejemplos a continuación)

4. (Opcional) Habilite desencadenadores impulsados por menciones:

- Agregue un webhook de proyecto para “Comentarios (notas)” a su escucha de eventos (si utiliza uno)
- Haga que el escucha llame a la API de desencadenador de canalización con variables como `AI_FLOW_INPUT` y `AI_FLOW_CONTEXT` cuando un comentario contenga `@claude`

Casos de uso de ejemplo

Convertir problemas en MRs

En un comentario de problema:

```
@claude implement this feature based on the issue description
```

Claude analiza el problema y la base de código, escribe cambios en una rama y abre un MR para revisión.

Obtener ayuda de implementación

En una discusión de MR:

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude propone cambios, agrega código con almacenamiento en caché apropiado y actualiza el MR.

Corregir errores rápidamente

En un comentario de problema o MR:

```
@claude fix the TypeError in the user dashboard component
```

Claude localiza el error, implementa una corrección y actualiza la rama o abre un nuevo MR.

Usar con AWS Bedrock y Google Vertex AI

Para entornos empresariales, puede ejecutar Claude Code completamente en su infraestructura en la nube con la misma experiencia de desarrollador.

AWS Bedrock

Requisitos previos

Antes de configurar Claude Code con AWS Bedrock, necesita:

1. Una cuenta de AWS con acceso a Amazon Bedrock para los modelos Claude deseados
2. GitLab configurado como proveedor de identidad OIDC en AWS IAM
3. Un rol de IAM con permisos de Bedrock y una política de confianza restringida a su proyecto/referencias de GitLab
4. Variables de CI/CD de GitLab para asumir el rol:
 - `AWS_ROLE_TO_ASSUME` (ARN del rol)

- `AWS_REGION` (región de Bedrock)

Instrucciones de configuración

Configure AWS para permitir que los trabajos de CI de GitLab asuman un rol de IAM a través de OIDC (sin claves estáticas).

Configuración requerida:

1. Habilite Amazon Bedrock y solicite acceso a sus modelos Claude objetivo
2. Cree un proveedor OIDC de IAM para GitLab si aún no está presente
3. Cree un rol de IAM confiado por el proveedor OIDC de GitLab, restringido a su proyecto y referencias protegidas
4. Adjunte permisos de menor privilegio para las API de invocación de Bedrock

Valores requeridos para almacenar en variables de CI/CD:

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

Agregue variables en Configuración → CI/CD → Variables:

```
## Para AWS Bedrock:
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

Utilice el ejemplo de trabajo de AWS Bedrock anterior para intercambiar el token de trabajo de GitLab por credenciales temporales de AWS en tiempo de ejecución.

Google Vertex AI

Requisitos previos

Antes de configurar Claude Code con Google Vertex AI, necesita:

1. Un proyecto de Google Cloud con:
 - API de Vertex AI habilitada
 - Federación de Identidad de Carga de Trabajo configurada para confiar en OIDC de GitLab
1. Una cuenta de servicio dedicada con solo los roles de Vertex AI requeridos
2. Variables de CI/CD de GitLab para WIF:
 - `GCP_WORKLOAD_IDENTITY_PROVIDER` (nombre de recurso completo)

- `GCP_SERVICE_ACCOUNT` (correo electrónico de la cuenta de servicio)

Instrucciones de configuración

Configure Google Cloud para permitir que los trabajos de CI de GitLab suplanten una cuenta de servicio a través de la Federación de Identidad de Carga de Trabajo.

Configuración requerida:

1. Habilite la API de Credenciales de IAM, la API de STS y la API de Vertex AI
2. Cree un Grupo de Identidad de Carga de Trabajo y un proveedor para OIDC de GitLab
3. Cree una cuenta de servicio dedicada con roles de Vertex AI
4. Otorgue al principal de WIF permiso para suplantar la cuenta de servicio

Valores requeridos para almacenar en variables de CI/CD:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

Agregue variables en Configuración → CI/CD → Variables:

```
## Para Google Vertex AI:
- GCP_WORKLOAD_IDENTITY_PROVIDER
- GCP_SERVICE_ACCOUNT
- CLOUD_ML_REGION (por ejemplo, us-east5)
```

Utilice el ejemplo de trabajo de Google Vertex AI anterior para autenticarse sin almacenar claves.

Ejemplos de configuración

A continuación se muestran fragmentos listos para usar que puede adaptar a su canalización.

.gitlab-ci.yml básico (Claude API)

```

stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    - /bin/gitlab-mcp-server || true
    - >

    claude
    -p "${AI_FLOW_INPUT:-'Summarize recent changes and suggest improvements'}"
    --permission-mode acceptEdits
    --allowedTools "Bash Read Edit Write mcp__gitlab"
    --debug

# Claude Code utilizará ANTHROPIC_API_KEY de las variables de CI/CD

```

Ejemplo de trabajo de AWS Bedrock (OIDC)

Requisitos previos:

- Amazon Bedrock habilitado con acceso a su modelo Claude elegido
- OIDC de GitLab configurado en AWS con un rol que confía en su proyecto y referencias de GitLab
- Rol de IAM con permisos de Bedrock (se recomienda menor privilegio)

Variables de CI/CD requeridas:

- **AWS_ROLE_TO_ASSUME** : ARN del rol de IAM para acceso a Bedrock
- **AWS_REGION** : Región de Bedrock (por ejemplo, **us-west-2**)

```

claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # Intercambie el token OIDC de GitLab por credenciales de AWS
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${CI_JOB_JWT_FILE:-/tmp/oidc_token}"
    - if [ -n "${CI_JOB_JWT_V2}" ]; then printf "%s" "${CI_JOB_JWT_V2}" >
"$AWS_WEB_IDENTITY_TOKEN_FILE"; fi
    - >
      aws sts assume-role-with-web-identity
      --role-arn "${AWS_ROLE_TO_ASSUME}"
      --role-session-name "gitlab-claude-$(date +%s)"
      --web-identity-token "file://$AWS_WEB_IDENTITY_TOKEN_FILE"
      --duration-seconds 3600 > /tmp/aws_creds.json
    - export AWS_ACCESS_KEY_ID="$(jq -r .Credentials.AccessKeyId /tmp/
aws_creds.json)"
    - export AWS_SECRET_ACCESS_KEY="$(jq -r .Credentials.SecretAccessKey /tmp/
aws_creds.json)"
    - export AWS_SESSION_TOKEN="$(jq -r .Credentials.SessionToken /tmp/
aws_creds.json)"
  script:
    - /bin/gitlab-mcp-server || true
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Implement the requested changes and open an MR'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    AWS_REGION: "us-west-2"

```

Note:

Los ID de modelo para Bedrock incluyen prefijos específicos de región (por ejemplo, `us.anthropic.claude-sonnet-4-6`). Pase el modelo deseado a través de su configuración de trabajo o indicación si su flujo de trabajo lo admite.

Ejemplo de trabajo de Google Vertex AI (Federación de Identidad de Carga de Trabajo)

Requisitos previos:

- API de Vertex AI habilitada en su proyecto de GCP
- Federación de Identidad de Carga de Trabajo configurada para confiar en OIDC de GitLab
- Una cuenta de servicio con permisos de Vertex AI

Variables de CI/CD requeridas:

- `GCP_WORKLOAD_IDENTITY_PROVIDER` : Nombre de recurso completo del proveedor
- `GCP_SERVICE_ACCOUNT` : Correo electrónico de la cuenta de servicio
- `CLOUD_ML_REGION` : Región de Vertex (por ejemplo, `us-east5`)

```

claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # Autentíquese en Google Cloud a través de WIF (sin claves descargadas)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${GCP_WORKLOAD_IDENTITY_PROVIDER}",
        "subject_token_type": "urn:ietf:params:oauth:token-type:jwt",
        "service_account_impersonation_url": "https://
iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/${
GCP_SERVICE_ACCOUNT}:generateAccessToken",
        "token_url": "https://sts.googleapis.com/v1/token"
      }
      EOF
    )
    - gcloud config set project "$(gcloud projects list --
format='value(projectId)' --filter="name:${CI_PROJECT_NAMESPACE}" | head -n1)" ||
true
  script:
    - /bin/gitlab-mcp-server || true
    - >
      CLOUD_ML_REGION="${CLOUD_ML_REGION:-us-east5}"
      claude
      -p "${AI_FLOW_INPUT:-'Review and update code as requested'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    CLOUD_ML_REGION: "us-east5"

```

Note:

Con la Federación de Identidad de Carga de Trabajo, no necesita almacenar claves de cuenta de servicio. Utilice condiciones de confianza específicas del repositorio y cuentas de servicio con menor privilegio.

Mejores prácticas

Configuración de `CLAUDE.md`

Cree un archivo `CLAUDE.md` en la raíz del repositorio para definir estándares de codificación, criterios de revisión y reglas específicas del proyecto. Claude lee este archivo durante las ejecuciones y sigue sus convenciones al proponer cambios.

Consideraciones de seguridad

Nunca confirme claves de API o credenciales en la nube en su repositorio.

Siempre utilice variables de CI/CD de GitLab:

- Agregue `ANTHROPIC_API_KEY` como una variable enmascarada (y protéjala si es necesario)
- Utilice OIDC específico del proveedor donde sea posible (sin claves de larga duración)
- Limite los permisos de trabajo y la salida de red
- Revise los MRs de Claude como cualquier otro colaborador

Optimización del rendimiento

- Mantenga `CLAUDE.md` enfocado y conciso
- Proporcione descripciones claras de problemas/MR para reducir iteraciones
- Configure tiempos de espera de trabajo sensatos para evitar ejecuciones descontroladas
- Almacene en caché npm e instalaciones de paquetes en ejecutores donde sea posible

Costos de CI

Cuando utiliza Claude Code con GitLab CI/CD, tenga en cuenta los costos asociados:

- **Tiempo de ejecución de GitLab:**
 - Claude se ejecuta en sus ejecutores de GitLab y consume minutos de cálculo
 - Consulte la facturación de ejecutores de su plan de GitLab para obtener detalles

• Costos de API:

- Cada interacción de Claude consume tokens según el tamaño de la indicación y la respuesta
- El uso de tokens varía según la complejidad de la tarea y el tamaño de la base de código
- Consulte [Precios de Anthropic](#) para obtener detalles

• Consejos de optimización de costos:

- Utilice comandos específicos de `@claude` para reducir turnos innecesarios
- Establezca valores apropiados de `max_turns` y tiempo de espera de trabajo
- Limite la concurrencia para controlar ejecuciones paralelas

Seguridad y gobernanza

- Cada trabajo se ejecuta en un contenedor aislado con acceso de red restringido
- Los cambios de Claude fluyen a través de MRs para que los revisores vean cada diff
- Las reglas de protección de rama y aprobación se aplican al código generado por IA
- Claude Code utiliza permisos con alcance de espacio de trabajo para restringir escrituras
- Los costos permanecen bajo su control porque usted proporciona sus propias credenciales de proveedor

Solución de problemas

Claude no responde a comandos `@claude`

- Verifique que su canalización se esté desencadenando (manualmente, evento de MR o a través de un escucha de eventos de nota/webhook)
- Asegúrese de que las variables de CI/CD (`ANTHROPIC_API_KEY` o configuración del proveedor en la nube) estén presentes y no enmascaradas
- Compruebe que el comentario contiene `@claude` (no `/claude`) y que su desencadenador de mención está configurado

El trabajo no puede escribir comentarios ni abrir MRs

- Asegúrese de que `CI_JOB_TOKEN` tenga permisos suficientes para el proyecto, o utilice un Token de Acceso de Proyecto con alcance `api`
- Compruebe que la herramienta `mcp__gitlab` esté habilitada en `--allowedTools`

- Confirme que el trabajo se ejecuta en el contexto del MR o tiene suficiente contexto a través de variables `AI_FLOW_*`

Errores de autenticación

- **Para Claude API:** Confirme que `ANTHROPIC_API_KEY` es válida y no ha expirado
- **Para Bedrock/Vertex:** Verifique la configuración de OIDC/WIF, la suplantación de rol y los nombres secretos; confirme la disponibilidad de región y modelo

Configuración avanzada

Parámetros y variables comunes

Claude Code admite estas entradas comúnmente utilizadas:

- `prompt` / `prompt_file` : Proporcione instrucciones en línea (`-p`) o a través de un archivo
- `max_turns` : Limite el número de iteraciones de ida y vuelta
- `timeout_minutes` : Limite el tiempo total de ejecución
- `ANTHROPIC_API_KEY` : Requerido para Claude API (no se utiliza para Bedrock/Vertex)
- Entorno específico del proveedor: `AWS_REGION` , variables de proyecto/región para Vertex

Note:

Las banderas y parámetros exactos pueden variar según la versión de `@anthropic-ai/claude-code` . Ejecute `claude --help` en su trabajo para ver las opciones admitidas.

Personalización del comportamiento de Claude

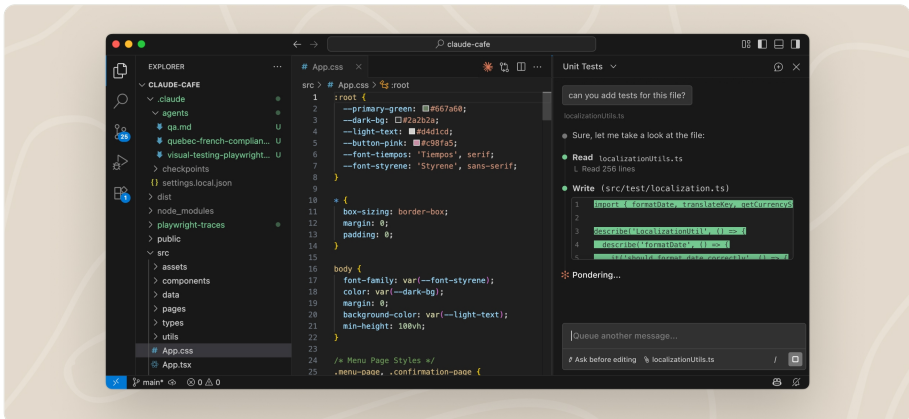
Puede guiar a Claude de dos formas principales:

1. **CLAUDE.md:** Defina estándares de codificación, requisitos de seguridad y convenciones de proyecto. Claude lee esto durante las ejecuciones y sigue sus reglas.
2. **Indicaciones personalizadas:** Pase instrucciones específicas de tareas a través de `prompt` / `prompt_file` en el trabajo. Utilice diferentes indicaciones para diferentes trabajos (por ejemplo, revisión, implementación, refactorización).

Part 8: IDE & Platform Integration

Usar Claude Code en VS Code

Instala y configura la extensión Claude Code para VS Code. Obtén asistencia de co-dificación con IA con diffs en línea, menciones @, revisión de planes y atajos de teclado.



Editor de VS Code con el panel de extensión Claude Code abierto en el lado derecho, mostrando una conversación con Claude

La extensión de VS Code proporciona una interfaz gráfica nativa para Claude Code, integrada directamente en tu IDE. Esta es la forma recomendada de usar Claude Code en VS Code.

Con la extensión, puedes revisar y editar los planes de Claude antes de aceptarlos, aceptar automáticamente ediciones a medida que se realizan, mencionar archivos con rangos de líneas específicas desde tu selección, acceder al historial de conversaciones y abrir múltiples conversaciones en pestañas o ventanas separadas.

Requisitos previos

Antes de instalar, asegúrate de tener:

- VS Code 1.98.0 o superior

- Una cuenta de Anthropic (iniciarás sesión cuando abras la extensión por primera vez). Si estás usando un proveedor de terceros como Amazon Bedrock o Google Vertex AI, consulta [Usar proveedores de terceros](#) en su lugar.

Tip:

La extensión incluye la CLI (interfaz de línea de comandos), a la que puedes acceder desde la terminal integrada de VS Code para funciones avanzadas. Consulta [Extensión de VS Code frente a Claude Code CLI](#) para obtener más detalles.

Instalar la extensión

Haz clic en el enlace de tu IDE para instalar directamente:

- [Instalar para VS Code](#)
- [Instalar para Cursor](#)

O en VS Code, presiona **Cmd+Shift+X** (Mac) o **Ctrl+Shift+X** (Windows/Linux) para abrir la vista de Extensiones, busca “Claude Code” y haz clic en **Instalar**.

Note:

Si la extensión no aparece después de la instalación, reinicia VS Code o ejecuta “Developer: Reload Window” desde la Paleta de comandos.

Comenzar

Una vez instalada, puedes comenzar a usar Claude Code a través de la interfaz de VS Code:

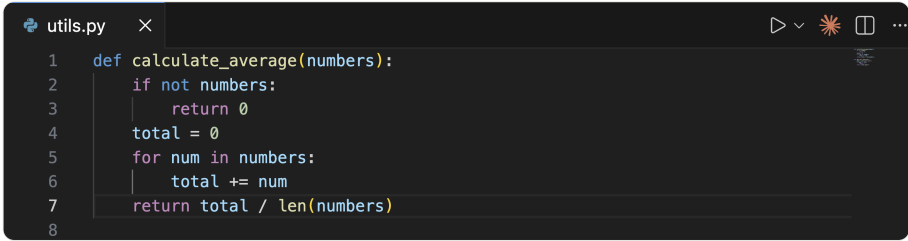
Step 1: Abre el panel de Claude Code

En todo VS Code, el icono Spark indica Claude Code:



Icono Spark

La forma más rápida de abrir Claude es hacer clic en el icono Spark en la **Barra de herramientas del editor** (esquina superior derecha del editor). El icono solo aparece cuando tienes un archivo abierto.



Editor de VS Code mostrando el icono Spark en la Barra de herramientas del editor

Otras formas de abrir Claude Code:

- **Barra de actividades:** haz clic en el icono Spark en la barra lateral izquierda para abrir la lista de sesiones. Haz clic en cualquier sesión para abrirla como una pestaña de editor completa, o inicia una nueva. Este icono siempre es visible en la Barra de actividades.
- **Paleta de comandos:** `Cmd+Shift+P` (Mac) o `Ctrl+Shift+P` (Windows/Linux), escribe “Claude Code” y selecciona una opción como “Abrir en Nueva Pestaña”
- **Barra de estado:** haz clic en **Claude Code** en la esquina inferior derecha de la ventana. Esto funciona incluso cuando no hay ningún archivo abierto.

Cuando abres el panel por primera vez, aparece una lista de verificación **Aprender Claude Code**. Trabaja en cada elemento haciendo clic en **Mostrarme**, o descártalo con la X. Para reabrirlo más tarde, desactiva **Ocultar incorporación** en la configuración de VS Code en Extensiones → Claude Code.

Puedes arrastrar el panel de Claude para reposicionarlo en cualquier lugar de VS Code. Consulta [Personaliza tu flujo de trabajo](#) para obtener más detalles.

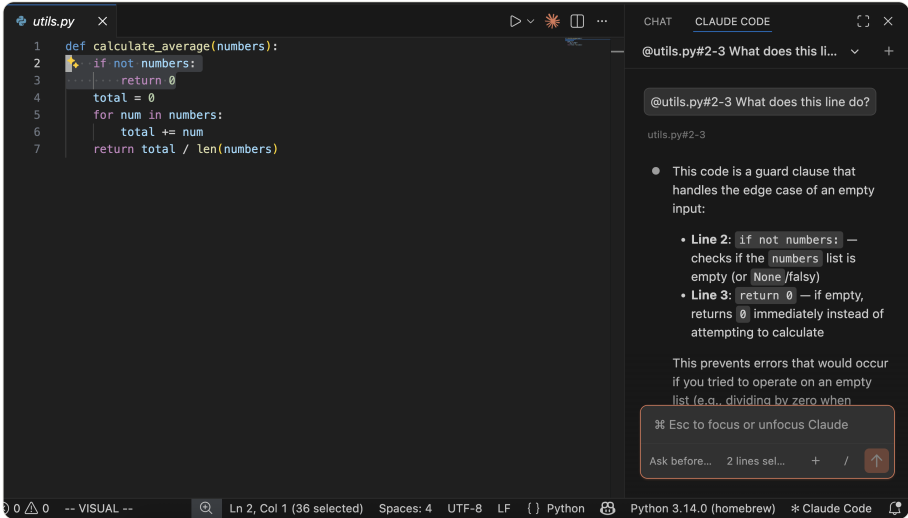
Step 2: Envía un mensaje

Pídele a Claude que te ayude con tu código o archivos, ya sea explicando cómo funciona algo, depurando un problema o realizando cambios.

Tip:

Claude ve automáticamente tu texto seleccionado. Presiona `Option+K` (Mac) / `Alt+K` (Windows/Linux) para también insertar una referencia de mención @ (como `@file.ts#5-10`) en tu mensaje.

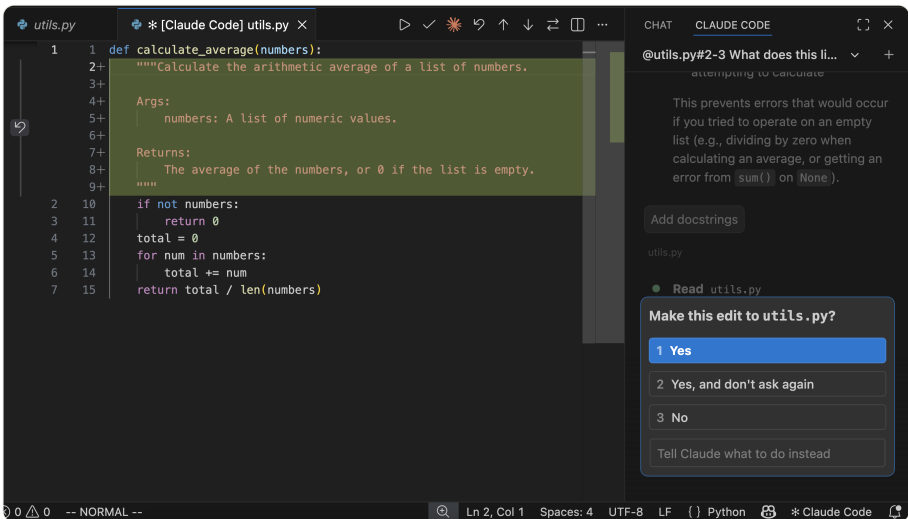
Aquí hay un ejemplo de cómo hacer una pregunta sobre una línea particular en un archivo:



Editor de VS Code con las líneas 2-3 seleccionadas en un archivo Python, y el panel de Claude Code mostrando una pregunta sobre esas líneas con una referencia de mención @

Step 3: Revisa los cambios

Cuando Claude quiere editar un archivo, muestra una comparación lado a lado del original y los cambios propuestos, luego solicita permiso. Puedes aceptar, rechazar o decirle a Claude qué hacer en su lugar.



VS Code mostrando un diff de los cambios propuestos por Claude con un mensaje de permiso preguntando si realizar la edición

Para más ideas sobre lo que puedes hacer con Claude Code, consulta [Flujos de trabajo comunes](#).

Tip:

Ejecuta “Claude Code: Open Walkthrough” desde la Paleta de comandos para un tour guiado de los conceptos básicos.

Usa el cuadro de mensaje

El cuadro de mensaje admite varias funciones:

- **Modos de permiso:** haz clic en el indicador de modo en la parte inferior del cuadro de mensaje para cambiar de modo. En modo normal, Claude solicita permiso antes de cada acción. En Plan Mode, Claude describe lo que hará y espera aprobación antes de realizar cambios. VS Code abre automáticamente el plan como un documento markdown completo donde puedes agregar comentarios en línea para dar retroalimentación antes de que Claude comience. En modo de aceptación automática, Claude realiza ediciones sin preguntar. Establece el valor predeterminado en la configuración de VS Code en `claudeCode.initialPermissionMode`.
- **Menú de comandos:** haz clic en `/` o escribe `/` para abrir el menú de comandos. Las opciones incluyen adjuntar archivos, cambiar modelos, alternar pensamiento extendido y ver el uso del plan (`/usage`). La sección Personalizar proporciona acceso a MCP servers, hooks, memoria, permisos y plugins. Los elementos con un icono de terminal se abren en la terminal integrada.
- **Indicador de contexto:** el cuadro de mensaje muestra cuánto de la ventana de contexto de Claude estás usando. Claude se compacta automáticamente cuando es necesario, o puedes ejecutar `/compact` manualmente.
- **Pensamiento extendido:** permite que Claude dedique más tiempo a razonar sobre problemas complejos. Actívalo a través del menú de comandos (`/`). Consulta [Pensamiento extendido](#) para obtener más detalles.
- **Entrada multilínea:** presiona `Shift+Enter` para agregar una nueva línea sin enviar. Esto también funciona en la entrada de texto libre “Otro” de los diálogos de preguntas.

Referencia de archivos y carpetas

Usa menciones `@` para darle a Claude contexto sobre archivos o carpetas específicas. Cuando escribes `@` seguido de un nombre de archivo o carpeta, Claude lee ese contenido y puede responder preguntas sobre él o realizar cambios en él. Claude Code admite coincidencia difusa, por lo que puedes escribir nombres parciales para encontrar lo que necesitas:

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

Para archivos PDF grandes, puedes pedirle a Claude que lea páginas específicas en lugar del archivo completo: una sola página, un rango como páginas 1-10, o un rango abierto como página 3 en adelante.

Cuando seleccionas texto en el editor, Claude puede ver tu código resaltado automáticamente. El pie de página del cuadro de mensaje muestra cuántas líneas están seleccionadas. Presiona `Option+K` (Mac) / `Alt+K` (Windows/Linux) para insertar una mención `@` con la ruta del archivo y los números de línea (por ejemplo, `@app.ts#5-10`). Haz clic en el indicador de selección para alternar si Claude puede ver tu texto resaltado: el icono de barra diagonal significa que la selección está oculta para Claude.

También puedes mantener presionado `Shift` mientras arrastras archivos al cuadro de mensaje para agregarlos como adjuntos. Haz clic en la X en cualquier adjunto para eliminarlo del contexto.

Reanuda conversaciones pasadas

Haz clic en el menú desplegable en la parte superior del panel de Claude Code para acceder a tu historial de conversaciones. Puedes buscar por palabra clave o explorar por tiempo (Hoy, Ayer, Últimos 7 días, etc.). Haz clic en cualquier conversación para reanudarla con el historial de mensajes completo. Pasa el cursor sobre una sesión para revelar acciones de cambio de nombre y eliminación: cambia el nombre para darle un título descriptivo, o elimina para borrarlo de la lista. Para más información sobre cómo reanudar sesiones, consulta [Flujos de trabajo comunes](#).

Reanuda sesiones remotas desde Claude.ai

Si usas [Claude Code en la web](#), puedes reanudar esas sesiones remotas directamente en VS Code. Esto requiere iniciar sesión con **Claude.ai Subscription**, no Anthropic Console.

Step 1: Abre Conversaciones pasadas

Haz clic en el menú desplegable **Conversaciones pasadas** en la parte superior del panel de Claude Code.

Step 2: Selecciona la pestaña Remota

El diálogo muestra dos pestañas: Local y Remota. Haz clic en **Remota** para ver sesiones desde claude.ai.

Step 3: Selecciona una sesión para reanudar

Explora o busca tus sesiones remotas. Haz clic en cualquier sesión para descargarla y continuar la conversación localmente.

Note:

Solo las sesiones web iniciadas con un repositorio de GitHub aparecen en la pestaña Remota. Reanudar carga el historial de conversaciones localmente; los cambios no se sincronizan de vuelta a claude.ai.

Personaliza tu flujo de trabajo

Una vez que estés en funcionamiento, puedes reposicionar el panel de Claude, ejecutar múltiples sesiones o cambiar al modo terminal.

Elige dónde vive Claude

Puedes arrastrar el panel de Claude para reposicionarlo en cualquier lugar de VS Code. Agarra la pestaña o barra de título del panel y arrástralo a:

- **Barra lateral secundaria:** el lado derecho de la ventana. Mantiene a Claude visible mientras codificas.
- **Barra lateral principal:** la barra lateral izquierda con iconos para Explorador, Búsqueda, etc.
- **Área del editor:** abre Claude como una pestaña junto a tus archivos. Útil para tareas secundarias.

Tip:

Usa la barra lateral para tu sesión principal de Claude y abre pestañas adicionales para tareas secundarias. Claude recuerda tu ubicación preferida. El icono de lista de sesiones de la Barra de actividades es separado del panel de Claude: la lista de sesiones siempre es visible en la Barra de actividades, mientras que el icono del panel de Claude solo aparece allí cuando el panel está acoplado a la barra lateral izquierda.

Ejecuta múltiples conversaciones

Usa **Abrir en Nueva Pestaña** u **Abrir en Nueva Ventana** desde la Paleta de comandos para iniciar conversaciones adicionales. Cada conversación mantiene su propio historial y contexto, permitiéndote trabajar en diferentes tareas en paralelo.

Cuando usas pestañas, un pequeño punto de color en el icono spark indica el estado: azul significa que hay una solicitud de permiso pendiente, naranja significa que Claude terminó mientras la pestaña estaba oculta.

Cambia al modo terminal

De forma predeterminada, la extensión abre un panel de chat gráfico. Si prefieres la interfaz de estilo CLI, abre la [configuración Usar terminal](#) y marca la casilla.

También puedes abrir la configuración de VS Code (`Cmd+,` en Mac o `Ctrl+,` en Windows/Linux), ve a Extensiones → Claude Code y marca **Usar terminal**.

Gestiona plugins

La extensión de VS Code incluye una interfaz gráfica para instalar y gestionar [plugins](#). Escribe `/plugins` en el cuadro de mensaje para abrir la interfaz **Gestionar plugins**.

Instala plugins

El diálogo de plugins muestra dos pestañas: **Plugins** y **Marketplaces**.

En la pestaña Plugins:

- Los **plugins instalados** aparecen en la parte superior con interruptores de alternancia para habilitarlos o deshabilitarlos
- Los **plugins disponibles** de tus marketplaces configurados aparecen a continuación
- Busca para filtrar plugins por nombre o descripción
- Haz clic en **Instalar** en cualquier plugin disponible

Cuando instalas un plugin, elige el alcance de instalación:

- **Instalar para ti:** disponible en todos tus proyectos (alcance de usuario)
- **Instalar para este proyecto:** compartido con colaboradores del proyecto (alcance del proyecto)
- **Instalar localmente:** solo para ti, solo en este repositorio (alcance local)

Gestiona marketplaces

Cambia a la pestaña **Marketplaces** para agregar o eliminar fuentes de plugins:

- Ingresa un repositorio de GitHub, URL o ruta local para agregar un nuevo marketplace
- Haz clic en el icono de actualización para actualizar la lista de plugins de un marketplace
- Haz clic en el icono de papelera para eliminar un marketplace

Después de realizar cambios, un banner te solicita que reinicies Claude Code para aplicar las actualizaciones.

Note:

La gestión de plugins en VS Code usa los mismos comandos CLI bajo el capó. Los plugins y marketplaces que configures en la extensión también están disponibles en la CLI, y viceversa.

Para más información sobre el sistema de plugins, consulta [Plugins](#) y [Marketplaces de plugins](#).

Automatiza tareas del navegador con Chrome

Conecta Claude a tu navegador Chrome para probar aplicaciones web, depurar con registros de consola y automatizar flujos de trabajo del navegador sin salir de VS Code. Esto requiere la [extensión Claude in Chrome](#) versión 1.0.36 o superior.

Escribe `@browser` en el cuadro de mensaje seguido de lo que quieres que Claude haga:

```
@browser go to localhost:3000 and check the console for errors
```

También puedes abrir el menú de adjuntos para seleccionar herramientas específicas del navegador como abrir una nueva pestaña o leer el contenido de la página.

Claude abre nuevas pestañas para tareas del navegador y comparte el estado de inicio de sesión de tu navegador, por lo que puede acceder a cualquier sitio en el que ya hayas iniciado sesión.

Para instrucciones de configuración, la lista completa de capacidades y solución de problemas, consulta [Usar Claude Code con Chrome](#).

Comandos y atajos de teclado de VS Code

Abre la Paleta de comandos (`Cmd+Shift+P` en Mac o `Ctrl+Shift+P` en Windows/Linux) y escribe “Claude Code” para ver todos los comandos de VS Code disponibles para la extensión Claude Code.

Algunos atajos dependen de qué panel esté “enfocado” (recibiendo entrada de teclado). Cuando tu cursor está en un archivo de código, el editor está enfocado. Cuando tu cursor está en el cuadro de mensaje de Claude, Claude está enfocado. Usa `Cmd+Esc` / `Ctrl+Esc` para alternar entre ellos.

Note:

Estos son comandos de VS Code para controlar la extensión. No todos los comandos integrados de Claude Code están disponibles en la extensión. Consulta [Extensión de VS Code frente a Claude Code CLI](#) para obtener más detalles.

Comando	Atajo	Descripción
Focus Input	<code>Cmd+Esc</code> (Mac) / <code>Ctrl+Esc</code> (Windows/Linux)	Alternar el enfoque entre el editor y Claude
Open in Side Bar	-	Abre Claude en la barra lateral izquierda
Open in Terminal	-	Abre Claude en modo terminal
Open in New Tab	<code>Cmd+Shift+Esc</code> (Mac) / <code>Ctrl+Shift+Esc</code> (Windows/Linux)	Abre una nueva conversación como una pestaña del editor
Open in New Window	-	Abre una nueva conversación en una ventana separada
New Conversation	<code>Cmd+N</code> (Mac) / <code>Ctrl+N</code> (Windows/Linux)	Inicia una nueva conversación (requiere que Claude esté enfocado)
Insert @-Mention Reference	<code>Option+K</code> (Mac) / <code>Alt+K</code> (Windows/Linux)	Inserta una referencia al archivo actual y la selección (requiere que el editor esté enfocado)

Comando	Atajo	Descripción
Show Logs	-	Ver registros de depuración de la extensión
Logout	-	Cierra sesión de tu cuenta de Anthropic

Configura la configuración

La extensión tiene dos tipos de configuración:

- **Configuración de extensión** en VS Code: controla el comportamiento de la extensión dentro de VS Code. Abre con `Cmd+,` (Mac) o `Ctrl+,` (Windows/Linux), luego ve a Extensiones → Claude Code. También puedes escribir `/` y seleccionar **General Config** para abrir la configuración.
- **Configuración de Claude Code** en `~/.claude/settings.json` : compartida entre la extensión y la CLI. Úsala para comandos permitidos, variables de entorno, hooks y MCP servers. Consulta [Configuración](#) para obtener más detalles.

Tip:

Agrega `"$schema": "https://json.schemastore.org/claude-code-settings.json"` a tu `settings.json` para obtener autocompletado y validación en línea para todas las configuraciones disponibles directamente en VS Code.

Configuración de extensión

Configuración	Predeterminado	Descripción
<code>selectedModel</code>	<code>default</code>	Modelo para nuevas conversaciones. Cambia por sesión con <code>/model</code> .
<code>useTerminal</code>	<code>false</code>	Lanza Claude en modo terminal en lugar de panel gráfico
<code>initialPermissionMode</code>	<code>default</code>	Controla mensajes de aprobación: <code>default</code> (preguntar cada vez), <code>planned</code> , <code>acceptEdits</code> o <code>bypassPermissions</code>

Configuración	Predeterminado	Descripción
<code>preferredLocation</code>	<code>panel</code>	Dónde se abre Claude: <code>sidebar</code> (derecha) o <code>panel</code> (nueva pestaña)
<code>autosave</code>	<code>true</code>	Guarda automáticamente archivos antes de que Claude los lea o escriba
<code>useCtrlEnterToSend</code>	<code>false</code>	Usa Ctrl/Cmd+Enter en lugar de Enter para enviar mensajes
<code>enableNewConversationShortcut</code>	<code>true</code>	Habilita Cmd/Ctrl+N para iniciar una nueva conversación
<code>hideOnboarding</code>	<code>false</code>	Oculto la lista de verificación de incorporación (icono de gorro de graduación)
<code>respectGitIgnore</code>	<code>true</code>	Excluye patrones de .gitignore de búsquedas de archivos
<code>environmentVariables</code>	<code>[]</code>	Establece variables de entorno para el proceso de Claude. Usa la configuración de Claude Code en su lugar para configuración compartida.
<code>disableLoginPrompt</code>	<code>false</code>	Omite mensajes de autenticación (para configuraciones de proveedores de terceros)
<code>allowDangerouslySkipPermissions</code>	<code>false</code>	Omite todos los mensajes de permiso. Usa con extrema precaución.

Configuración	Predeterminado	Descripción
<code>claudeProcessWrapper</code>	-	Ruta ejecutable utilizada para lanzar el proceso de Claude

Extensión de VS Code frente a Claude Code CLI

Claude Code está disponible tanto como una extensión de VS Code (panel gráfico) como una CLI (interfaz de línea de comandos en la terminal). Algunas funciones solo están disponibles en la CLI. Si necesitas una función solo de CLI, ejecuta `claude` en la terminal integrada de VS Code.

Función	CLI	Extensión de VS Code
Comandos y skills	Todos	Subconjunto (escribe <code>/</code> para ver disponibles)
Configuración de MCP server	Sí	Parcial (agrega servidores a través de CLI; gestiona servidores existentes con <code>/mcp</code> en el panel de chat)
Checkpoints	Sí	Sí
Atajo bash <code>!</code>	Sí	No
Autocompletado de pestañas	Sí	No

Retrocede con checkpoints

La extensión de VS Code admite checkpoints, que rastrean las ediciones de archivos de Claude y te permiten retroceder a un estado anterior. Pasa el cursor sobre cualquier mensaje para revelar el botón de retroceso, luego elige entre tres opciones:

- **Bifurcar conversación desde aquí:** inicia una nueva rama de conversación desde este mensaje mientras mantienes todos los cambios de código intactos
- **Retroceder código a aquí:** revierte los cambios de archivo a este punto en la conversación mientras mantienes el historial de conversación completo
- **Bifurcar conversación y retroceder código:** inicia una nueva rama de conversación y revierte los cambios de archivo a este punto

Para obtener detalles completos sobre cómo funcionan los checkpoints y sus limitaciones, consulta [Checkpointing](#).

Ejecuta CLI en VS Code

Para usar la CLI mientras permaneces en VS Code, abre la terminal integrada (`Ctrl+`` en Windows/Linux o `Cmd+`` en Mac) y ejecuta `claude`. La CLI se integra automáticamente con tu IDE para funciones como visualización de diffs y compartir diagnósticos.

Si usas una terminal externa, ejecuta `/ide` dentro de Claude Code para conectarlo a VS Code.

Cambia entre extensión y CLI

La extensión y la CLI comparten el mismo historial de conversaciones. Para continuar una conversación de extensión en la CLI, ejecuta `claude --resume` en la terminal. Esto abre un selector interactivo donde puedes buscar y seleccionar tu conversación.

Incluye salida de terminal en mensajes

Referencia la salida de terminal en tus mensajes usando `@terminal:name` donde `name` es el título de la terminal. Esto permite que Claude vea la salida del comando, mensajes de error o registros sin copiar y pegar.

Monitorea procesos en segundo plano

Cuando Claude ejecuta comandos de larga duración, la extensión muestra el progreso en la barra de estado. Sin embargo, la visibilidad de tareas en segundo plano es limitada en comparación con la CLI. Para mejor visibilidad, haz que Claude genere el comando para que puedas ejecutarlo en la terminal integrada de VS Code.

Conecta con herramientas externas con MCP

Los servidores MCP (Model Context Protocol) dan a Claude acceso a herramientas externas, bases de datos y APIs.

Para agregar un servidor MCP, abre la terminal integrada (`Ctrl+`` o `Cmd+``) y ejecuta:

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Una vez configurado, pídele a Claude que use las herramientas (por ejemplo, “Review PR #456”).

Para gestionar servidores MCP sin salir de VS Code, escribe `/mcp` en el panel de chat. El diálogo de gestión de MCP te permite habilitar o deshabilitar servidores, reconectarse a un servidor y gestionar la autenticación OAuth. Consulta la [documentación de MCP](#) para servidores disponibles.

Trabaja con git

Claude Code se integra con git para ayudarte con flujos de trabajo de control de versiones directamente en VS Code. Pídele a Claude que confirme cambios, cree solicitudes de extracción o trabaje en diferentes ramas.

Crea commits y solicitudes de extracción

Claude puede preparar cambios, escribir mensajes de commit y crear solicitudes de extracción basadas en tu trabajo:

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

Al crear solicitudes de extracción, Claude genera descripciones basadas en los cambios de código reales y puede agregar contexto sobre pruebas o decisiones de implementación.

Usa git worktrees para tareas paralelas

Usa el indicador `--worktree` (`-w`) para iniciar Claude en un worktree aislado con sus propios archivos y rama:

```
claude --worktree feature-auth
```

Cada worktree mantiene un estado de archivo independiente mientras comparte el historial de git. Esto evita que las instancias de Claude interfieran entre sí cuando trabajan en diferentes tareas. Para más detalles, consulta [Ejecuta sesiones paralelas de Claude Code con Git worktrees](#).

Usa proveedores de terceros

De forma predeterminada, Claude Code se conecta directamente a la API de Anthropic. Si tu organización usa Amazon Bedrock, Google Vertex AI o Microsoft Foundry para acceder a Claude, configura la extensión para usar tu proveedor en su lugar:

Step 1: Desactiva el mensaje de inicio de sesión

Abre la [configuración Desactivar mensaje de inicio de sesión](#) y marca la casilla.

También puedes abrir la configuración de VS Code (`Cmd+`, en Mac o `Ctrl+`, en Windows/Linux), buscar “Claude Code login” y marcar **Desactivar mensaje de inicio de sesión**.

Step 2: Configura tu proveedor

Sigue la guía de configuración para tu proveedor:

- [Claude Code en Amazon Bedrock](#)
- [Claude Code en Google Vertex AI](#)
- [Claude Code en Microsoft Foundry](#)

Estas guías cubren la configuración de tu proveedor en `~/.claude/settings.json`, lo que garantiza que tus configuraciones se compartan entre la extensión de VS Code y la CLI.

Seguridad y privacidad

Tu código permanece privado. Claude Code procesa tu código para proporcionar asistencia pero no lo usa para entrenar modelos. Para obtener detalles sobre el manejo de datos y cómo optar por no participar en el registro, consulta [Datos y privacidad](#).

Con permisos de edición automática habilitados, Claude Code puede modificar archivos de configuración de VS Code (como `settings.json` o `tasks.json`) que VS Code puede ejecutar automáticamente. Para reducir el riesgo al trabajar con código no confiable:

- Habilita [Modo restringido de VS Code](#) para espacios de trabajo no confiables
- Usa modo de aprobación manual en lugar de aceptación automática para ediciones
- Revisa cuidadosamente los cambios antes de aceptarlos

Soluciona problemas comunes

La extensión no se instala

- Asegúrate de tener una versión compatible de VS Code (1.98.0 o posterior)
- Verifica que VS Code tenga permiso para instalar extensiones
- Intenta instalar directamente desde [VS Code Marketplace](#)

El icono Spark no es visible

El icono Spark aparece en la **Barra de herramientas del editor** (esquina superior derecha del editor) cuando tienes un archivo abierto. Si no lo ves:

1. **Abre un archivo:** El icono requiere que un archivo esté abierto. Solo tener una carpeta abierta no es suficiente.
2. **Verifica la versión de VS Code:** Requiere 1.98.0 o superior (Ayuda → Acerca de)
3. **Reinicia VS Code:** Ejecuta “Developer: Reload Window” desde la Paleta de comandos
4. **Desactiva extensiones conflictivas:** Desactiva temporalmente otras extensiones de IA (Cline, Continue, etc.)
5. **Verifica la confianza del espacio de trabajo:** La extensión no funciona en Modo restringido

Alternativamente, haz clic en “Claude Code” en la **Barra de estado** (esquina inferior derecha). Esto funciona incluso sin un archivo abierto. También puedes usar la **Paleta de comandos** (`Cmd+Shift+P` / `Ctrl+Shift+P`) y escribir “Claude Code”.

Claude Code nunca responde

Si Claude Code no responde a tus mensajes:

1. **Verifica tu conexión a Internet:** Asegúrate de tener una conexión a Internet estable
2. **Inicia una nueva conversación:** Intenta iniciar una conversación nueva para ver si el problema persiste
3. **Prueba la CLI:** Ejecuta `claude` desde la terminal para ver si obtienes mensajes de error más detallados

Si los problemas persisten, [abre un problema en GitHub](#) con detalles sobre el error.

Desinstala la extensión

Para desinstalar la extensión Claude Code:

1. Abre la vista de Extensiones (`Cmd+Shift+X` en Mac o `Ctrl+Shift+X` en Windows/Linux)
2. Busca “Claude Code”
3. Haz clic en **Desinstalar**

Para también eliminar datos de extensión y restablecer toda la configuración:

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

Para obtener ayuda adicional, consulta la [guía de solución de problemas](#).

Próximos pasos

Ahora que tienes Claude Code configurado en VS Code:

- [Explora flujos de trabajo comunes](#) para aprovechar al máximo Claude Code
- [Configura MCP servers](#) para extender las capacidades de Claude con herramientas externas. Agrega servidores usando la CLI, luego gestiónalos con `/mcp` en el panel de chat.
- [Configura la configuración de Claude Code](#) para personalizar comandos permitidos, hooks y más. Estas configuraciones se comparten entre la extensión y la CLI.

JetBrains IDEs

Utiliza Claude Code con JetBrains IDEs incluyendo IntelliJ, PyCharm, WebStorm y más

Claude Code se integra con JetBrains IDEs a través de un plugin dedicado, proporcionando características como visualización de diferencias interactivas, compartición de contexto de selección y más.

IDEs Compatibles

El plugin de Claude Code funciona con la mayoría de JetBrains IDEs, incluyendo:

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

Características

- **Lanzamiento rápido:** Utiliza `Cmd+Esc` (Mac) o `Ctrl+Esc` (Windows/Linux) para abrir Claude Code directamente desde tu editor, o haz clic en el botón de Claude Code en la interfaz
- **Visualización de diferencias:** Los cambios de código se pueden mostrar directamente en el visor de diferencias del IDE en lugar de la terminal
- **Contexto de selección:** La selección/pestaña actual en el IDE se comparte automáticamente con Claude Code
- **Atajos de referencia de archivos:** Utiliza `Cmd+Option+K` (Mac) o `Alt+Ctrl+K` (Linux/Windows) para insertar referencias de archivos (por ejemplo, `@File#L1-99`)
- **Compartición de diagnósticos:** Los errores de diagnóstico (lint, sintaxis, etc.) del IDE se comparten automáticamente con Claude mientras trabajas

Instalación

Instalación desde Marketplace

Busca e instala el [plugin de Claude Code](#) desde el marketplace de JetBrains y reinicia tu IDE.

Si aún no has instalado Claude Code, consulta [nuestra guía de inicio rápido](#) para obtener instrucciones de instalación.

Note:

Después de instalar el plugin, es posible que necesites reiniciar completamente tu IDE para que surta efecto.

Uso

Desde tu IDE

Ejecuta `claude` desde la terminal integrada de tu IDE, y todas las características de integración estarán activas.

Desde Terminales Externos

Utiliza el comando `/ide` en cualquier terminal externo para conectar Claude Code a tu JetBrains IDE y activar todas las características:

```
claude
```

```
/ide
```

Si deseas que Claude tenga acceso a los mismos archivos que tu IDE, inicia Claude Code desde el mismo directorio que la raíz del proyecto de tu IDE.

Configuración

Configuración de Claude Code

Configura la integración del IDE a través de la configuración de Claude Code:

1. Ejecuta `claude`
2. Ingresa el comando `/config`

3. Establece la herramienta de diferencias en **auto** para la detección automática del IDE

Configuración del Plugin

Configura el plugin de Claude Code yendo a **Settings** → **Tools** → **Claude Code [Beta]**:

Configuración General

- **Comando Claude:** Especifica un comando personalizado para ejecutar Claude (por ejemplo, `claude`, `/usr/local/bin/claude`, o `npx @anthropic/claude`)
- **Suprimir notificación para comando Claude no encontrado:** Omite notificaciones sobre no encontrar el comando Claude
- **Habilitar usar Option+Enter para indicadores de varias líneas** (solo macOS): Cuando está habilitado, Option+Enter inserta nuevas líneas en los indicadores de Claude Code. Desactívalo si experimentas problemas con la tecla Option siendo capturada inesperadamente (requiere reinicio de terminal)
- **Habilitar actualizaciones automáticas:** Verifica automáticamente e instala actualizaciones del plugin (se aplica al reiniciar)

Tip:

Para usuarios de WSL: Establece `wsl -d Ubuntu -- bash -lic "claude"` como tu comando Claude (reemplaza `Ubuntu` con el nombre de tu distribución WSL)

Configuración de la Tecla ESC

Si la tecla ESC no interrumpe las operaciones de Claude Code en terminales de JetBrains:

1. Ve a **Settings** → **Tools** → **Terminal**
2. Cualquiera de:
 - Desactiva “Move focus to the editor with Escape”, o
 - Haz clic en “Configure terminal keybindings” y elimina el atajo “Switch focus to Editor”
3. Aplica los cambios

Esto permite que la tecla ESC interrumpa correctamente las operaciones de Claude Code.

Configuraciones Especiales

Desarrollo Remoto

Warning:

Cuando utilices JetBrains Remote Development, debes instalar el plugin en el host remoto a través de **Settings** → **Plugin (Host)**.

El plugin debe instalarse en el host remoto, no en tu máquina cliente local.

Configuración de WSL

Warning:

Los usuarios de WSL pueden necesitar configuración adicional para que la detección del IDE funcione correctamente. Consulta nuestra [guía de solución de problemas de WSL](#) para obtener instrucciones de configuración detalladas.

La configuración de WSL puede requerir:

- Configuración adecuada de la terminal
- Ajustes del modo de red
- Actualizaciones de configuración del firewall

Solución de Problemas

Plugin No Funciona

- Asegúrate de que estés ejecutando Claude Code desde el directorio raíz del proyecto
- Verifica que el plugin de JetBrains esté habilitado en la configuración del IDE
- Reinicia completamente el IDE (es posible que necesites hacerlo varias veces)
- Para Desarrollo Remoto, asegúrate de que el plugin esté instalado en el host remoto

IDE No Detectado

- Verifica que el plugin esté instalado y habilitado
- Reinicia completamente el IDE
- Comprueba que estés ejecutando Claude Code desde la terminal integrada
- Para usuarios de WSL, consulta la [guía de solución de problemas de WSL](#)

Comando No Encontrado

Si hacer clic en el icono de Claude muestra “command not found”:

1. Verifica que Claude Code esté instalado: `npm list -g @anthropic-ai/claude-code`
2. Configura la ruta del comando Claude en la configuración del plugin
3. Para usuarios de WSL, utiliza el formato de comando WSL mencionado en la sección de configuración

Consideraciones de Seguridad

Cuando Claude Code se ejecuta en un JetBrains IDE con permisos de edición automática habilitados, puede ser capaz de modificar archivos de configuración del IDE que pueden ser ejecutados automáticamente por tu IDE. Esto puede aumentar el riesgo de ejecutar Claude Code en modo de edición automática y permitir eludir los indicadores de permiso de Claude Code para la ejecución de bash.

Cuando se ejecuta en JetBrains IDEs, considera:

- Usar el modo de aprobación manual para ediciones
- Tener especial cuidado para asegurar que Claude solo se use con indicadores de confianza
- Ser consciente de qué archivos Claude Code tiene acceso para modificar

Para obtener ayuda adicional, consulta nuestra [guía de solución de problemas](#).

Usar Claude Code Desktop

Aproveche al máximo Claude Code Desktop: sesiones paralelas con aislamiento de Git, revisión visual de diferencias, vistas previas de aplicaciones, monitoreo de PR, modos de permisos, conectores y configuración empresarial.

La pestaña Code dentro de la aplicación Claude Desktop le permite usar Claude Code a través de una interfaz gráfica en lugar de la terminal.

Desktop añade estas capacidades además de la experiencia estándar de Claude Code:

- [Revisión visual de diferencias](#) con comentarios en línea
- [Vista previa de aplicación en vivo](#) con servidores de desarrollo
- [Monitoreo de PR de GitHub](#) con corrección automática y fusión automática
- [Sesiones paralelas](#) con aislamiento automático de Git worktree
- [Tareas programadas](#) que ejecutan Claude en un horario recurrente
- [Conectores](#) para GitHub, Slack, Linear y más
- Entornos locales, [SSH](#) y [en la nube](#)

Tip:

¿Nuevo en Desktop? Comience con [Introducción](#) para instalar la aplicación y realizar su primera edición.

Esta página cubre [trabajar con código](#), [gestionar sesiones](#), [extender Claude Code](#), [tareas programadas](#) y [configuración](#). También incluye una [comparación de CLI](#) y [solución de problemas](#).

Iniciar una sesión

Antes de enviar su primer mensaje, configure cuatro cosas en el área de indicaciones:

- **Entorno:** elija dónde se ejecuta Claude. Seleccione **Local** para su máquina, **Remote** para sesiones en la nube alojadas por Anthropic, o una [conexión SSH](#) para una máquina remota que usted administra. Consulte [configuración del entorno](#).
- **Carpeta del proyecto:** seleccione la carpeta o repositorio en el que Claude trabaja. Para sesiones remotas, puede agregar [múltiples repositorios](#).

- **Modelo:** elija un [modelo](#) del menú desplegable junto al botón de envío. El modelo se bloquea una vez que la sesión comienza.
- **Modo de permisos:** elija cuánta autonomía tiene Claude desde el [selector de modo](#). Puede cambiar esto durante la sesión.

Escriba su tarea y presione **Enter** para comenzar. Cada sesión rastrea su propio contexto y cambios de forma independiente.

Trabajar con código

Proporcione a Claude el contexto correcto, controle cuánto hace por su cuenta y revise lo que cambió.

Usar el cuadro de indicaciones

Escriba lo que desea que Claude haga y presione **Enter** para enviar. Claude lee los archivos de su proyecto, realiza cambios y ejecuta comandos según su [modo de permisos](#). Puede interrumpir a Claude en cualquier momento: haga clic en el botón de parada o escriba su corrección y presione **Enter**. Claude detiene lo que está haciendo y se ajusta según su entrada.

El botón + junto al cuadro de indicaciones le da acceso a adjuntos de archivos, [skills](#), [conectores](#) y [plugins](#).

Agregar archivos y contexto a las indicaciones

El cuadro de indicaciones admite dos formas de traer contexto externo:

- **Archivos @mention:** escriba  seguido de un nombre de archivo para agregar un archivo al contexto de la conversación. Claude puede entonces leer y hacer referencia a ese archivo.
- **Adjuntar archivos:** adjunte imágenes, PDF y otros archivos a su indicación usando el botón de adjuntos, o arrastre y suelte archivos directamente en la indicación. Esto es útil para compartir capturas de pantalla de errores, maquetas de diseño o documentos de referencia.

Elegir un modo de permisos

Los modos de permisos controlan cuánta autonomía tiene Claude durante una sesión: si pregunta antes de editar archivos, ejecutar comandos o ambos. Puede cambiar de modo en cualquier momento usando el selector de modo junto al botón de envío. Comience con Ask permissions para ver exactamente qué hace Claude, luego pase a Auto accept edits o Plan mode a medida que se sienta cómodo.

Modo	Clave de configuración	Comportamiento
Ask permissions	<code>default</code>	Claude pregunta antes de editar archivos o ejecutar comandos. Usted ve una diferencia y puede aceptar o rechazar cada cambio. Recomendado para nuevos usuarios.
Auto accept edits	<code>acceptEdits</code>	Claude acepta automáticamente ediciones de archivos pero aún pregunta antes de ejecutar comandos de terminal. Use esto cuando confíe en cambios de archivos y desee una iteración más rápida.
Plan mode	<code>plan</code>	Claude analiza su código y crea un plan sin modificar archivos ni ejecutar comandos. Bueno para tareas complejas donde desea revisar el enfoque primero.
Bypass permissions	<code>bypassPermissions</code>	Claude se ejecuta sin ningún aviso de permisos, equivalente a <code>--dangerously-skip-permissions</code> en la CLI. Habilite en su Configuración → Claude Code bajo “Allow bypass permissions mode”. Use solo en contenedores o máquinas virtuales sandboxed. Los administradores empresariales pueden deshabilitar esta opción.

El modo de permisos `dontAsk` está disponible solo en la [CLI](#).

Tip:

Comience tareas complejas en Plan mode para que Claude mapee un enfoque antes de realizar cambios. Una vez que apruebe el plan, cambie a Auto accept edits o Ask permissions para ejecutarlo. Consulte [explorar primero, luego planificar, luego codificar](#) para obtener más información sobre este flujo de trabajo.

Las sesiones remotas admiten Auto accept edits y Plan mode. Ask permissions no está disponible porque las sesiones remotas aceptan automáticamente ediciones de archivos de forma predeterminada, y Bypass permissions no está disponible porque el entorno remoto ya está sandboxed.

Los administradores empresariales pueden restringir qué modos de permisos están disponibles. Consulte [configuración empresarial](#) para obtener detalles.

Vista previa de su aplicación

Claude puede iniciar un servidor de desarrollo y abrir un navegador integrado para verificar sus cambios. Esto funciona tanto para aplicaciones web frontend como para servidores backend: Claude puede probar puntos finales de API, ver registros del servidor e iterar sobre problemas que encuentra. En la mayoría de los casos, Claude inicia el servidor automáticamente después de editar archivos del proyecto. También puede pedirle a Claude que haga una vista previa en cualquier momento. De forma predeterminada, Claude [verifica automáticamente](#) cambios después de cada edición.

Desde el panel de vista previa, puede:

- Interactuar con su aplicación en ejecución directamente en el navegador integrado
- Ver a Claude verificar sus propios cambios automáticamente: toma capturas de pantalla, inspecciona el DOM, hace clic en elementos, completa formularios y corrige problemas que encuentra
- Iniciar o detener servidores desde el menú desplegable **Preview** en la barra de herramientas de la sesión
- Persistir cookies y almacenamiento local en reinicios del servidor seleccionando **Persist sessions** en el menú desplegable, para que no tenga que volver a iniciar sesión durante el desarrollo
- Editar la configuración del servidor o detener todos los servidores a la vez

Claude crea la configuración inicial del servidor basada en su proyecto. Si su aplicación usa un comando de desarrollo personalizado, edite `.claude/launch.json` para que coincida con su configuración. Consulte [Configurar servidores de vista previa](#) para la referencia completa.

Para borrar datos de sesión guardados, alterne **Persist preview sessions** en Configuración → Claude Code. Para deshabilitar la vista previa por completo, alterne **Preview** en Configuración → Claude Code.

Revisar cambios con vista de diferencias

Después de que Claude realiza cambios en su código, la vista de diferencias le permite revisar modificaciones archivo por archivo antes de crear una solicitud de extracción.

Cuando Claude cambia archivos, aparece un indicador de estadísticas de diferencias que muestra el número de líneas agregadas y eliminadas, como **+12 -1**. Haga clic en este indicador para abrir el visor de diferencias, que muestra una lista de archivos a la izquierda y los cambios para cada archivo a la derecha.

Para comentar en líneas específicas, haga clic en cualquier línea en la diferencia para abrir un cuadro de comentarios. Escriba su comentario y presione **Enter** para agregar el comentario. Después de agregar comentarios a varias líneas, envíe todos los comentarios a la vez:

- **macOS:** presione **Cmd+Enter**
- **Windows:** presione **Ctrl+Enter**

Claude lee sus comentarios y realiza los cambios solicitados, que aparecen como una nueva diferencia que puede revisar.

Revisar su código

En la vista de diferencias, haga clic en **Review code** en la barra de herramientas superior derecha para pedirle a Claude que evalúe los cambios antes de confirmar. Claude examina las diferencias actuales y deja comentarios directamente en la vista de diferencias. Puede responder a cualquier comentario o pedirle a Claude que revise.

La revisión se enfoca en problemas de alta señal: errores de compilación, errores de lógica definitivos, vulnerabilidades de seguridad y errores obvios. No marca estilo, formato, problemas preexistentes o nada que un linter detectaría.

Monitorear el estado de la solicitud de extracción

Después de abrir una solicitud de extracción, aparece una barra de estado de CI en la sesión. Claude Code usa la CLI de GitHub para sondear resultados de verificación y mostrar fallas.

- **Auto-fix:** cuando está habilitado, Claude intenta automáticamente corregir verificaciones de CI fallidas leyendo la salida de falla e iterando.

- **Auto-merge:** cuando está habilitado, Claude fusiona el PR una vez que todas las verificaciones pasan. El método de fusión es squash. Auto-merge debe estar [habilitado en la configuración de su repositorio de GitHub](#) para que esto funcione.

Use los controles **Auto-fix** y **Auto-merge** en la barra de estado de CI para habilitar cualquiera de las opciones. Claude Code también envía una notificación de escritorio cuando CI finaliza.

Note:

El monitoreo de PR requiere que la [CLI de GitHub \(gh \)](#) esté instalada y autenticada en su máquina. Si **gh** no está instalado, Desktop le solicita que lo instale la primera vez que intente crear un PR.

Gestionar sesiones

Cada sesión es una conversación independiente con su propio contexto y cambios. Puede ejecutar múltiples sesiones en paralelo o enviar trabajo a la nube.

Trabajar en paralelo con sesiones

Haga clic en + **New session** en la barra lateral para trabajar en múltiples tareas en paralelo. Para repositorios de Git, cada sesión obtiene su propia copia aislada de su proyecto usando [Git worktrees](#), por lo que los cambios en una sesión no afectan otras sesiones hasta que los confirme.

Los worktrees se almacenan en `<project-root>/ .claude/worktrees/` de forma predeterminada. Puede cambiar esto a un directorio personalizado en Configuración → Claude Code bajo “Worktree location”. También puede establecer un prefijo de rama que se anteponga a cada nombre de rama de worktree, lo que es útil para mantener las ramas creadas por Claude organizadas. Para eliminar un worktree cuando haya terminado, pase el cursor sobre la sesión en la barra lateral y haga clic en el icono de archivo.

Note:

El aislamiento de sesión requiere [Git](#). La mayoría de las Macs incluyen Git de forma predeterminada. Ejecute `git --version` en Terminal para verificar. En Windows, Git es necesario para que la pestaña Code funcione: [descargue Git para Windows](#), instálelo y reinicie la aplicación. Si encuentra errores de Git, intente una sesión de Cowork para ayudar a solucionar problemas de su configuración.

Use el icono de filtro en la parte superior de la barra lateral para filtrar sesiones por estado (Active, Archived) y entorno (Local, Cloud). Para renombrar una sesión o verificar el uso del contexto, haga clic en el título de la sesión en la barra de herramientas en la parte superior de la sesión activa. Cuando el contexto se llena, Claude resume automáticamente la conversación y continúa trabajando. También puede escribir `/compact` para activar la compresión antes y liberar espacio de contexto. Consulte [la ventana de contexto](#) para obtener detalles sobre cómo funciona la compresión.

Ejecutar tareas de larga duración de forma remota

Para refactorizaciones grandes, suites de pruebas, migraciones u otras tareas de larga duración, seleccione **Remote** en lugar de **Local** al iniciar una sesión. Las sesiones remotas se ejecutan en la infraestructura en la nube de Anthropic y continúan incluso si cierra la aplicación o apaga su computadora. Regrese en cualquier momento para ver el progreso o dirigir a Claude en una dirección diferente. También puede monitorear sesiones remotas desde claude.ai/code o la aplicación Claude iOS.

Las sesiones remotas también admiten múltiples repositorios. Después de seleccionar un entorno en la nube, haga clic en el botón + junto a la píldora de repositorio para agregar repositorios adicionales a la sesión. Cada repositorio obtiene su propio selector de rama. Esto es útil para tareas que abarcan múltiples bases de código, como actualizar una biblioteca compartida y sus consumidores.

Consulte [Claude Code en la web](#) para obtener más información sobre cómo funcionan las sesiones remotas.

Continuar en otra superficie

El menú **Continue in**, accesible desde el icono de VS Code en la esquina inferior derecha de la barra de herramientas de la sesión, le permite mover su sesión a otra superficie:

- **Claude Code on the Web:** envía su sesión local para continuar ejecutándose de forma remota. Desktop empuja su rama, genera un resumen de la conversación y crea una nueva sesión remota con el contexto completo. Luego puede elegir archivar la sesión local o mantenerla. Esto requiere un árbol de trabajo limpio y no está disponible para sesiones SSH.
- **Your IDE:** abre su proyecto en un IDE compatible en el directorio de trabajo actual.

Extender Claude Code

Conecte servicios externos, agregue flujos de trabajo reutilizables, personalice el comportamiento de Claude y configure servidores de vista previa.

Conectar herramientas externas

Para sesiones locales y [SSH](#), haga clic en el botón + junto al cuadro de indicaciones y seleccione **Connectors** para agregar integraciones como Google Calendar, Slack, GitHub, Linear, Notion y más. Puede agregar conectores antes o durante una sesión. Los conectores no están disponibles para sesiones remotas.

Para administrar o desconectar conectores, vaya a Configuración → Connectors en la aplicación de escritorio, o seleccione **Manage connectors** desde el menú Connectors en el cuadro de indicaciones.

Una vez conectado, Claude puede leer su calendario, enviar mensajes, crear problemas e interactuar con sus herramientas directamente. Puede preguntarle a Claude qué conectores están configurados en su sesión.

Los conectores son [MCP servers](#) con un flujo de configuración gráfico. Úselos para integración rápida con servicios compatibles. Para integraciones no listadas en Connectors, agregue MCP servers manualmente a través de [archivos de configuración](#). También puede [crear conectores personalizados](#).

Usar skills

[Skills](#) extienden lo que Claude puede hacer. Claude los carga automáticamente cuando son relevantes, o puede invocar uno directamente: escriba `/` en el cuadro de indicaciones o haga clic en el botón + y seleccione **Slash commands** para ver lo que está disponible. Esto incluye [comandos integrados](#), sus [skills personalizados](#), skills del proyecto desde su base de código y skills de cualquier [plugin instalado](#). Seleccione uno y aparecerá resaltado en el campo de entrada. Escriba su tarea después de él y envíe como de costumbre.

Instalar plugins

[Plugins](#) son paquetes reutilizables que agregan skills, agents, hooks, MCP servers y configuraciones LSP a Claude Code. Puede instalar plugins desde la aplicación de escritorio sin usar la terminal.

Para sesiones locales y [SSH](#), haga clic en el botón + junto al cuadro de indicaciones y seleccione **Plugins** para ver sus plugins instalados y sus comandos. Para agregar un plugin, seleccione **Add plugin** del submenú para abrir el navegador de plugins, que muestra plugins disponibles desde sus [marketplaces](#) configurados incluyendo el marketplace oficial de Anthropic. Seleccione **Manage plugins** para habilitar, deshabilitar o desinstalar plugins.

Los plugins pueden estar limitados a su cuenta de usuario, un proyecto específico o solo locales. Los plugins no están disponibles para sesiones remotas. Para la referencia completa de plugins incluyendo crear sus propios plugins, consulte [plugins](#).

Configurar servidores de vista previa

Claude detecta automáticamente su configuración de servidor de desarrollo y almacena la configuración en `.claude/launch.json` en la raíz de la carpeta que seleccionó al iniciar la sesión. Preview usa esta carpeta como su directorio de trabajo, por lo que si seleccionó una carpeta principal, las subcarpetas con sus propios servidores de desarrollo no se detectarán automáticamente. Para trabajar con el servidor de una subcarpeta, inicie una sesión en esa carpeta directamente o agregue una configuración manualmente.

Para personalizar cómo se inicia su servidor, por ejemplo para usar `yarn dev` en lugar de `npm run dev` o para cambiar el puerto, edite el archivo manualmente o haga clic en **Edit configuration** en el menú desplegable Preview para abrirlo en su editor de código. El archivo admite JSON con comentarios.

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

Puede definir múltiples configuraciones para ejecutar diferentes servidores desde el mismo proyecto, como un frontend y una API. Consulte los [ejemplos](#) a continuación.

Verificación automática de cambios

Cuando `autoVerify` está habilitado, Claude verifica automáticamente cambios de código después de editar archivos. Toma capturas de pantalla, verifica errores y confirma que los cambios funcionan antes de completar su respuesta.

Auto-verify está habilitado de forma predeterminada. Desactívelo por proyecto agregando `"autoVerify": false` a `.claude/launch.json`, o alterne desde el menú desplegable **Preview**.

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

Cuando está deshabilitado, las herramientas de vista previa aún están disponibles y puede pedirle a Claude que verifique en cualquier momento. Auto-verify lo hace automático después de cada edición.

Campos de configuración

Cada entrada en el array `configurations` acepta los siguientes campos:

Campo	Tipo	Descripción
<code>name</code>	string	Un identificador único para este servidor
<code>runtimeExecutable</code>	string	El comando a ejecutar, como <code>npm</code> , <code>yarn</code> o <code>node</code>
<code>runtimeArgs</code>	string[]	Argumentos pasados a <code>runtimeExecutable</code> , como <code>["run", "dev"]</code>
<code>port</code>	number	El puerto en el que escucha su servidor. Por defecto es 3000
<code>cwd</code>	string	Directorio de trabajo relativo a la raíz de su proyecto. Por defecto es la raíz del proyecto. Use <code>\$ {workspaceFolder}</code> para hacer referencia a la raíz del proyecto explícitamente

Campo	Tipo	Descripción
<code>env</code>	object	Variables de entorno adicionales como pares clave-valor, como <code>{ "NODE_ENV": "development" }</code> . No ponga secretos aquí ya que este archivo se confirma en su repositorio. Los secretos establecidos en su perfil de shell se heredan automáticamente.
<code>autoPort</code>	boolean	Cómo manejar conflictos de puerto. Consulte a continuación
<code>program</code>	string	Un script a ejecutar con <code>node</code> . Consulte cuándo usar program vs runtimeExecutable
<code>args</code>	string[]	Argumentos pasados a <code>program</code> . Solo se usa cuando <code>program</code> está establecido

Cuándo usar `program` vs `runtimeExecutable`

Use `runtimeExecutable` con `runtimeArgs` para iniciar un servidor de desarrollo a través de un administrador de paquetes. Por ejemplo, `"runtimeExecutable": "npm"` con `"runtimeArgs": ["run", "dev"]` ejecuta `npm run dev`.

Use `program` cuando tenga un script independiente que desee ejecutar con `node` directamente. Por ejemplo, `"program": "server.js"` ejecuta `node server.js`. Pase banderas adicionales con `args`.

Conflictos de puerto

El campo `autoPort` controla qué sucede cuando su puerto preferido ya está en uso:

- `true` : Claude encuentra y usa un puerto libre automáticamente. Adecuado para la mayoría de servidores de desarrollo.

- **false** : Claude falla con un error. Use esto cuando su servidor debe usar un puerto específico, como para devoluciones de llamada de OAuth o listas de permitidos de CORS.
- **No establecido (predeterminado)**: Claude pregunta si el servidor necesita ese puerto exacto, luego guarda su respuesta.

Cuando Claude elige un puerto diferente, pasa el puerto asignado a su servidor a través de la variable de entorno **PORT**.

Ejemplos

Estas configuraciones muestran configuraciones comunes para diferentes tipos de proyectos:

Next.js

Esta configuración ejecuta una aplicación Next.js usando Yarn en el puerto 3000:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Multiple servers

Para un monorepo con un servidor frontend y API, defina múltiples configuraciones. El frontend usa **autoPort: true** para que elija un puerto libre si 3000 está ocupado, mientras que el servidor API requiere el puerto 8080 exactamente:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "frontend",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "cwd": "apps/web",
      "port": 3000,
      "autoPort": true
    },
    {
      "name": "api",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "start"],
      "cwd": "server",
      "port": 8080,
      "env": { "NODE_ENV": "development" },
      "autoPort": false
    }
  ]
}
```

Node.js script

Para ejecutar un script Node.js directamente en lugar de usar un comando del administrador de paquetes, use el campo `program` :

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

Programar tareas recurrentes

Las tareas programadas inician una nueva sesión local automáticamente en una hora y frecuencia que usted elige. Úselas para trabajo recurrente como revisiones de código diarias, verificaciones de actualización de dependencias o resúmenes matutinos que extraigan de su calendario e bandeja de entrada.

Las tareas se ejecutan en su máquina, por lo que la aplicación de escritorio debe estar abierta y su computadora despierta para que se ejecuten. Consulte [Cómo se ejecutan las tareas programadas](#) para obtener detalles sobre ejecuciones perdidas y comportamiento de recuperación.

Note:

De forma predeterminada, las tareas programadas se ejecutan contra cualquier estado en el que se encuentre su directorio de trabajo, incluyendo cambios no confirmados. Habilite el control de worktree en la entrada de indicaciones para dar a cada ejecución su propio Git worktree aislado, de la misma manera que [sesiones paralelas](#) funcionan.

Para crear una tarea programada, haga clic en **Schedule** en la barra lateral, luego **+ New task**. Configure estos campos:

Cam-po	Descripción
Name	Identificador para la tarea. Se convierte a kebab-case en minúsculas y se usa como nombre de carpeta en disco. Debe ser único en todas sus tareas.

Cam-po	Descripción
Des-crip-tion	Resumen corto mostrado en la lista de tareas.
Prompt	Las instrucciones enviadas a Claude cuando se ejecuta la tarea. Escriba esto de la misma manera que escribiría cualquier mensaje en el cuadro de indicaciones. La entrada de indicaciones también incluye controles para modelo, modo de permisos, carpeta de trabajo y worktree.
Fre-quency	Con qué frecuencia se ejecuta la tarea. Consulte opciones de frecuencia a continuación.

También puede crear una tarea describiendo lo que desea en cualquier sesión. Por ejemplo, “configurar una revisión de código diaria que se ejecute cada mañana a las 9am.”

Opciones de frecuencia

- **Manual:** sin horario, solo se ejecuta cuando hace clic en **Run now**. Útil para guardar una indicación que activa bajo demanda
- **Hourly:** se ejecuta cada hora. Cada tarea obtiene un desplazamiento fijo de hasta 10 minutos desde la parte superior de la hora para escalonar el tráfico de API
- **Daily:** muestra un selector de hora, por defecto a las 9:00 AM hora local
- **Weekdays:** igual que Daily pero omite sábado y domingo
- **Weekly:** muestra un selector de hora y un selector de día

Para intervalos que el selector no ofrece (cada 15 minutos, primer día de cada mes, etc.), pídale a Claude en cualquier sesión de Desktop que establezca el horario. Use lenguaje natural; por ejemplo, “programar una tarea para ejecutar todas las pruebas cada 6 horas.”

Cómo se ejecutan las tareas programadas

Las tareas programadas se ejecutan localmente en su máquina. Desktop verifica el horario cada minuto mientras la aplicación está abierta e inicia una sesión nueva cuando una tarea vence, independientemente de cualquier sesión manual que tenga abierta. Cada tarea obtiene un retraso fijo de hasta 10 minutos después de la hora programada para escalonar el tráfico de API. El retraso es determinista: la misma tarea siempre comienza en el mismo desplazamiento.

Cuando se ejecuta una tarea, obtiene una notificación de escritorio y aparece una nueva sesión bajo una sección **Scheduled** en la barra lateral. Ábrala para ver qué hizo Claude, revisar cambios o responder a avisos de permisos. La sesión funciona como cualquier otra: Claude puede editar archivos, ejecutar comandos, crear confirmaciones y abrir solicitudes de extracción.

Las tareas solo se ejecutan mientras la aplicación de escritorio está en ejecución y su computadora está despierta. Si su computadora se duerme durante una hora programada, la ejecución se omite. Para evitar el sueño inactivo, habilite **Keep computer awake** en Configuración bajo **Desktop app** → **General**. Cerrar la tapa del portátil aún lo pone a dormir.

Ejecuciones perdidas

Cuando la aplicación se inicia o su computadora se despierta, Desktop verifica si cada tarea perdió alguna ejecución en los últimos siete días. Si lo hizo, Desktop inicia exactamente una ejecución de recuperación para la hora más recientemente perdida y descarta cualquier cosa más antigua. Una tarea diaria que perdió seis días se ejecuta una vez al despertar. Desktop muestra una notificación cuando comienza una ejecución de recuperación.

Tenga esto en cuenta al escribir indicaciones. Una tarea programada para las 9am podría ejecutarse a las 11pm si su computadora estuvo dormida todo el día. Si el tiempo es importante, agregue protecciones a la indicación misma, por ejemplo: “Solo revise las confirmaciones de hoy. Si es después de las 5pm, omita la revisión y solo publique un resumen de lo que se perdió.”

Permisos para tareas programadas

Cada tarea tiene su propio modo de permisos, que establece al crear o editar la tarea. Las reglas de permitir de `~/.claude/settings.json` también se aplican a sesiones de tareas programadas. Si una tarea se ejecuta en modo Ask y necesita ejecutar una herramienta para la que no tiene permiso, la ejecución se detiene hasta que la apruebe. La sesión permanece abierta en la barra lateral para que pueda responder más tarde.

Para evitar detenciones, haga clic en **Run now** después de crear una tarea, observe los avisos de permisos y seleccione “always allow” para cada uno. Las ejecuciones futuras de esa tarea aprueban automáticamente las mismas herramientas sin solicitar. Puede revisar y revocar estas aprobaciones desde la página de detalles de la tarea.

Gestionar tareas programadas

Haga clic en una tarea en la lista **Schedule** para abrir su página de detalles. Desde aquí puede:

- **Run now:** inicie la tarea inmediatamente sin esperar la próxima hora programada
- **Toggle repeats:** pause o reanude ejecuciones programadas sin eliminar la tarea
- **Edit:** cambie la indicación, frecuencia, carpeta u otras configuraciones
- **Review history:** vea cada ejecución anterior, incluyendo las que se omitieron porque su computadora estaba dormida
- **Review allowed permissions:** vea y revoque aprobaciones de herramientas guardadas para esta tarea desde el panel **Always allowed**
- **Delete:** elimine la tarea y archive todas las sesiones que creó

También puede gestionar tareas pidiendo a Claude en cualquier sesión de Desktop. Por ejemplo, “pause my dependency-audit task”, “delete the standup-prep task” o “show me my scheduled tasks.”

Para editar la indicación de una tarea en disco, abra `~/.claude/scheduled-tasks/<task-name>/SKILL.md` (o bajo `CLAUDE_CONFIG_DIR` si está establecido). El archivo usa frontmatter YAML para `name` y `description`, con la indicación como cuerpo. Los cambios surten efecto en la próxima ejecución. El horario, carpeta, modelo y estado habilitado no están en este archivo: cámbielos a través del formulario Edit o pídale a Claude.

Configuración del entorno

El entorno que elige al [iniciar una sesión](#) determina dónde Claude se ejecuta y cómo se conecta:

- **Local:** se ejecuta en su máquina con acceso directo a sus archivos
- **Remote:** se ejecuta en la infraestructura en la nube de Anthropic. Las sesiones continúan incluso si cierra la aplicación.
- **SSH:** se ejecuta en una máquina remota a la que se conecta a través de SSH, como sus propios servidores, máquinas virtuales en la nube o contenedores de desarrollo

Sesiones locales

Las sesiones locales heredan variables de entorno de su shell. Si necesita variables adicionales, establézcalas en su perfil de shell, como `~/.zshrc` o `~/.bashrc`, y reinicie la aplicación de escritorio. Consulte [variables de entorno](#) para la lista completa de variables admitidas.

[Extended thinking](#) está habilitado de forma predeterminada, lo que mejora el rendimiento en tareas de razonamiento complejo pero usa tokens adicionales. Para deshabilitar el pensamiento por completo, establezca `MAX_THINKING_TOKENS=0` en su perfil de shell. En Opus, `MAX_THINKING_TOKENS` se ignora excepto para `0` porque el razonamiento adaptativo controla la profundidad del pensamiento en su lugar.

Sesiones remotas

Las sesiones remotas continúan en segundo plano incluso si cierra la aplicación. El uso cuenta hacia los [límites de su plan de suscripción](#) sin cargos de computación separados.

Puede crear entornos en la nube personalizados con diferentes niveles de acceso a la red y variables de entorno. Seleccione el menú desplegable de entorno al iniciar una sesión remota y elija **Add environment**. Consulte [entornos en la nube](#) para obtener detalles sobre la configuración del acceso a la red y variables de entorno.

Sesiones SSH

Las sesiones SSH le permiten ejecutar Claude Code en una máquina remota mientras usa la aplicación de escritorio como su interfaz. Esto es útil para trabajar con bases de código que viven en máquinas virtuales en la nube, contenedores de desarrollo o servidores con hardware o dependencias específicas.

Para agregar una conexión SSH, haga clic en el menú desplegable de entorno antes de iniciar una sesión y seleccione + **Add SSH connection**. El diálogo solicita:

- **Name:** una etiqueta amigable para esta conexión
- **SSH Host:** `user@hostname` o un host definido en `~/.ssh/config`
- **SSH Port:** por defecto es 22 si se deja vacío, o usa el puerto de su configuración SSH
- **Identity File:** ruta a su clave privada, como `~/.ssh/id_rsa`. Déjelo vacío para usar la clave predeterminada o su configuración SSH.

Una vez agregada, la conexión aparece en el menú desplegable de entorno. Selecciónela para iniciar una sesión en esa máquina. Claude se ejecuta en la máquina remota con acceso a sus archivos y herramientas.

Claude Code debe estar instalado en la máquina remota. Una vez conectado, las sesiones SSH admiten modos de permisos, conectores, plugins y MCP servers.

Configuración empresarial

Las organizaciones en planes Teams o Enterprise pueden gestionar el comportamiento de la aplicación de escritorio a través de controles de consola de administración, archivos de configuración administrados y políticas de gestión de dispositivos.

Controles de consola de administración

Estas configuraciones se configuran a través de la [consola de configuración de administración](#):

- **Habilitar o deshabilitar la pestaña Code:** controle si los usuarios de su organización pueden acceder a Claude Code en la aplicación de escritorio
- **Deshabilitar modo Bypass permissions:** evite que los usuarios de su organización habiliten el modo bypass permissions
- **Deshabilitar Claude Code en la web:** habilite o deshabilite sesiones remotas para su organización

Configuración administrada

La configuración administrada anula la configuración del proyecto y usuario y se aplica cuando Desktop genera sesiones de CLI. Puede establecer estas claves en el archivo de [configuración administrada](#) de su organización o enviarlas de forma remota a través de la consola de administración.

Clave	Descripción
<code>disableBypassPermissionsMode</code>	establezca en "disable" para evitar que los usuarios habiliten el modo bypass permissions. Consulte configuración administrada .

Para la lista completa de configuraciones solo administradas incluyendo `allowManagedPermissionRulesOnly` y `allowManagedHooksOnly` , consulte [configuraciones solo administradas](#).

La configuración administrada remota cargada a través de la consola de administración actualmente se aplica solo a sesiones de CLI e IDE. Para restricciones específicas de Desktop, use los controles de consola de administración anteriores.

Políticas de gestión de dispositivos

Los equipos de TI pueden gestionar la aplicación de escritorio a través de MDM en macOS o política de grupo en Windows. Las políticas disponibles incluyen habilitar o deshabilitar la función Claude Code, controlar actualizaciones automáticas y establecer una URL de implementación personalizada.

- **macOS:** configure a través del dominio de preferencia `com.anthropic.Claude` usando herramientas como Jamf o Kandji
- **Windows:** configure a través del registro en `SOFTWARE\Policies\Claude`

Autenticación y SSO

Las organizaciones empresariales pueden requerir SSO para todos los usuarios. Consulte [autenticación](#) para obtener detalles a nivel de plan y [Configurar SSO](#) para la configuración de SAML y OIDC.

Manejo de datos

Claude Code procesa su código localmente en sesiones locales o en la infraestructura en la nube de Anthropic en sesiones remotas. Las conversaciones y el contexto del código se envían a la API de Anthropic para procesamiento. Consulte [manejo de datos](#) para obtener detalles sobre retención de datos, privacidad y cumplimiento.

Implementación

Desktop se puede distribuir a través de herramientas de implementación empresarial:

- **macOS:** distribuya a través de MDM como Jamf o Kandji usando el instalador `.dmg`
- **Windows:** implemente a través del paquete MSIX o instalador `.exe`. Consulte [Deploy Claude Desktop for Windows](#) para opciones de implementación empresarial incluyendo instalación silenciosa

Para configuración de red como configuración de proxy, lista de permitidos de firewall y puertas de enlace LLM, consulte [configuración de red](#).

Para la referencia completa de configuración empresarial, consulte la [guía de configuración empresarial](#).

¿Viene de la CLI?

Si ya usa la CLI de Claude Code, Desktop ejecuta el mismo motor subyacente con una interfaz gráfica. Puede ejecutar ambos simultáneamente en la misma máquina, incluso en el mismo proyecto. Cada uno mantiene un historial de sesión separado, pero comparten configuración y memoria del proyecto a través de archivos CLAUDE.md.

Para mover una sesión de CLI a Desktop, ejecute `/desktop` en la terminal. Claude guarda su sesión y la abre en la aplicación de escritorio, luego sale de la CLI. Este comando está disponible solo en macOS y Windows.

Tip:

Cuándo usar Desktop vs CLI: use Desktop cuando desee revisión visual de diferencias, adjuntos de archivos o gestión de sesiones en una barra lateral. Use la CLI cuando necesite scripting, automatización, proveedores de terceros o prefiera un flujo de trabajo de terminal.

Equivalentes de banderas de CLI

Esta tabla muestra el equivalente de la aplicación de escritorio para banderas de CLI comunes. Las banderas no listadas no tienen equivalente de escritorio porque están diseñadas para scripting o automatización.

CLI	Equivalente de Desktop
<code>--model sonnet</code>	menú desplegable de modelo junto al botón de envío, antes de iniciar una sesión
<code>--resume</code> , <code>--continue</code>	haga clic en una sesión en la barra lateral
<code>--permission-mode</code>	selector de modo junto al botón de envío
<code>--dangerously-skip-permissions</code>	Modo Bypass permissions. Habilite en Configuración → Claude Code → “Allow bypass permissions mode”. Los administradores empresariales pueden deshabilitar esta configuración.
<code>--add-dir</code>	agregue múltiples repositorios con el botón + en sesiones remotas
<code>--allowedTools</code> , <code>--disallowedTools</code>	no disponible en Desktop
<code>--verbose</code>	no disponible. Verifique los registros del sistema: Console.app en macOS, Event Viewer → Windows Logs → Application en Windows
<code>--print</code> , <code>--output-format</code>	no disponible. Desktop es solo interactivo.
Variable de entorno <code>ANTHROPIC_MODEL</code>	menú desplegable de modelo junto al botón de envío
Variable de entorno <code>MAX_THINKING_TOKENS</code>	establezca en el perfil de shell; se aplica a sesiones locales. Consulte configuración del entorno .

Configuración compartida

Desktop y CLI leen los mismos archivos de configuración, por lo que su configuración se transfiere:

- Los archivos [CLAUDE.md](#) en su proyecto son utilizados por ambos
- Los [MCP servers](#) configurados en `~/.claude.json` o `.mcp.json` funcionan en ambos
- Los [Hooks](#) y [skills](#) definidos en configuración se aplican a ambos

- La **Configuración** en `~/.claude.json` y `~/.claude/settings.json` se comparte. Las reglas de permisos, herramientas permitidas y otras configuraciones en `settings.json` se aplican a sesiones de Desktop.
- **Modelos:** Sonnet, Opus y Haiku están disponibles en ambos. En Desktop, seleccione el modelo del menú desplegable junto al botón de envío antes de iniciar una sesión. No puede cambiar el modelo durante una sesión activa.

Note:

MCP servers: aplicación de chat de Desktop vs Claude Code: Los MCP servers configurados para la aplicación de chat de Claude Desktop en `claude_desktop_config.json` son separados de Claude Code y no aparecerán en la pestaña Code. Para usar MCP servers en Claude Code, configúrelos en `~/.claude.json` o en el archivo `.mcp.json` de su proyecto. Consulte [configuración de MCP](#) para obtener detalles.

Comparación de características

Esta tabla compara capacidades principales entre la CLI y Desktop. Para una lista completa de banderas de CLI, consulte la [referencia de CLI](#).

Característica	CLI	Desktop
Modos de permisos	todos los modos incluyendo <code>dontAsk</code>	Ask permissions, Auto accept edits, Plan mode y Bypass permissions a través de Configuración
<code>--dangerously-skip-permissions</code>	Bandera de CLI	Modo Bypass permissions. Habilite en Configuración → Claude Code → “Allow bypass permissions mode”
Proveedores de terceros	Bedrock, Vertex, Foundry	no disponible. Desktop se conecta directamente a la API de Anthropic.
MCP servers	configurar en archivos de configuración	UI de Connectors para sesiones locales y SSH, o archivos de configuración
Plugins	comando <code>/plugin</code>	UI del administrador de plugins
Archivos @mention	basado en texto	con autocompletado

Característica	CLI	Desktop
Adjuntos de archivos	no disponible	imágenes, PDF
Aislamiento de sesión	bandera <code>--worktree</code>	worktrees automáticos
Múltiples sesiones	terminales separadas	pestañas de barra lateral
Tareas recurrentes	trabajos cron, tuberías de CI	tareas programadas
Scripting y automatización	<code>--print</code> , Agent SDK	no disponible

Lo que no está disponible en Desktop

Las siguientes características están disponibles solo en la CLI o extensión de VS Code:

- **Proveedores de terceros:** Desktop se conecta directamente a la API de Anthropic. Use la [CLI](#) con Bedrock, Vertex o Foundry en su lugar.
- **Linux:** la aplicación de escritorio está disponible solo en macOS y Windows.
- **Sugerencias de código en línea:** Desktop no proporciona sugerencias de estilo autocompletado. Funciona a través de indicaciones conversacionales y cambios de código explícitos.
- **Equipos de agentes:** la orquestación de múltiples agentes está disponible a través de la [CLI](#) y [Agent SDK](#), no en Desktop.

Solución de problemas

Verificar su versión

Para ver qué versión de la aplicación de escritorio está ejecutando:

- **macOS:** haga clic en **Claude** en la barra de menú, luego **About Claude**
- **Windows:** haga clic en **Help**, luego **About**

Haga clic en el número de versión para copiarlo a su portapapeles.

Errores 403 o de autenticación en la pestaña Code

Si ve `Error 403: Forbidden` u otros fallos de autenticación al usar la pestaña Code:

1. Cierre sesión e inicie sesión nuevamente desde el menú de la aplicación. Esta es la solución más común.
2. Verifique que tenga una suscripción de pago activa: Pro, Max, Teams o Enterprise.

3. Si la CLI funciona pero Desktop no, cierre completamente la aplicación de escritorio, no solo cierre la ventana, luego reabrirla e inicie sesión nuevamente.
4. Verifique su conexión a Internet y configuración de proxy.

Pantalla en blanco o atascada al iniciar

Si la aplicación se abre pero muestra una pantalla en blanco o sin respuesta:

1. Reinicie la aplicación.
2. Verifique si hay actualizaciones pendientes. La aplicación se actualiza automáticamente al iniciar.
3. En Windows, verifique el Visor de eventos para registros de bloqueo bajo **Windows Logs → Application**.

“Failed to load session”

Si ve `Failed to load session`, la carpeta seleccionada puede que ya no exista, un repositorio de Git puede requerir Git LFS que no está instalado, o los permisos de archivo pueden impedir el acceso. Intente seleccionar una carpeta diferente o reinicie la aplicación.

La sesión no encuentra herramientas instaladas

Si Claude no puede encontrar herramientas como `npm`, `node` u otros comandos de CLI, verifique que las herramientas funcionen en su terminal regular, verifique que su perfil de shell configure correctamente PATH y reinicie la aplicación de escritorio para recargar variables de entorno.

Errores de Git y Git LFS

En Windows, Git es necesario para que la pestaña Code inicie sesiones locales. Si ve “Git is required,” instale [Git para Windows](#) y reinicie la aplicación.

Si ve “Git LFS is required by this repository but is not installed,” instale Git LFS desde [git-lfs.com](#), ejecute `git lfs install` y reinicie la aplicación.

Los MCP servers no funcionan en Windows

Si los controles de MCP server no responden o los servidores no se conectan en Windows, verifique que el servidor esté configurado correctamente en su configuración, reinicie la aplicación, verifique que el proceso del servidor se esté ejecutando en el Administrador de tareas y revise los registros del servidor para errores de conexión.

La aplicación no se cierra

- **macOS:** presione Cmd+Q. Si la aplicación no responde, use Force Quit con Cmd+Option+Esc, seleccione Claude y haga clic en Force Quit.
- **Windows:** use el Administrador de tareas con Ctrl+Shift+Esc para finalizar el proceso de Claude.

Problemas específicos de Windows

- **PATH no actualizado después de instalar:** abra una nueva ventana de terminal. Las actualizaciones de PATH solo se aplican a nuevas sesiones de terminal.
- **Error de instalación concurrente:** si ve un error sobre otra instalación en progreso pero no la hay, intente ejecutar el instalador como Administrador.
- **ARM64:** los dispositivos Windows ARM64 son totalmente compatibles.

Pestaña Cowork no disponible en Macs Intel

La pestaña Cowork requiere Apple Silicon (M1 o posterior) en macOS. En Windows, Cowork está disponible en todo el hardware compatible. Las pestañas Chat y Code funcionan normalmente en Macs Intel.

“Branch doesn’t exist yet” al abrir en CLI

Las sesiones remotas pueden crear ramas que no existen en su máquina local. Haga clic en el nombre de la rama en la barra de herramientas de la sesión para copiarlo, luego obténgalo localmente:

```
git fetch origin <branch-name>
git checkout <branch-name>
```

¿Aún atascado?

- Busque o presente un error en [GitHub Issues](#)
- Visite el [centro de soporte de Claude](#)

Al presentar un error, incluya la versión de su aplicación de escritorio, su sistema operativo, el mensaje de error exacto y registros relevantes. En macOS, verifique Conso-
le.app. En Windows, verifique Event Viewer → Windows Logs → Application.

Usar Claude Code con Chrome (beta)

Conecta Claude Code a tu navegador Chrome para probar aplicaciones web, depurar con registros de consola, automatizar el relleno de formularios y extraer datos de páginas web.

Claude Code se integra con la extensión del navegador Claude en Chrome para brindarte capacidades de automatización del navegador desde la CLI o la [extensión de VS Code](#). Construye tu código, luego prueba y depura en el navegador sin cambiar de contexto.

Claude abre nuevas pestañas para tareas del navegador y comparte el estado de inicio de sesión de tu navegador, por lo que puede acceder a cualquier sitio en el que ya hayas iniciado sesión. Las acciones del navegador se ejecutan en una ventana de Chrome visible en tiempo real. Cuando Claude encuentra una página de inicio de sesión o CAPTCHA, se detiene y te pide que lo manejes manualmente.

Note:

La integración de Chrome está en beta y actualmente funciona solo con Google Chrome. Aún no es compatible con Brave, Arc u otros navegadores basados en Chromium. WSL (Subsistema de Windows para Linux) tampoco es compatible.

Capacidades

Con Chrome conectado, puedes encadenar acciones del navegador con tareas de codificación en un único flujo de trabajo:

- **Depuración en vivo:** lee errores de consola y estado del DOM directamente, luego corrige el código que los causó
- **Verificación de diseño:** construye una interfaz de usuario a partir de un mock de Figma, luego ábrelo en el navegador para verificar que coincida
- **Prueba de aplicaciones web:** prueba la validación de formularios, verifica regresiones visuales o verifica flujos de usuario
- **Aplicaciones web autenticadas:** interactúa con Google Docs, Gmail, Notion o cualquier aplicación en la que hayas iniciado sesión sin conectores de API
- **Extracción de datos:** extrae información estructurada de páginas web y guárdala localmente

- **Automatización de tareas:** automatiza tareas repetitivas del navegador como entrada de datos, relleno de formularios o flujos de trabajo multisitio
- **Grabación de sesión:** graba interacciones del navegador como GIF para documentar o compartir lo que sucedió

Requisitos previos

Antes de usar Claude Code con Chrome, necesitas:

- Navegador [Google Chrome](#)
- Extensión [Claude en Chrome](#) versión 1.0.36 o superior
- [Claude Code](#) versión 2.0.73 o superior
- Un plan directo de Anthropic (Pro, Max, Team o Enterprise)

Note:

La integración de Chrome no está disponible a través de proveedores de terceros como Amazon Bedrock, Google Cloud Vertex AI o Microsoft Foundry. Si accedes a Claude exclusivamente a través de un proveedor de terceros, necesitas una cuenta separada de claude.ai para usar esta función.

Comenzar en la CLI

Step 1: Lanzar Claude Code con Chrome

Inicia Claude Code con la bandera `--chrome`:

```
claude --chrome
```

También puedes habilitar Chrome dentro de una sesión existente ejecutando `/chrome`.

Step 2: Pídele a Claude que use el navegador

Este ejemplo navega a una página, interactúa con ella e informa lo que encuentra, todo desde tu terminal o editor:

```
Go to code.claude.com/docs, click on the search box,  
type "hooks", and tell me what results appear
```

Ejecuta `/chrome` en cualquier momento para verificar el estado de la conexión, administrar permisos o reconectar la extensión.

Para VS Code, consulta [automatización del navegador en VS Code](#).

Habilitar Chrome de forma predeterminada

Para evitar pasar `--chrome` en cada sesión, ejecuta `/chrome` y selecciona “Habilitado de forma predeterminada”.

En la [extensión de VS Code](#), Chrome está disponible siempre que la extensión de Chrome esté instalada. No se necesita ninguna bandera adicional.

Note:

Habilitar Chrome de forma predeterminada en la CLI aumenta el uso del contexto ya que las herramientas del navegador siempre se cargan. Si notas un aumento en el consumo de contexto, deshabilita esta configuración y usa `--chrome` solo cuando sea necesario.

Administrar permisos del sitio

Los permisos a nivel de sitio se heredan de la extensión de Chrome. Administra los permisos en la configuración de la extensión de Chrome para controlar qué sitios puede examinar, hacer clic e introducir texto Claude.

Flujos de trabajo de ejemplo

Estos ejemplos muestran formas comunes de combinar acciones del navegador con tareas de codificación. Ejecuta `/mcp` y selecciona `claude-in-chrome` para ver la lista completa de herramientas del navegador disponibles.

Probar una aplicación web local

Al desarrollar una aplicación web, pídele a Claude que verifique que tus cambios funcionen correctamente:

```
I just updated the login form validation. Can you open localhost:3000,
try submitting the form with invalid data, and check if the error
messages appear correctly?
```

Claude navega a tu servidor local, interactúa con el formulario e informa lo que observa.

Depurar con registros de consola

Claude puede leer la salida de la consola para ayudar a diagnosticar problemas. Dile a Claude qué patrones buscar en lugar de pedirle toda la salida de la consola, ya que los registros pueden ser detallados:

```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude lee los mensajes de la consola y puede filtrar patrones específicos o tipos de error.

Automatizar el relleno de formularios

Acelera tareas repetitivas de entrada de datos:

```
I have a spreadsheet of customer contacts in contacts.csv. For each row,
go to the CRM at crm.example.com, click "Add Contact", and fill in the
name, email, and phone fields.
```

Claude lee tu archivo local, navega por la interfaz web e introduce los datos para cada registro.

Redactar contenido en Google Docs

Usa Claude para escribir directamente en tus documentos sin configuración de API:

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude abre el documento, hace clic en el editor e introduce el contenido. Esto funciona con cualquier aplicación web en la que hayas iniciado sesión: Gmail, Notion, Sheets y más.

Extraer datos de páginas web

Extrae información estructurada de sitios web:

```
Go to the product listings page and extract the name, price, and
availability for each item. Save the results as a CSV file.
```

Claude navega a la página, lee el contenido y compila los datos en un formato estructurado.

Ejecutar flujos de trabajo multisitio

Coordina tareas en múltiples sitios web:

```
Check my calendar for meetings tomorrow, then for each meeting with
an external attendee, look up their company website and add a note
about what they do.
```

Claude trabaja en pestañas para recopilar información y completar el flujo de trabajo.

Grabar un GIF de demostración

Crea grabaciones compartibles de interacciones del navegador:

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude graba la secuencia de interacción y la guarda como un archivo GIF.

Solución de problemas

Extensión no detectada

Si Claude Code muestra “Extensión de Chrome no detectada”:

1. Verifica que la extensión de Chrome esté instalada y habilitada en `chrome://extensions`
2. Verifica que Claude Code esté actualizado ejecutando `claude --version`
3. Comprueba que Chrome se está ejecutando
4. Ejecuta `/chrome` y selecciona “Reconectar extensión” para restablecer la conexión
5. Si el problema persiste, reinicia tanto Claude Code como Chrome

La primera vez que habilitas la integración de Chrome, Claude Code instala un archivo de configuración del host de mensajería nativa. Chrome lee este archivo al iniciarse, por lo que si la extensión no se detecta en tu primer intento, reinicia Chrome para recoger la nueva configuración.

Si la conexión aún falla, verifica que el archivo de configuración del host exista en:

- **macOS:** `~/Library/Application Support/Google/Chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`

- **Linux:** `~/ .config/google-chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Windows:** comprueba `HKCU\Software\Google\Chrome\NativeMessagingHosts\` en el Registro de Windows

El navegador no responde

Si los comandos del navegador de Claude dejan de funcionar:

1. Comprueba si un cuadro de diálogo modal (alerta, confirmación, solicitud) está bloqueando la página. Los cuadros de diálogo de JavaScript bloquean eventos del navegador e impiden que Claude reciba comandos. Descarta el cuadro de diálogo manualmente, luego dile a Claude que continúe.
2. Pídele a Claude que cree una nueva pestaña e intente de nuevo
3. Reinicia la extensión de Chrome deshabilitándola y volviéndola a habilitar en `chrome://extensions`

La conexión se cae durante sesiones largas

El trabajador de servicio de la extensión de Chrome puede quedarse inactivo durante sesiones extendidas, lo que rompe la conexión. Si las herramientas del navegador dejan de funcionar después de un período de inactividad, ejecuta `/chrome` y selecciona “Reconectar extensión”.

Problemas específicos de Windows

En Windows, puedes encontrar:

- **Conflictos de tuberías nombradas (EADDRINUSE):** si otro proceso está usando la misma tubería nombrada, reinicia Claude Code. Cierra cualquier otra sesión de Claude Code que pueda estar usando Chrome.
- **Errores del host de mensajería nativa:** si el host de mensajería nativa falla al iniciarse, intenta reinstalar Claude Code para regenerar la configuración del host.

Mensajes de error comunes

Estos son los errores más frecuentes y cómo resolverlos:

Error	Causa	Solución
“La extensión del navegador no está conectada”	El host de mensajería nativa no puede alcanzar la extensión	Reinicia Chrome y Claude Code, luego ejecuta <code>/chrome</code> para reconectar

Error	Causa	Solución
“Extensión no detectada”	La extensión de Chrome no está instalada o está deshabilitada	Instala o habilita la extensión en <code>chrome://extensions</code>
“No hay pestaña disponible”	Claude intentó actuar antes de que una pestaña estuviera lista	Pídele a Claude que cree una nueva pestaña e inténtelo de nuevo
“El extremo receptor no existe”	El trabajador de servicio de la extensión se quedó inactivo	Ejecuta <code>/chrome</code> y selecciona “Reconectar extensión”

Ver también

- [Usar Claude Code en VS Code](#): automatización del navegador en la extensión de VS Code
- [Referencia de CLI](#): banderas de línea de comandos incluyendo `--chrome`
- [Flujos de trabajo comunes](#): más formas de usar Claude Code
- [Datos y privacidad](#): cómo Claude Code maneja tus datos
- [Comenzar con Claude en Chrome](#): documentación completa para la extensión de Chrome, incluyendo atajos de teclado, programación y permisos

Claude Code en Slack

Delega tareas de codificación directamente desde tu espacio de trabajo de Slack

Claude Code en Slack trae el poder de Claude Code directamente a tu espacio de trabajo de Slack. Cuando mencionas `@Claude` con una tarea de codificación, Claude detecta automáticamente la intención y crea una sesión de Claude Code en la web, permitiéndote delegar trabajo de desarrollo sin salir de tus conversaciones de equipo.

Esta integración se basa en la aplicación Claude para Slack existente pero agrega enrutamiento inteligente a Claude Code en la web para solicitudes relacionadas con codificación.

Casos de uso

- **Investigación y corrección de errores:** Pídele a Claude que investigue y corrija errores tan pronto como se reporten en los canales de Slack.
- **Revisiones de código rápidas y modificaciones:** Haz que Claude implemente pequeñas características o refactorice código basado en comentarios del equipo.
- **Depuración colaborativa:** Cuando las discusiones del equipo proporcionan contexto crucial (por ejemplo, reproducciones de errores o reportes de usuarios), Claude puede usar esa información para informar su enfoque de depuración.
- **Ejecución de tareas en paralelo:** Inicia tareas de codificación en Slack mientras continúas con otro trabajo, recibiendo notificaciones cuando se completan.

Requisitos previos

Antes de usar Claude Code en Slack, asegúrate de tener lo siguiente:

Requisito	Detalles
Plan de Claude	Pro, Max, Team o Enterprise con acceso a Claude Code (asientos premium)
Claude Code en la web	El acceso a Claude Code en la web debe estar habilitado
Cuenta de GitHub	Conectada a Claude Code en la web con al menos un repositorio autenticado

Requisito	Detalles
Autenticación de Slack	Tu cuenta de Slack vinculada a tu cuenta de Claude a través de la aplicación Claude

Configuración de Claude Code en Slack

Step 1: Instala la aplicación Claude en Slack

Un administrador del espacio de trabajo debe instalar la aplicación Claude desde el Slack App Marketplace. Visita el [Slack App Marketplace](#) y haz clic en “Add to Slack” para comenzar el proceso de instalación.

Step 2: Conecta tu cuenta de Claude

Después de que la aplicación esté instalada, autentica tu cuenta individual de Claude:

1. Abre la aplicación Claude en Slack haciendo clic en “Claude” en tu sección de Aplicaciones
2. Navega a la pestaña App Home
3. Haz clic en “Connect” para vincular tu cuenta de Slack con tu cuenta de Claude
4. Completa el flujo de autenticación en tu navegador

Step 3: Configura Claude Code en la web

Asegúrate de que tu Claude Code en la web esté correctamente configurado:

- Visita [claude.ai/code](#) e inicia sesión con la misma cuenta que conectaste a Slack
- Conecta tu cuenta de GitHub si aún no está conectada
- Autentica al menos un repositorio con el que quieras que Claude trabaje

Step 4: Elige tu modo de enrutamiento

Después de conectar tus cuentas, configura cómo Claude maneja tus mensajes en Slack. Navega a la App Home de Claude en Slack para encontrar la configuración de **Routing Mode**.

Modo	Comportamiento
Code only	Claude enruta todas las @menciones a sesiones de Claude Code. Mejor para equipos que usan Claude en Slack exclusivamente para tareas de desarrollo.

Modo	Comportamiento
Code + Chat	Claude analiza cada mensaje y enruta inteligentemente entre Claude Code (para tareas de codificación) y Claude Chat (para escritura, análisis y preguntas generales). Mejor para equipos que quieren un único punto de entrada @Claude para todos los tipos de trabajo.

Note:

En modo Code + Chat, si Claude enruta un mensaje a Chat pero querías una sesión de codificación, puedes hacer clic en “Retry as Code” para crear una sesión de Claude Code en su lugar. De manera similar, si se enruta a Code pero querías una sesión de Chat, puedes elegir esa opción en ese hilo.

Cómo funciona

Detección automática

Cuando mencionas @Claude en un canal o hilo de Slack, Claude analiza automáticamente tu mensaje para determinar si es una tarea de codificación. Si Claude detecta intención de codificación, enrutará tu solicitud a Claude Code en la web en lugar de responder como un asistente de chat regular.

También puedes decirle explícitamente a Claude que maneje una solicitud como una tarea de codificación, incluso si no la detecta automáticamente.

Note:

Claude Code en Slack solo funciona en canales (públicos o privados). No funciona en mensajes directos (DMs).

Recopilación de contexto

De hilos: Cuando @mencionas a Claude en un hilo, recopila contexto de todos los mensajes en ese hilo para entender la conversación completa.

De canales: Cuando se menciona directamente en un canal, Claude observa los mensajes recientes del canal para obtener contexto relevante.

Este contexto ayuda a Claude a entender el problema, seleccionar el repositorio apropiado e informar su enfoque para la tarea.

Warning:

Cuando @Claude se invoca en Slack, Claude tiene acceso al contexto de la conversación para entender mejor tu solicitud. Claude puede seguir direcciones de otros mensajes en el contexto, por lo que los usuarios deben asegurarse de usar Claude solo en conversaciones de Slack de confianza.

Flujo de sesión

1. **Iniciación:** @mencionas a Claude con una solicitud de codificación
2. **Detección:** Claude analiza tu mensaje y detecta intención de codificación
3. **Creación de sesión:** Se crea una nueva sesión de Claude Code en claude.ai/code
4. **Actualizaciones de progreso:** Claude publica actualizaciones de estado en tu hilo de Slack a medida que avanza el trabajo
5. **Finalización:** Cuando termina, Claude te @menciona con un resumen y botones de acción
6. **Revisión:** Haz clic en “View Session” para ver la transcripción completa, o “Create PR” para abrir una solicitud de extracción

Elementos de la interfaz de usuario

App Home

La pestaña App Home muestra tu estado de conexión y te permite conectar o desconectar tu cuenta de Claude de Slack.

Acciones de mensaje

- **View Session:** Abre la sesión completa de Claude Code en tu navegador donde puedes ver todo el trabajo realizado, continuar la sesión o hacer solicitudes adicionales.
- **Create PR:** Crea una solicitud de extracción directamente desde los cambios de la sesión.
- **Retry as Code:** Si Claude inicialmente responde como un asistente de chat pero querías una sesión de codificación, haz clic en este botón para reintentar la solicitud como una tarea de Claude Code.
- **Change Repo:** Te permite seleccionar un repositorio diferente si Claude eligió incorrectamente.

Selección de repositorio

Claude selecciona automáticamente un repositorio basado en el contexto de tu conversación de Slack. Si múltiples repositorios podrían aplicarse, Claude puede mostrar un menú desplegable permitiéndote elegir el correcto.

Acceso y permisos

Acceso a nivel de usuario

Tipo de acceso	Requisito
Sesiones de Claude Code	Cada usuario ejecuta sesiones bajo su propia cuenta de Claude
Uso y límites de velocidad	Las sesiones cuentan contra los límites del plan del usuario individual
Acceso al repositorio	Los usuarios solo pueden acceder a repositorios que han conectado personalmente
Historial de sesiones	Las sesiones aparecen en tu historial de Claude Code en claude.ai/code

Permisos de administrador del espacio de trabajo

Los administradores del espacio de trabajo de Slack controlan si la aplicación Claude puede instalarse en el espacio de trabajo. Los usuarios individuales luego se autentican con sus propias cuentas de Claude para usar la integración.

Qué es accesible dónde

En Slack: Verás actualizaciones de estado, resúmenes de finalización y botones de acción. La transcripción completa se conserva y siempre es accesible.

En la web: La sesión completa de Claude Code con historial de conversación completo, todos los cambios de código, operaciones de archivo y la capacidad de continuar la sesión o crear solicitudes de extracción.

Mejores prácticas

Escribir solicitudes efectivas

- **Sé específico:** Incluye nombres de archivos, nombres de funciones o mensajes de error cuando sea relevante.

- **Proporciona contexto:** Menciona el repositorio o proyecto si no está claro en la conversación.
- **Define el éxito:** Explica qué significa “hecho”—¿debería Claude escribir pruebas? ¿Actualizar documentación? ¿Crear un PR?
- **Usa hilos:** Responde en hilos cuando discutas errores o características para que Claude pueda recopilar el contexto completo.

Cuándo usar Slack vs. web

Usa Slack cuando: El contexto ya existe en una discusión de Slack, quieres iniciar una tarea de forma asincrónica, o estás colaborando con compañeros de equipo que necesitan visibilidad.

Usa la web directamente cuando: Necesitas cargar archivos, quieres interacción en tiempo real durante el desarrollo, o estás trabajando en tareas más largas y complejas.

Solución de problemas

Las sesiones no se inician

1. Verifica que tu cuenta de Claude esté conectada en la App Home de Claude
2. Comprueba que tengas acceso a Claude Code en la web habilitado
3. Asegúrate de tener al menos un repositorio de GitHub conectado a Claude Code

El repositorio no se muestra

1. Conecta el repositorio en Claude Code en la web en claude.ai/code
2. Verifica tus permisos de GitHub para ese repositorio
3. Intenta desconectar y reconectar tu cuenta de GitHub

Se seleccionó el repositorio incorrecto

1. Haz clic en el botón “Change Repo” para seleccionar un repositorio diferente
2. Incluye el nombre del repositorio en tu solicitud para una selección más precisa

Errores de autenticación

1. Desconecta y reconecta tu cuenta de Claude en la App Home
2. Asegúrate de estar conectado a la cuenta de Claude correcta en tu navegador
3. Comprueba que tu plan de Claude incluya acceso a Claude Code

Expiración de sesión

1. Las sesiones permanecen accesibles en tu historial de Claude Code en la web

2. Puedes continuar o hacer referencia a sesiones pasadas desde claude.ai/code

Limitaciones actuales

- **Solo GitHub:** Actualmente admite repositorios en GitHub.
- **Un PR a la vez:** Cada sesión puede crear una solicitud de extracción.
- **Se aplican límites de velocidad:** Las sesiones usan los límites de velocidad del plan de Claude individual.
- **Se requiere acceso web:** Los usuarios deben tener acceso a Claude Code en la web; aquellos sin él solo recibirán respuestas de chat estándar de Claude.

Recursos relacionados

- [Claude Code en la web](#) Obtén más información sobre Claude Code en la web
- [Claude para Slack](#) Documentación general de Claude para Slack
- [Slack App Marketplace](#) Instala la aplicación Claude desde el Slack Marketplace
- [Centro de ayuda de Claude](#) Obtén soporte adicional

Claude Code en la web

Ejecuta tareas de Claude Code de forma asincrónica en infraestructura en la nube segura

Note:

Claude Code en la web está actualmente en vista previa de investigación.

¿Qué es Claude Code en la web?

Claude Code en la web permite a los desarrolladores iniciar Claude Code desde la aplicación Claude. Esto es perfecto para:

- **Responder preguntas:** Pregunta sobre la arquitectura del código y cómo se implementan las características
- **Correcciones de errores y tareas rutinarias:** Tareas bien definidas que no requieren dirección frecuente
- **Trabajo en paralelo:** Aborda múltiples correcciones de errores en paralelo
- **Repositorios no en su máquina local:** Trabaja en código que no tiene desprotegido localmente
- **Cambios de backend:** Donde Claude Code puede escribir pruebas y luego escribir código para pasar esas pruebas

Claude Code también está disponible en la aplicación Claude para [iOS](#) y [Android](#) para iniciar tareas sobre la marcha y monitorear el trabajo en progreso.

Puede [iniciar nuevas tareas en la web desde su terminal](#) con `--remote`, o [teleportar sesiones web de vuelta a su terminal](#) para continuar localmente. Para usar la interfaz web mientras ejecuta Claude Code en su propia máquina en lugar de infraestructura en la nube, consulte [Control Remoto](#).

¿Quién puede usar Claude Code en la web?

Claude Code en la web está disponible en vista previa de investigación para:

- **Usuarios Pro**
- **Usuarios Max**

- **Usuarios de Team**
- **Usuarios Enterprise** con asientos premium o asientos Chat + Claude Code

Primeros pasos

1. Visite claude.ai/code
2. Conecte su cuenta de GitHub
3. Instale la aplicación Claude GitHub en sus repositorios
4. Seleccione su entorno predeterminado
5. Envíe su tarea de codificación
6. Revise los cambios en la vista de diferencias, itere con comentarios y luego cree una solicitud de extracción

Cómo funciona

Cuando inicia una tarea en Claude Code en la web:

1. **Clonación de repositorio:** Su repositorio se clona en una máquina virtual administrada por Anthropic
2. **Configuración del entorno:** Claude prepara un entorno en la nube seguro con su código, luego ejecuta su [script de configuración](#) si está configurado
3. **Configuración de red:** El acceso a Internet se configura según su configuración
4. **Ejecución de tareas:** Claude analiza el código, realiza cambios, ejecuta pruebas y verifica su trabajo
5. **Finalización:** Se le notifica cuando finaliza y puede crear una PR con los cambios
6. **Resultados:** Los cambios se insertan en una rama, listos para la creación de solicitud de extracción

Revise los cambios con la vista de diferencias

La vista de diferencias le permite ver exactamente qué cambió Claude antes de crear una solicitud de extracción. En lugar de hacer clic en “Crear PR” para revisar cambios en GitHub, vea la diferencia directamente en la aplicación e itere con Claude hasta que los cambios estén listos.

Cuando Claude realiza cambios en archivos, aparece un indicador de estadísticas de diferencias que muestra el número de líneas agregadas y eliminadas (por ejemplo, **+12 -1**). Seleccione este indicador para abrir el visor de diferencias, que muestra una lista de archivos a la izquierda y los cambios para cada archivo a la derecha.

Desde la vista de diferencias, puede:

- Revisar cambios archivo por archivo
- Comentar sobre cambios específicos para solicitar modificaciones
- Continuar iterando con Claude según lo que vea

Esto le permite refinar cambios a través de múltiples rondas de retroalimentación sin crear PRs de borrador o cambiar a GitHub.

Mover tareas entre web y terminal

Puede iniciar nuevas tareas en la web desde su terminal, o extraer sesiones web en su terminal para continuar localmente. Las sesiones web persisten incluso si cierra su portátil, y puede monitorearlas desde cualquier lugar, incluida la aplicación móvil Claude.

Note:

La transferencia de sesión es unidireccional: puede extraer sesiones web en su terminal, pero no puede insertar una sesión de terminal existente en la web. La bandera `--remote` crea una *nueva* sesión web para su repositorio actual.

De terminal a web

Inicie una sesión web desde la línea de comandos con la bandera `--remote`:

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

Esto crea una nueva sesión web en claude.ai. La tarea se ejecuta en la nube mientras continúa trabajando localmente. Use `/tasks` para verificar el progreso, o abra la sesión en claude.ai o la aplicación móvil Claude para interactuar directamente. Desde allí puede dirigir Claude, proporcionar retroalimentación o responder preguntas como en cualquier otra conversación.

Consejos para tareas remotas

Planifique localmente, ejecute remotamente: Para tareas complejas, inicie Claude en modo de plan para colaborar en el enfoque, luego envíe el trabajo a la web:

```
claude --permission-mode plan
```

En modo de plan, Claude solo puede leer archivos y explorar la base de código. Una vez que esté satisfecho con el plan, inicie una sesión remota para ejecución autónoma:

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

Este patrón le da control sobre la estrategia mientras permite que Claude ejecute de forma autónoma en la nube.

Ejecute tareas en paralelo: Cada comando `--remote` crea su propia sesión web que se ejecuta de forma independiente. Puede iniciar múltiples tareas y todas se ejecutarán simultáneamente en sesiones separadas:

```
claude --remote "Fix the flaky test in auth.spec.ts"
claude --remote "Update the API documentation"
claude --remote "Refactor the logger to use structured output"
```

Monitoree todas las sesiones con `/tasks`. Cuando una sesión se completa, puede crear una PR desde la interfaz web o [teleportar](#) la sesión a su terminal para continuar trabajando.

De web a terminal

Hay varias formas de extraer una sesión web en su terminal:

- **Usando `/teleport`**: Desde Claude Code, ejecute `/teleport` (o `/tp`) para ver un selector interactivo de sus sesiones web. Si tiene cambios sin confirmar, se le pedirá que los guarde primero.
- **Usando `--teleport`**: Desde la línea de comandos, ejecute `claude --teleport` para un selector de sesión interactivo, o `claude --teleport <session-id>` para reanudar una sesión específica directamente.
- **Desde `/tasks`**: Ejecute `/tasks` para ver sus sesiones de fondo, luego presione `t` para teleportarse a una
- **Desde la interfaz web**: Haga clic en “Abrir en CLI” para copiar un comando que puede pegar en su terminal

Cuando teleporta una sesión, Claude verifica que esté en el repositorio correcto, obtiene y verifica la rama de la sesión remota, y carga el historial de conversación completo en su terminal.

Requisitos para teleportar

Teleport verifica estos requisitos antes de reanudar una sesión. Si algún requisito no se cumple, verá un error o se le pedirá que resuelva el problema.

Requisi-to	Detalles
Estado de git limpio	Su directorio de trabajo no debe tener cambios sin confirmar. Teleport le pide que guarde los cambios si es necesario.
Reposito-rio correc-to	Debe ejecutar <code>--teleport</code> desde una desprotección del mismo repositorio, no desde una bifurcación.
Rama dis-ponible	La rama de la sesión web debe haber sido insertada en el remoto. Teleport la obtiene y verifica automáticamente.
Misma cuenta	Debe estar autenticado en la misma cuenta de Claude.ai utilizada en la sesión web.

Compartir sesiones

Para compartir una sesión, alterne su visibilidad según los tipos de cuenta a continuación. Después de eso, comparta el enlace de sesión tal como está. Los destinatarios que abran su sesión compartida verán el estado más reciente de la sesión al cargar, pero la página del destinatario no se actualizará en tiempo real.

Compartir desde una cuenta Enterprise o Teams

Para cuentas Enterprise y Teams, las dos opciones de visibilidad son **Privada** y **Team**. La visibilidad de Team hace que la sesión sea visible para otros miembros de su organización de Claude.ai. La verificación de acceso al repositorio está habilitada de forma predeterminada, según la cuenta de GitHub conectada a la cuenta del destinatario. El nombre para mostrar de su cuenta es visible para todos los destinatarios con acceso. Las sesiones de [Claude en Slack](#) se comparten automáticamente con visibilidad de Team.

Compartir desde una cuenta Max o Pro

Para cuentas Max y Pro, las dos opciones de visibilidad son **Privada** y **Pública**. La visibilidad pública hace que la sesión sea visible para cualquier usuario que haya iniciado sesión en claude.ai.

Verifique su sesión para contenido sensible antes de compartir. Las sesiones pueden contener código y credenciales de repositorios privados de GitHub. La verificación de acceso al repositorio no está habilitada de forma predeterminada.

Habilite la verificación de acceso al repositorio y/o retenga su nombre de sus sesiones compartidas yendo a Configuración > Claude Code > Configuración de uso compartido.

Administrar sesiones

Archivar sesiones

Puede archivar sesiones para mantener su lista de sesiones organizada. Las sesiones archivadas se ocultan de la lista de sesiones predeterminada pero se pueden ver filtrando sesiones archivadas.

Para archivar una sesión, pase el cursor sobre la sesión en la barra lateral y haga clic en el icono de archivo.

Eliminar sesiones

Eliminar una sesión elimina permanentemente la sesión y sus datos. Esta acción no se puede deshacer. Puede eliminar una sesión de dos formas:

- **Desde la barra lateral:** Filtre sesiones archivadas, luego pase el cursor sobre la sesión que desea eliminar y haga clic en el icono de eliminar
- **Desde el menú de sesión:** Abra una sesión, haga clic en el menú desplegable junto al título de la sesión y seleccione **Eliminar**

Se le pedirá que confirme antes de que se elimine una sesión.

Entorno en la nube

Imagen predeterminada

Construimos y mantenemos una imagen universal con cadenas de herramientas comunes y ecosistemas de lenguaje preinstalados. Esta imagen incluye:

- Lenguajes de programación populares y tiempos de ejecución
- Herramientas de compilación comunes y administradores de paquetes
- Marcos de prueba y linters

Verificar herramientas disponibles

Para ver qué está preinstalado en su entorno, pida a Claude Code que ejecute:

```
check-tools
```

Este comando muestra:

- Lenguajes de programación y sus versiones
- Administradores de paquetes disponibles
- Herramientas de desarrollo instaladas

Configuraciones específicas del idioma

La imagen universal incluye entornos preconfigurados para:

- **Python:** Python 3.x con pip, poetry y bibliotecas científicas comunes
- **Node.js:** Últimas versiones LTS con npm, yarn, pnpm y bun
- **Ruby:** Versiones 3.1.6, 3.2.6, 3.3.6 (predeterminado: 3.3.6) con gem, bundler y rbenv para gestión de versiones
- **PHP:** Versión 8.4.14
- **Java:** OpenJDK con Maven y Gradle
- **Go:** Última versión estable con soporte de módulos
- **Rust:** Cadena de herramientas Rust con cargo
- **C++:** Compiladores GCC y Clang

Bases de datos

La imagen universal incluye las siguientes bases de datos:

- **PostgreSQL:** Versión 16
- **Redis:** Versión 7.0

Configuración del entorno

Cuando inicia una sesión en Claude Code en la web, esto es lo que sucede bajo el capó:

1. **Preparación del entorno:** Clonamos su repositorio y ejecutamos cualquier [script de configuración](#) configurado. El repositorio se clonará con la rama predeterminada en su repositorio de GitHub. Si desea desproteger una rama específica, puede especificar eso en el mensaje.
2. **Configuración de red:** Configuramos el acceso a Internet para el agente. El acceso a Internet es limitado de forma predeterminada, pero puede configurar el entorno para que no tenga acceso a Internet o acceso completo a Internet según sus necesidades.

- 3. Ejecución de Claude Code:** Claude Code se ejecuta para completar su tarea, escribiendo código, ejecutando pruebas y verificando su trabajo. Puede guiar y dirigir Claude durante toda la sesión a través de la interfaz web. Claude respeta el contexto que ha definido en su `CLAUDE.md`.
- 4. Resultado:** Cuando Claude completa su trabajo, insertará la rama en remoto. Podrá crear una PR para la rama.

Note:

Claude opera completamente a través del terminal y las herramientas CLI disponibles en el entorno. Utiliza las herramientas preinstaladas en la imagen universal y cualquier herramienta adicional que instale a través de hooks o gestión de dependencias.

Para agregar un nuevo entorno: Seleccione el entorno actual para abrir el selector de entorno y luego seleccione “Agregar entorno”. Esto abrirá un diálogo donde puede especificar el nombre del entorno, el nivel de acceso de red, las variables de entorno y un [script de configuración](#).

Para actualizar un entorno existente: Seleccione el entorno actual, a la derecha del nombre del entorno, y seleccione el botón de configuración. Esto abrirá un diálogo donde puede actualizar el nombre del entorno, el acceso de red, las variables de entorno y el script de configuración.

Para seleccionar su entorno predeterminado desde el terminal: Si tiene múltiples entornos configurados, ejecute `/remote-env` para elegir cuál usar al iniciar sesiones web desde su terminal con `--remote`. Con un único entorno, este comando muestra su configuración actual.

Note:

Las variables de entorno deben especificarse como pares clave-valor, en [formato](#) `.env`. Por ejemplo:

```
API_KEY=your_api_key
DEBUG=true
```

Scripts de configuración

Un script de configuración es un script Bash que se ejecuta cuando comienza una nueva sesión en la nube, antes de que se lance Claude Code. Use scripts de configuración para instalar dependencias, configurar herramientas o preparar cualquier cosa que el entorno en la nube necesite que no esté en la [imagen predeterminada](#).

Los scripts se ejecutan como root en Ubuntu 24.04, por lo que `apt install` y la mayoría de los administradores de paquetes de lenguaje funcionan.

Tip:

Para verificar qué ya está instalado antes de agregarlo a su script, pida a Claude que ejecute `check-tools` en una sesión en la nube.

Para agregar un script de configuración, abra el diálogo de configuración del entorno e ingrese su script en el campo **Script de configuración**.

Este ejemplo instala la CLI `gh`, que no está en la imagen predeterminada:

```
#!/bin/bash
apt update && apt install -y gh
```

Los scripts de configuración se ejecutan solo cuando se crea una nueva sesión. Se omiten cuando se reanuda una sesión existente.

Si el script sale con un código distinto de cero, la sesión no se inicia. Agregue `|| true` a comandos no críticos para evitar bloquear la sesión en una instalación inestable.

Note:

Los scripts de configuración que instalan paquetes necesitan acceso a la red para llegar a los registros. El acceso a la red predeterminado permite conexiones a [registros de paquetes comunes](#) incluyendo npm, PyPI, RubyGems y crates.io. Los scripts fallarán al instalar paquetes si su entorno tiene acceso a la red deshabilitado.

Scripts de configuración vs. hooks SessionStart

Use un script de configuración para instalar cosas que la nube necesita pero su portátil ya tiene, como un tiempo de ejecución de lenguaje o herramienta CLI. Use un [hook SessionStart](#) para la configuración del proyecto que debe ejecutarse en todas partes, nube y local, como `npm install`.

Ambos se ejecutan al inicio de una sesión, pero pertenecen a diferentes lugares:

	Scripts de configuración	Hooks SessionStart
Adjunto a	El entorno en la nube	Su repositorio
Configurado en	Interfaz de usuario del entorno en la nube	<code>.claude/settings.json</code> en su repositorio
Se ejecuta	Antes de que se lance Claude Code, solo en sesiones nuevas	Después de que se lance Claude Code, en cada sesión incluyendo reanudadas
Alcance	Solo entornos en la nube	Tanto local como nube

Los hooks SessionStart también se pueden definir en su `~/.claude/settings.json` a nivel de usuario localmente, pero la configuración a nivel de usuario no se transfiere a sesiones en la nube. En la nube, solo se ejecutan los hooks comprometidos con el repositorio.

Gestión de dependencias

Las imágenes de entorno personalizadas y las instantáneas aún no son compatibles. Use [scripts de configuración](#) para instalar paquetes cuando comienza una sesión, o [hooks SessionStart](#) para la instalación de dependencias que también debe ejecutarse en entornos locales. Los hooks SessionStart tienen [limitaciones conocidas](#).

Para configurar la instalación automática de dependencias con un script de configuración, abra la configuración de su entorno y agregue un script:

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

Alternativamente, puede usar hooks SessionStart en el archivo `.claude/settings.json` de su repositorio para la instalación de dependencias que también debe ejecutarse en entornos locales:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

Cree el script correspondiente en `scripts/install_pkgs.sh`:

```
#!/bin/bash

## Only run in remote environments
if [ "$CLAUDE_CODE_REMOTE" ≠ "true" ]; then
  exit 0
fi

npm install
pip install -r requirements.txt
exit 0
```

Hágalo ejecutable: `chmod +x scripts/install_pkgs.sh`

Persistir variables de entorno

Los hooks SessionStart pueden persistir variables de entorno para comandos Bash posteriores escribiendo en el archivo especificado en la variable de entorno `CLAUDE_ENV_FILE`. Para más detalles, consulte [hooks SessionStart](#) en la referencia de hooks.

Limitaciones de gestión de dependencias

- **Los hooks se disparan para todas las sesiones:** Los hooks SessionStart se ejecutan en entornos locales y remotos. No hay configuración de hook para limitar un hook solo a sesiones remotas. Para omitir la ejecución local, verifique la variable de entorno `CLAUDE_CODE_REMOTE` en su script como se muestra arriba.
- **Requiere acceso a la red:** Los comandos de instalación necesitan acceso a la red para llegar a los registros de paquetes. Si su entorno está configurado con acceso “Sin internet”, estos hooks fallarán. Use acceso de red “Limitado” (el predeterminado) o “Completo”. La [lista de permitidos predeterminada](#) incluye registros comunes como npm, PyPI, RubyGems y crates.io.
- **Compatibilidad de proxy:** Todo el tráfico saliente en entornos remotos pasa a través de un [proxy de seguridad](#). Algunos administradores de paquetes no funcionan correctamente con este proxy. Bun es un ejemplo conocido.
- **Se ejecuta en cada inicio de sesión:** Los hooks se ejecutan cada vez que comienza o se reanuda una sesión, agregando latencia de inicio. Mantenga los scripts de instalación rápidos verificando si las dependencias ya están presentes antes de reinstalar.

Acceso a la red y seguridad

Política de red

Proxy de GitHub

Por seguridad, todas las operaciones de GitHub pasan a través de un servicio de proxy dedicado que maneja de forma transparente todas las interacciones de git. Dentro del sandbox, el cliente de git se autentica usando una credencial de alcance personalizada. Este proxy:

- Gestiona la autenticación de GitHub de forma segura: el cliente de git usa una credencial de alcance dentro del sandbox, que el proxy verifica y traduce a su token de autenticación real de GitHub
- Restringe las operaciones de inserción de git a la rama de trabajo actual por seguridad
- Permite operaciones de clonación, obtención y PR sin problemas mientras mantiene límites de seguridad

Proxy de seguridad

Los entornos se ejecutan detrás de un proxy de red HTTP/HTTPS para propósitos de seguridad y prevención de abuso. Todo el tráfico de Internet saliente pasa a través de este proxy, que proporciona:

- Protección contra solicitudes maliciosas
- Limitación de velocidad y prevención de abuso
- Filtrado de contenido para mayor seguridad

Niveles de acceso

De forma predeterminada, el acceso a la red se limita a [dominios en la lista de permitidos](#).

Puede configurar acceso a la red personalizado, incluyendo deshabilitar el acceso a la red.

Dominios permitidos predeterminados

Cuando se usa acceso de red “Limitado”, los siguientes dominios están permitidos de forma predeterminada:

Servicios Anthropic

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

Control de versiones

- github.com
- www.github.com
- api.github.com
- npm.pkg.github.com
- raw.githubusercontent.com
- pkg-npm.githubusercontent.com
- objects.githubusercontent.com
- codeload.github.com
- avatars.githubusercontent.com
- camo.githubusercontent.com
- gist.github.com

- gitlab.com
- www.gitlab.com
- registry.gitlab.com
- bitbucket.org
- www.bitbucket.org
- api.bitbucket.org

Registros de contenedores

- registry-1.docker.io
- auth.docker.io
- index.docker.io
- hub.docker.com
- www.docker.com
- production.cloudflare.docker.com
- download.docker.com
- gcr.io
- *.gcr.io
- ghcr.io
- mcr.microsoft.com
- *.data.mcr.microsoft.com
- public.ecr.aws

Plataformas en la nube

- cloud.google.com
- accounts.google.com
- gcloud.google.com
- *.googleapis.com
- storage.googleapis.com
- compute.googleapis.com
- container.googleapis.com
- azure.com
- portal.azure.com
- microsoft.com
- www.microsoft.com

- *.microsoftonline.com
- packages.microsoft.com
- dotnet.microsoft.com
- dot.net
- visualstudio.com
- dev.azure.com
- *.amazonaws.com
- *.api.aws
- oracle.com
- www.oracle.com
- java.com
- www.java.com
- java.net
- www.java.net
- download.oracle.com
- yum.oracle.com

Administradores de paquetes - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

Administradores de paquetes - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org
- pypa.io

- www.pypa.io

Administradores de paquetes - Ruby

- rubygems.org
- www.rubygems.org
- api.rubygems.org
- index.rubygems.org
- ruby-lang.org
- www.ruby-lang.org
- rubyforge.org
- www.rubyforge.org
- rubyonrails.org
- www.rubyonrails.org
- rvm.io
- get.rvm.io

Administradores de paquetes - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

Administradores de paquetes - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

Administradores de paquetes - JVM

- maven.org
- repo.maven.org
- central.maven.org
- repo1.maven.org
- jcenter.bintray.com
- gradle.org
- www.gradle.org
- services.gradle.org
- plugins.gradle.org
- kotlin.org
- www.kotlin.org
- spring.io
- repo.spring.io

Administradores de paquetes - Otros idiomas

- packagist.org (PHP Composer)
- www.packagist.org
- repo.packagist.org
- nuget.org (.NET NuGet)
- www.nuget.org
- api.nuget.org
- pub.dev (Dart/Flutter)
- api.pub.dev
- hex.pm (Elixir/Erlang)
- www.hex.pm
- cpan.org (Perl CPAN)
- www.cpan.org
- metacpan.org
- www.metacpan.org
- api.metacpan.org
- cocoapods.org (iOS/macOS)
- www.cocoapods.org

- cdn.cocoapods.org
- haskell.org
- www.haskell.org
- hackage.haskell.org
- swift.org
- www.swift.org

Distribuciones de Linux

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

Herramientas de desarrollo y plataformas

- dl.k8s.io (Kubernetes)
- pkgs.k8s.io
- k8s.io
- www.k8s.io
- releases.hashicorp.com (HashiCorp)
- apt.releases.hashicorp.com
- rpm.releases.hashicorp.com
- archive.releases.hashicorp.com
- hashicorp.com
- www.hashicorp.com
- repo.anaconda.com (Anaconda/Conda)
- conda.anaconda.org
- anaconda.org
- www.anaconda.com
- anaconda.com
- continuum.io

- apache.org (Apache)
- www.apache.org
- archive.apache.org
- downloads.apache.org
- eclipse.org (Eclipse)
- www.eclipse.org
- download.eclipse.org
- nodejs.org (Node.js)
- www.nodejs.org

Servicios en la nube y monitoreo

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

Entrega de contenido y espejos

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

Esquema y configuración

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Protocolo de contexto de modelo

- *.modelcontextprotocol.io

Note:

Los dominios marcados con * indican coincidencia de subdominio comodín. Por ejemplo, *.gcr.io permite acceso a cualquier subdominio de gcr.io.

Mejores prácticas de seguridad para acceso a la red personalizado

1. **Principio de menor privilegio:** Solo habilite el acceso a la red mínimo requerido
2. **Audite regularmente:** Revise los dominios permitidos periódicamente
3. **Use HTTPS:** Siempre prefiera puntos finales HTTPS sobre HTTP

Seguridad y aislamiento

Claude Code en la web proporciona garantías de seguridad sólidas:

- **Máquinas virtuales aisladas:** Cada sesión se ejecuta en una VM aislada administrada por Anthropic
- **Controles de acceso a la red:** El acceso a la red es limitado de forma predeterminada y puede deshabilitarse

Note:

Cuando se ejecuta con acceso a la red deshabilitado, Claude Code puede comunicarse con la API de Anthropic, lo que aún puede permitir que los datos salgan de la VM aislada de Claude Code.

- **Protección de credenciales:** Las credenciales sensibles (como credenciales de git o claves de firma) nunca están dentro del sandbox con Claude Code. La autenticación se maneja a través de un proxy seguro usando credenciales de alcance
- **Análisis seguro:** El código se analiza y modifica dentro de VMs aisladas antes de crear PRs

Precios y límites de velocidad

Claude Code en la web comparte límites de velocidad con todo otro uso de Claude y Claude Code dentro de su cuenta. Ejecutar múltiples tareas en paralelo consumirá más límites de velocidad proporcionalmente.

Limitaciones

- **Autenticación de repositorio:** Solo puede mover sesiones de web a local cuando está autenticado en la misma cuenta

- **Restricciones de plataforma:** Claude Code en la web solo funciona con código alojado en GitHub. Los repositorios de GitLab y otros que no sean GitHub no se pueden usar con sesiones en la nube

Mejores prácticas

1. **Automatice la configuración del entorno:** Use [scripts de configuración](#) para instalar dependencias y configurar herramientas antes de que se lance Claude Code. Para escenarios más avanzados, configure [hooks SessionStart](#).
2. **Documente los requisitos:** Especifique claramente las dependencias y comandos en su archivo `CLAUDE.md`. Si tiene un archivo `AGENTS.md`, puede obtenerlo en su `CLAUDE.md` usando `@AGENTS.md` para mantener una única fuente de verdad.

Recursos relacionados

- [Configuración de hooks](#)
- [Referencia de configuración](#)
- [Seguridad](#)
- [Uso de datos](#)

Part 9: Security & Privacy

Seguridad

Aprenda sobre las medidas de seguridad de Claude Code y las mejores prácticas para un uso seguro.

Cómo abordamos la seguridad

Fundamento de seguridad

La seguridad de su código es primordial. Claude Code está construido con la seguridad en su núcleo, desarrollado de acuerdo con el programa de seguridad integral de Anthropic. Obtenga más información y acceda a recursos (informe SOC 2 Type 2, certificado ISO 27001, etc.) en [Anthropic Trust Center](#).

Arquitectura basada en permisos

Claude Code utiliza permisos de solo lectura estrictos de forma predeterminada. Cuando se necesitan acciones adicionales (editar archivos, ejecutar pruebas, ejecutar comandos), Claude Code solicita permiso explícito. Los usuarios controlan si aprobar acciones una sola vez o permitir las automáticamente.

Diseñamos Claude Code para ser transparente y seguro. Por ejemplo, requerimos aprobación para comandos bash antes de ejecutarlos, lo que le da control directo. Este enfoque permite a los usuarios y organizaciones configurar permisos directamente.

Para la configuración detallada de permisos, consulte [Permissions](#).

Protecciones integradas

Para mitigar riesgos en sistemas agénticos:

- **Herramienta bash en sandbox:** [Sandbox](#) comandos bash con aislamiento del sistema de archivos y red, reduciendo solicitudes de permiso mientras se mantiene la seguridad. Habilite con `/sandbox` para definir límites donde Claude Code puede trabajar de forma autónoma
- **Restricción de acceso de escritura:** Claude Code solo puede escribir en la carpeta donde se inició y sus subcarpetas—no puede modificar archivos en directorios principales sin permiso explícito. Aunque Claude Code puede leer archivos fuera del directo-

rio de trabajo (útil para acceder a bibliotecas del sistema y dependencias), las operaciones de escritura están estrictamente limitadas al alcance del proyecto, creando un límite de seguridad claro

- **Mitigación de fatiga de solicitudes:** Soporte para listas de permitidos de comandos seguros frecuentemente utilizados por usuario, por base de código u por organización
- **Modo Aceptar Ediciones:** Aceptar por lotes múltiples ediciones mientras se mantienen solicitudes de permiso para comandos con efectos secundarios

Responsabilidad del usuario

Claude Code solo tiene los permisos que usted le otorga. Usted es responsable de revisar el código y los comandos propuestos para verificar su seguridad antes de aprobarlos.

Protégase contra la inyección de solicitudes

La inyección de solicitudes es una técnica donde un atacante intenta anular o manipular las instrucciones de un asistente de IA insertando texto malicioso. Claude Code incluye varias medidas de protección contra estos ataques:

Protecciones principales

- **Sistema de permisos:** Las operaciones sensibles requieren aprobación explícita
- **Análisis consciente del contexto:** Detecta instrucciones potencialmente dañinas analizando la solicitud completa
- **Sanitización de entrada:** Previene la inyección de comandos procesando entradas del usuario
- **Lista de bloqueo de comandos:** Bloquea comandos arriesgados que obtienen contenido arbitrario de la web como `curl` y `wget` de forma predeterminada. Cuando se permite explícitamente, tenga en cuenta las [limitaciones del patrón de permisos](#)

Medidas de protección de privacidad

Hemos implementado varias medidas de protección para proteger sus datos, incluyendo:

- Períodos de retención limitados para información sensible (consulte el [Privacy Center](#) para obtener más información)
- Acceso restringido a datos de sesión del usuario
- Control del usuario sobre preferencias de entrenamiento de datos. Los usuarios de consumidor pueden cambiar su [configuración de privacidad](#) en cualquier momento.

Para obtener detalles completos, consulte nuestros [Términos de Servicio Comerciales](#) (para usuarios de Team, Enterprise y API) o [Términos de Consumidor](#) (para usuarios de Free, Pro y Max) y [Política de Privacidad](#).

Medidas de protección adicionales

- **Aprobación de solicitudes de red:** Las herramientas que realizan solicitudes de red requieren aprobación del usuario de forma predeterminada
- **Ventanas de contexto aisladas:** La obtención web utiliza una ventana de contexto separada para evitar inyectar solicitudes potencialmente maliciosas
- **Verificación de confianza:** Las primeras ejecuciones de base de código y los nuevos servidores MCP requieren verificación de confianza
 - Nota: La verificación de confianza está deshabilitada cuando se ejecuta de forma no interactiva con la bandera `-p`
- **Detección de inyección de comandos:** Los comandos bash sospechosos requieren aprobación manual incluso si fueron permitidos previamente
- **Coincidencia de cierre seguro:** Los comandos no coincidentes se establecen de forma predeterminada para requerir aprobación manual
- **Descripciones en lenguaje natural:** Los comandos bash complejos incluyen explicaciones para la comprensión del usuario
- **Almacenamiento seguro de credenciales:** Las claves API y tokens están encriptados. Consulte [Credential Management](#)

Warning:

Riesgo de seguridad de WebDAV en Windows: Cuando ejecute Claude Code en Windows, le recomendamos que no habilite WebDAV ni permita que Claude Code acceda a rutas como `\\*` que pueden contener subdirectorios de WebDAV. [WebDAV ha sido deprecado por Microsoft](#) debido a riesgos de seguridad. Habilitar WebDAV puede permitir que Claude Code desencadene solicitudes de red a hosts remotos, eludiendo el sistema de permisos.

Mejores prácticas para trabajar con contenido no confiable:

1. Revise los comandos sugeridos antes de aprobarlos
2. Evite canalizar contenido no confiable directamente a Claude
3. Verifique los cambios propuestos en archivos críticos
4. Utilice máquinas virtuales (VMs) para ejecutar scripts y realizar llamadas de herramientas, especialmente cuando interactúe con servicios web externos
5. Reporte comportamiento sospechoso con `/bug`

Warning:

Aunque estas protecciones reducen significativamente el riesgo, ningún sistema es completamente inmune a todos los ataques. Siempre mantenga buenas prácticas de seguridad cuando trabaje con cualquier herramienta de IA.

Seguridad de MCP

Claude Code permite a los usuarios configurar servidores del Protocolo de Contexto del Modelo (MCP). La lista de servidores MCP permitidos se configura en su código fuente, como parte de la configuración de Claude Code que los ingenieros verifican en el control de versiones.

Le recomendamos que escriba sus propios servidores MCP o utilice servidores MCP de proveedores en los que confíe. Puede configurar permisos de Claude Code para servidores MCP. Anthropic no gestiona ni audita ningún servidor MCP.

Seguridad del IDE

Consulte [Seguridad y privacidad de VS Code](#) para obtener más información sobre cómo ejecutar Claude Code en un IDE.

Seguridad de ejecución en la nube

Cuando utiliza [Claude Code en la web](#), hay controles de seguridad adicionales en su lugar:

- **Máquinas virtuales aisladas:** Cada sesión en la nube se ejecuta en una VM aislada gestionada por Anthropic
- **Controles de acceso a la red:** El acceso a la red está limitado de forma predeterminada y se puede configurar para deshabilitarse o permitir solo dominios específicos
- **Protección de credenciales:** La autenticación se maneja a través de un proxy seguro que utiliza una credencial con alcance dentro del sandbox, que luego se traduce a su token de autenticación de GitHub real
- **Restricciones de rama:** Las operaciones de inserción de Git están restringidas a la rama de trabajo actual
- **Registro de auditoría:** Todas las operaciones en entornos en la nube se registran para fines de cumplimiento y auditoría
- **Limpieza automática:** Los entornos en la nube se terminan automáticamente después de la finalización de la sesión

Para obtener más detalles sobre la ejecución en la nube, consulte [Claude Code en la web](#).

Las sesiones de [Remote Control](#) funcionan de manera diferente: la interfaz web se conecta a un proceso de Claude Code que se ejecuta en su máquina local. Toda la ejecución de código y el acceso a archivos permanecen locales, y los mismos datos que fluyen durante cualquier sesión local de Claude Code viajan a través de la API de Anthropic sobre TLS. No hay VMs en la nube ni sandboxing involucrados. La conexión utiliza múltiples credenciales de corta duración y alcance estrecho, cada una limitada a un propósito específico y expirando independientemente, para limitar el radio de explosión de cualquier credencial comprometida.

Mejores prácticas de seguridad

Trabajar con código sensible

- Revise todos los cambios sugeridos antes de aprobarlos
- Utilice configuración de permisos específica del proyecto para repositorios sensibles
- Considere utilizar [devcontainers](#) para aislamiento adicional
- Audite regularmente su configuración de permisos con `/permissions`

Seguridad del equipo

- Utilice [configuración gestionada](#) para aplicar estándares organizacionales
- Comparta configuraciones de permisos aprobadas a través del control de versiones
- Capacite a los miembros del equipo sobre mejores prácticas de seguridad
- Monitoree el uso de Claude Code a través de [métricas de OpenTelemetry](#)
- Audite o bloquee cambios de configuración durante sesiones con [hooks](#) [ConfigChange](#)

Reportar problemas de seguridad

Si descubre una vulnerabilidad de seguridad en Claude Code:

1. No la divulgue públicamente
2. Repórtela a través de nuestro [programa HackerOne](#)
3. Incluya pasos de reproducción detallados
4. Permita tiempo para que abordemos el problema antes de la divulgación pública

Recursos relacionados

- [Sandboxing](#) - Aislamiento del sistema de archivos y red para comandos bash
- [Permissions](#) - Configure permisos y controles de acceso
- [Monitoring usage](#) - Rastree y audite la actividad de Claude Code
- [Development containers](#) - Entornos seguros y aislados

- [Anthropic Trust Center](#) - Certificaciones de seguridad y cumplimiento

Sandboxing

Aprenda cómo la herramienta bash aislada de Claude Code proporciona aislamiento del sistema de archivos y la red para una ejecución de agentes más segura y autónoma.

Descripción general

Claude Code incluye sandboxing nativo para proporcionar un entorno más seguro para la ejecución de agentes mientras reduce la necesidad de solicitudes de permiso constantes. En lugar de pedir permiso para cada comando bash, el sandboxing crea límites definidos de antemano donde Claude Code puede trabajar más libremente con riesgo reducido.

La herramienta bash aislada utiliza primitivas a nivel del sistema operativo para aplicar tanto aislamiento del sistema de archivos como de la red.

Por qué el sandboxing es importante

La seguridad tradicional basada en permisos requiere aprobación constante del usuario para comandos bash. Si bien esto proporciona control, puede llevar a:

- **Fatiga de aprobación:** Hacer clic repetidamente en “aprobar” puede hacer que los usuarios presten menos atención a lo que están aprobando
- **Productividad reducida:** Las interrupciones constantes ralentizan los flujos de trabajo de desarrollo
- **Autonomía limitada:** Claude Code no puede trabajar de manera eficiente cuando espera aprobaciones

El sandboxing aborda estos desafíos mediante:

1. **Definir límites claros:** Especifique exactamente qué directorios y hosts de red puede acceder Claude Code
2. **Reducir solicitudes de permiso:** Los comandos seguros dentro del sandbox no requieren aprobación
3. **Mantener la seguridad:** Los intentos de acceder a recursos fuera del sandbox desencadenan notificaciones inmediatas
4. **Habilitar autonomía:** Claude Code puede ejecutarse de manera más independiente dentro de límites definidos

Warning:

El sandboxing efectivo requiere **tanto** aislamiento del sistema de archivos como de la red. Sin aislamiento de red, un agente comprometido podría exfiltrar archivos sensibles como claves SSH. Sin aislamiento del sistema de archivos, un agente comprometido podría instalar una puerta trasera en recursos del sistema para obtener acceso a la red. Al configurar el sandboxing, es importante asegurarse de que la configuración no cree omisiones en estos sistemas.

Cómo funciona

Aislamiento del sistema de archivos

La herramienta bash aislada restringe el acceso al sistema de archivos a directorios específicos:

- **Comportamiento de escritura predeterminado:** Acceso de lectura y escritura al directorio de trabajo actual y sus subdirectorios
- **Comportamiento de lectura predeterminado:** Acceso de lectura a toda la computadora, excepto ciertos directorios denegados
- **Acceso bloqueado:** No puede modificar archivos fuera del directorio de trabajo actual sin permiso explícito
- **Configurable:** Defina rutas permitidas y denegadas personalizadas a través de la configuración

Puede otorgar acceso de escritura a rutas adicionales usando

`sandbox.filesystem.allowWrite` en su configuración. Estas restricciones se aplican a nivel del sistema operativo (Seatbelt en macOS, bubblewrap en Linux), por lo que se aplican a todos los comandos de subproceso, incluidas herramientas como `kubectl`, `terraform` y `npm`, no solo a las herramientas de archivo de Claude.

Aislamiento de red

El acceso a la red se controla a través de un servidor proxy que se ejecuta fuera del sandbox:

- **Restricciones de dominio:** Solo se pueden acceder a dominios aprobados
- **Confirmación del usuario:** Las nuevas solicitudes de dominio desencadenan solicitudes de permiso (a menos que `allowManagedDomainsOnly` esté habilitado, que bloquea automáticamente dominios no permitidos)
- **Soporte de proxy personalizado:** Los usuarios avanzados pueden implementar reglas personalizadas en el tráfico saliente

- **Cobertura integral:** Las restricciones se aplican a todos los scripts, programas y subprocesos generados por comandos

Aplicación a nivel del sistema operativo

La herramienta bash aislada aprovecha las primitivas de seguridad del sistema operativo:

- **macOS:** Utiliza Seatbelt para la aplicación del sandbox
- **Linux:** Utiliza [bubblewrap](#) para el aislamiento
- **WSL2:** Utiliza bubblewrap, igual que Linux

WSL1 no es compatible porque bubblewrap requiere características del kernel solo disponibles en WSL2.

Estas restricciones a nivel del sistema operativo garantizan que todos los procesos secundarios generados por los comandos de Claude Code hereden los mismos límites de seguridad.

Primeros pasos

Requisitos previos

En **macOS**, el sandboxing funciona de inmediato usando el marco Seatbelt integrado.

En **Linux y WSL2**, instale primero los paquetes requeridos:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

Habilitar sandboxing

Puede habilitar el sandboxing ejecutando el comando `/sandbox` :

```
/sandbox
```

Esto abre un menú donde puede elegir entre modos de sandbox. Si faltan dependencias requeridas (como `bubblewrap` o `socat` en Linux), el menú muestra instrucciones de instalación para su plataforma.

Modos de sandbox

Claude Code ofrece dos modos de sandbox:

Modo de auto-permitir: Los comandos Bash intentarán ejecutarse dentro del sandbox y se permitirán automáticamente sin requerir permiso. Los comandos que no se pueden aislar (como aquellos que necesitan acceso a la red a hosts no permitidos) vuelven al flujo de permiso regular. Las reglas de solicitud/denegación explícitas que ha configurado siempre se respetan.

Modo de permisos regulares: Todos los comandos bash pasan por el flujo de permiso estándar, incluso cuando están aislados. Esto proporciona más control pero requiere más aprobaciones.

En ambos modos, el sandbox aplica las mismas restricciones de sistema de archivos y red. La diferencia es solo si los comandos aislados se aprueban automáticamente o requieren permiso explícito.

Info:

El modo de auto-permitir funciona independientemente de su configuración de modo de permiso. Incluso si no está en modo “aceptar ediciones”, los comandos bash aislados se ejecutarán automáticamente cuando el auto-permitir esté habilitado. Esto significa que los comandos bash que modifican archivos dentro de los límites del sandbox se ejecutarán sin solicitar, incluso cuando las herramientas de edición de archivos normalmente requerirían aprobación.

Configurar sandboxing

Personalice el comportamiento del sandbox a través de su archivo `settings.json`. Consulte [Configuración](#) para obtener la referencia de configuración completa.

Otorgar acceso de escritura de subprocesso a rutas específicas

De forma predeterminada, los comandos aislados solo pueden escribir en el directorio de trabajo actual. Si comandos de subprocesso como `kubectl`, `terraform` o `npm` necesitan escribir fuera del directorio del proyecto, use `sandbox.filesystem.allowWrite` para otorgar acceso a rutas específicas:

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

Estas rutas se aplican a nivel del sistema operativo, por lo que todos los comandos que se ejecutan dentro del sandbox, incluidos sus procesos secundarios, las respetan. Este es el enfoque recomendado cuando una herramienta necesita acceso de escritura a una ubicación específica, en lugar de excluir la herramienta del sandbox por completo con

`excludedCommands` .

Cuando `allowWrite` (o `denyWrite` / `denyRead`) se define en múltiples [ámbitos de configuración](#), los arrays se **fusionan**, lo que significa que las rutas de cada ámbito se combinan, no se reemplazan. Por ejemplo, si la configuración administrada permite escrituras en `//opt/company-tools` y un usuario agrega `~/.kube` en su configuración personal, ambas rutas se incluyen en la configuración final del sandbox. Esto significa que los usuarios y proyectos pueden extender la lista sin duplicar ni anular las rutas establecidas por ámbitos de mayor prioridad.

Los prefijos de ruta controlan cómo se resuelven las rutas:

Prefijo	Significado	Ejemplo
<code>//</code>	Ruta absoluta desde la raíz del sistema de archivos	<code>//tmp/build</code> se convierte en <code>/tmp/build</code>
<code>~/</code>	Relativo al directorio de inicio	<code>~/.kube</code> se convierte en <code>\$HOME/.kube</code>
<code>/</code>	Relativo al directorio del archivo de configuración	<code>/build</code> se convierte en <code>\$SETTINGS_DIR/build</code>
<code>./</code> o sin prefijo	Ruta relativa (resuelta por el tiempo de ejecución del sandbox)	<code>./output</code>

También puede denegar acceso de escritura o lectura usando

`sandbox.filesystem.denyWrite` y `sandbox.filesystem.denyRead` . Estos se fusionan con cualquier ruta de las reglas de permiso `Edit(...)` y `Read(...)` .

Tip:

No todos los comandos son compatibles con el sandboxing de inmediato. Algunas notas que pueden ayudarle a aprovechar al máximo el sandbox:

- Muchas herramientas CLI requieren acceder a ciertos hosts. A medida que utiliza estas herramientas, solicitarán permiso para acceder a ciertos hosts. Otorgar permiso les permitirá acceder a estos hosts ahora y en el futuro, permitiéndoles ejecutarse de manera segura dentro del sandbox.
- `watchman` es incompatible con la ejecución en el sandbox. Si está ejecutando `jest`, considere usar `jest --no-watchman`
- `docker` es incompatible con la ejecución en el sandbox. Considere especificar `docker` en `excludedCommands` para forzarlo a ejecutarse fuera del sandbox.

Note:

Claude Code incluye un mecanismo de escape intencional que permite que los comandos se ejecuten fuera del sandbox cuando sea necesario. Cuando un comando falla debido a restricciones del sandbox (como problemas de conectividad de red o herramientas incompatibles), se solicita a Claude que analice la falla y puede reintentar el comando con el parámetro `dangerouslyDisableSandbox`. Los comandos que utilizan este parámetro pasan por el flujo de permisos normal de Claude Code que requiere permiso del usuario para ejecutarse. Esto permite que Claude Code maneje casos extremos donde ciertas herramientas u operaciones de red no pueden funcionar dentro de las restricciones del sandbox.

Puede deshabilitar este mecanismo de escape configurando `"allowUnsandboxedCommands": false` en su [configuración de sandbox](#). Cuando está deshabilitado, el parámetro `dangerouslyDisableSandbox` se ignora completamente y todos los comandos deben ejecutarse aislados o estar explícitamente listados en `excludedCommands`.

Beneficios de seguridad

Protección contra inyección de solicitud

Incluso si un atacante manipula con éxito el comportamiento de Claude Code a través de inyección de solicitud, el sandbox garantiza que su sistema permanezca seguro:

Protección del sistema de archivos:

- No puede modificar archivos de configuración críticos como `~/.bashrc`
- No puede modificar archivos a nivel del sistema en `/bin/`
- No puede leer archivos que se deniegan en su [configuración de permisos de Claude](#)

Protección de red:

- No puede exfiltrar datos a servidores controlados por atacantes
- No puede descargar scripts maliciosos de dominios no autorizados
- No puede realizar llamadas API inesperadas a servicios no aprobados
- No puede contactar a ningún dominio que no esté explícitamente permitido

Monitoreo y control:

- Todos los intentos de acceso fuera del sandbox se bloquean a nivel del sistema operativo
- Recibe notificaciones inmediatas cuando se prueban los límites
- Puede elegir denegar, permitir una vez o actualizar permanentemente su configuración

Superficie de ataque reducida

El sandboxing limita el daño potencial de:

- **Dependencias maliciosas:** Paquetes NPM u otras dependencias con código dañino
- **Scripts comprometidos:** Scripts de compilación o herramientas con vulnerabilidades de seguridad
- **Ingeniería social:** Ataques que engañan a los usuarios para que ejecuten comandos peligrosos
- **Inyección de solicitud:** Ataques que engañan a Claude para que ejecute comandos peligrosos

Operación transparente

Cuando Claude Code intenta acceder a recursos de red fuera del sandbox:

1. La operación se bloquea a nivel del sistema operativo
2. Recibe una notificación inmediata
3. Puede elegir:
 - Denegar la solicitud
 - Permitirla una vez
 - Actualizar su configuración de sandbox para permitirla permanentemente

Limitaciones de seguridad

- Limitaciones de sandboxing de red: El sistema de filtrado de red funciona restringiendo los dominios a los que se permite que se conecten los procesos. De lo contrario, no

inspecciona el tráfico que pasa a través del proxy y los usuarios son responsables de asegurarse de que solo permitan dominios confiables en su política.

Warning:

Los usuarios deben ser conscientes de los riesgos potenciales que conlleva permitir dominios amplios como github.com que pueden permitir la exfiltración de datos. Además, en algunos casos puede ser posible eludir el filtrado de red a través de [domain fronting](#).

- Escalada de privilegios a través de sockets Unix: La configuración `allowUnixSockets` puede otorgar inadvertidamente acceso a servicios del sistema poderosos que podrían llevar a omisiones del sandbox. Por ejemplo, si se usa para permitir acceso a `/var/run/docker.sock`, esto efectivamente otorgaría acceso al sistema host explotando el socket de docker. Se anima a los usuarios a considerar cuidadosamente cualquier socket unix que permitan a través del sandbox.
- Escalada de permisos del sistema de archivos: Los permisos de escritura del sistema de archivos demasiado amplios pueden permitir ataques de escalada de privilegios. Permitir escrituras en directorios que contienen ejecutables en `$PATH`, directorios de configuración del sistema o archivos de configuración de shell del usuario (`.bashrc`, `.zshrc`) puede llevar a la ejecución de código en diferentes contextos de seguridad cuando otros usuarios o procesos del sistema acceden a estos archivos.
- Fortaleza del sandbox de Linux: La implementación de Linux proporciona un fuerte aislamiento del sistema de archivos y la red pero incluye un modo `enableWeakerNestedSandbox` que le permite funcionar dentro de entornos Docker sin espacios de nombres privilegiados. Esta opción debilita considerablemente la seguridad y solo debe usarse en casos donde se aplica aislamiento adicional de otra manera.

Cómo se relaciona el sandboxing con los permisos

El sandboxing y los [permisos](#) son capas de seguridad complementarias que funcionan juntas:

- **Permisos** controlan qué herramientas puede usar Claude Code y se evalúan antes de que se ejecute cualquier herramienta. Se aplican a todas las herramientas: Bash, Read, Edit, WebFetch, MCP y otras.
- **Sandboxing** proporciona aplicación a nivel del sistema operativo que restringe lo que los comandos Bash pueden acceder a nivel del sistema de archivos y la red. Se aplica solo a comandos Bash y sus procesos secundarios.

Las restricciones del sistema de archivos y la red se configuran tanto a través de la configuración del sandbox como de las reglas de permiso:

- Use `sandbox.filesystem.allowWrite` para otorgar acceso de escritura de subprocesso a rutas fuera del directorio de trabajo
- Use `sandbox.filesystem.denyWrite` y `sandbox.filesystem.denyRead` para bloquear el acceso de subprocesso a rutas específicas
- Use reglas de denegación `Read` y `Edit` para bloquear el acceso a archivos o directorios específicos
- Use reglas de permitir/denegar `WebFetch` para controlar el acceso al dominio
- Use `allowedDomains` del sandbox para controlar qué dominios pueden alcanzar los comandos Bash

Las rutas de ambas configuraciones `sandbox.filesystem` y reglas de permiso se fusionan en la configuración final del sandbox.

Este [repositorio](#) incluye configuraciones de configuración de inicio para escenarios de implementación comunes, incluidos ejemplos específicos del sandbox. Úselos como puntos de partida y ajústelos según sus necesidades.

Uso avanzado

Configuración de proxy personalizado

Para organizaciones que requieren seguridad de red avanzada, puede implementar un proxy personalizado para:

- Descifrar e inspeccionar tráfico HTTPS
- Aplicar reglas de filtrado personalizadas
- Registrar todas las solicitudes de red
- Integrar con infraestructura de seguridad existente

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

Integración con herramientas de seguridad existentes

La herramienta bash aislada funciona junto con:

- **Reglas de permiso:** Combinar con [configuración de permisos](#) para defensa en profundidad
- **Contenedores de desarrollo:** Usar con [devcontainers](#) para aislamiento adicional
- **Políticas empresariales:** Aplicar configuraciones de sandbox a través de [configuración administrada](#)

Mejores prácticas

1. **Comience restrictivo:** Comience con permisos mínimos y expanda según sea necesario
2. **Monitoree registros:** Revise los intentos de violación del sandbox para entender las necesidades de Claude Code
3. **Use configuraciones específicas del entorno:** Diferentes reglas de sandbox para contextos de desarrollo versus producción
4. **Combine con permisos:** Use sandboxing junto con políticas IAM para seguridad integral
5. **Pruebe configuraciones:** Verifique que su configuración de sandbox no bloquee flujos de trabajo legítimos

Código abierto

El tiempo de ejecución del sandbox está disponible como un paquete npm de código abierto para usar en sus propios proyectos de agentes. Esto permite que la comunidad más amplia de agentes de IA construya sistemas autónomos más seguros y protegidos. Esto también se puede usar para aislar otros programas que desee ejecutar. Por ejemplo, para aislar un servidor MCP podría ejecutar:

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

Para detalles de implementación y código fuente, visite el [repositorio de GitHub](#).

Limitaciones

- **Sobrecarga de rendimiento:** Mínima, pero algunas operaciones del sistema de archivos pueden ser ligeramente más lentas
- **Compatibilidad:** Algunas herramientas que requieren patrones de acceso específicos del sistema pueden necesitar ajustes de configuración, o incluso pueden necesitar ejecutarse fuera del sandbox
- **Soporte de plataforma:** Admite macOS, Linux y WSL2. WSL1 no es compatible. Se planea soporte nativo de Windows.

Ver también

- [Seguridad](#) - Características de seguridad integral y mejores prácticas
- [Permisos](#) - Configuración de permisos y control de acceso
- [Configuración](#) - Referencia de configuración completa
- [Referencia de CLI](#) - Opciones de línea de comandos

checkpoint

Realiza un seguimiento automático y revierte los cambios de Claude para recuperarse rápidamente de cambios no deseados.

Claude Code realiza un seguimiento automático de los cambios de archivos de Claude mientras trabaja, permitiéndole deshacer rápidamente cambios y revertir a estados anteriores si algo se sale del camino.

Cómo funciona checkpoint

Mientras trabaja con Claude, checkpoint captura automáticamente el estado de su código antes de cada edición. Esta red de seguridad le permite realizar tareas ambiciosas y a gran escala sabiendo que siempre puede volver a un estado de código anterior.

Seguimiento automático

Claude Code realiza un seguimiento de todos los cambios realizados por sus herramientas de edición de archivos:

- Cada mensaje del usuario crea un nuevo checkpoint
- Los checkpoints persisten entre sesiones, por lo que puede acceder a ellos en conversaciones reanudadas
- Se limpian automáticamente junto con las sesiones después de 30 días (configurable)

Revertir cambios

Presione **Esc** dos veces (**Esc** + **Esc**) o use el comando **/rewind** para abrir el menú de rewind. Puede elegir restaurar:

- **Solo conversación:** Revertir a un mensaje del usuario manteniendo los cambios de código
- **Solo código:** Revertir cambios de archivo manteniendo la conversación
- **Tanto código como conversación:** Restaurar ambos a un punto anterior en la sesión

Casos de uso comunes

Los checkpoints son particularmente útiles cuando:

- **Explorar alternativas:** Pruebe diferentes enfoques de implementación sin perder su punto de partida
- **Recuperarse de errores:** Deshaga rápidamente cambios que introdujeron errores o rompieron funcionalidad
- **Iterar en características:** Experimente con variaciones sabiendo que puede revertir a estados que funcionan

Limitaciones

Los cambios de comando Bash no se rastrean

checkpoint no rastrea archivos modificados por comandos bash. Por ejemplo, si Claude Code ejecuta:

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

Estas modificaciones de archivo no se pueden deshacer a través de rewind. Solo se rastrean las ediciones de archivo directo realizadas a través de las herramientas de edición de archivos de Claude.

Los cambios externos no se rastrean

checkpoint solo rastrea archivos que han sido editados dentro de la sesión actual. Los cambios manuales que realiza en archivos fuera de Claude Code y las ediciones de otras sesiones concurrentes normalmente no se capturan, a menos que modifiquen los mismos archivos que la sesión actual.

No es un reemplazo para control de versiones

Los checkpoints están diseñados para recuperación rápida a nivel de sesión. Para historial de versiones permanente y colaboración:

- Continúe usando control de versiones (ej. Git) para commits, ramas e historial a largo plazo
- Los checkpoints complementan pero no reemplazan el control de versiones adecuado
- Piense en los checkpoints como “deshacer local” y Git como “historial permanente”

Ver también

- [Modo interactivo](#) - Atajos de teclado y controles de sesión
- [Comandos integrados](#) - Acceso a checkpoints usando `/rewind`
- [Referencia CLI](#) - Opciones de línea de comandos

Uso de datos

Conozca las políticas de uso de datos de Anthropic para Claude

Políticas de datos

Política de entrenamiento de datos

Usuarios de consumidor (planes Free, Pro y Max): Le damos la opción de permitir que sus datos se utilicen para mejorar futuros modelos de Claude. Entrenaremos nuevos modelos utilizando datos de cuentas Free, Pro y Max cuando esta configuración esté activada (incluso cuando utiliza Claude Code desde estas cuentas).

Usuarios comerciales: (planes Team y Enterprise, API, plataformas de terceros y Claude Gov) mantienen políticas existentes: Anthropic no entrena modelos generativos utilizando código o indicaciones enviados a Claude Code bajo términos comerciales, a menos que el cliente haya elegido proporcionarnos sus datos para mejorar el modelo (por ejemplo, el [Development Partner Program](#)).

Development Partner Program

Si opta explícitamente por métodos para proporcionarnos materiales para entrenar, como a través del [Development Partner Program](#), podemos utilizar esos materiales proporcionados para entrenar nuestros modelos. Un administrador de la organización puede optar explícitamente por el Development Partner Program para su organización. Tenga en cuenta que este programa está disponible solo para API de primera parte de Anthropic, y no para usuarios de Bedrock o Vertex.

Comentarios usando el comando `/bug`

Si elige enviarnos comentarios sobre Claude Code usando el comando `/bug`, podemos utilizar sus comentarios para mejorar nuestros productos y servicios. Las transcripciones compartidas a través de `/bug` se retienen durante 5 años.

Encuestas de calidad de sesión

Cuando ve el mensaje “¿Cómo está funcionando Claude en esta sesión?” en Claude Code, responder a esta encuesta (incluyendo seleccionar “Descartar”), solo se registra su calificación numérica (1, 2, 3 o descartar). No recopilamos ni almacenamos transcripciones de conversación, entradas, salidas u otros datos de sesión como parte de esta encuesta. A

diferencia de los comentarios de pulgar hacia arriba/abajo o los informes `/bug`, esta encuesta de calidad de sesión es una métrica simple de satisfacción del producto. Sus respuestas a esta encuesta no afectan sus preferencias de entrenamiento de datos y no se pueden utilizar para entrenar nuestros modelos de IA.

Para desactivar estas encuestas, establezca `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1`. La encuesta también se desactiva automáticamente cuando se utilizan proveedores de terceros (Bedrock, Vertex, Foundry) o cuando la telemetría está desactivada.

Retención de datos

Anthropic retiene datos de Claude Code según su tipo de cuenta y preferencias.

Usuarios de consumidor (planes Free, Pro y Max):

- Usuarios que permiten el uso de datos para mejorar el modelo: período de retención de 5 años para apoyar el desarrollo del modelo y mejoras de seguridad
- Usuarios que no permiten el uso de datos para mejorar el modelo: período de retención de 30 días
- La configuración de privacidad se puede cambiar en cualquier momento en claude.ai/settings/data-privacy-controls.

Usuarios comerciales (Team, Enterprise y API):

- Estándar: período de retención de 30 días
- [Zero data retention](#): disponible para Claude Code en Claude for Enterprise. ZDR se habilita por organización; cada nueva organización debe tener ZDR habilitado por separado por su equipo de cuenta
- Almacenamiento en caché local: los clientes de Claude Code pueden almacenar sesiones localmente hasta 30 días para permitir la reanudación de sesiones (configurable)

Puede eliminar sesiones individuales de Claude Code en la web en cualquier momento. Eliminar una sesión elimina permanentemente los datos de eventos de la sesión. Para obtener instrucciones sobre cómo eliminar sesiones, consulte [Managing sessions](#).

Obtenga más información sobre las prácticas de retención de datos en nuestro [Privacy Center](#).

Para obtener todos los detalles, consulte nuestros [Commercial Terms of Service](#) (para usuarios de Team, Enterprise y API) o [Consumer Terms](#) (para usuarios de Free, Pro y Max) y [Privacy Policy](#).

Acceso a datos

Para todos los usuarios de primera parte, puede obtener más información sobre qué datos se registran para [Claude Code local](#) y [Claude Code remoto](#). Las sesiones de [Remote Control](#) siguen el flujo de datos local ya que toda la ejecución ocurre en su máquina. Tenga en cuenta que para Claude Code remoto, Claude accede al repositorio donde inicia su sesión de Claude Code. Claude no accede a repositorios que ha conectado pero en los que no ha iniciado una sesión.

Local Claude Code: Flujo de datos y dependencias

El diagrama a continuación muestra cómo Claude Code se conecta a servicios externos durante la instalación y operación normal. Las líneas sólidas indican conexiones requeridas, mientras que las líneas punteadas representan flujos de datos opcionales o iniciados por el usuario.

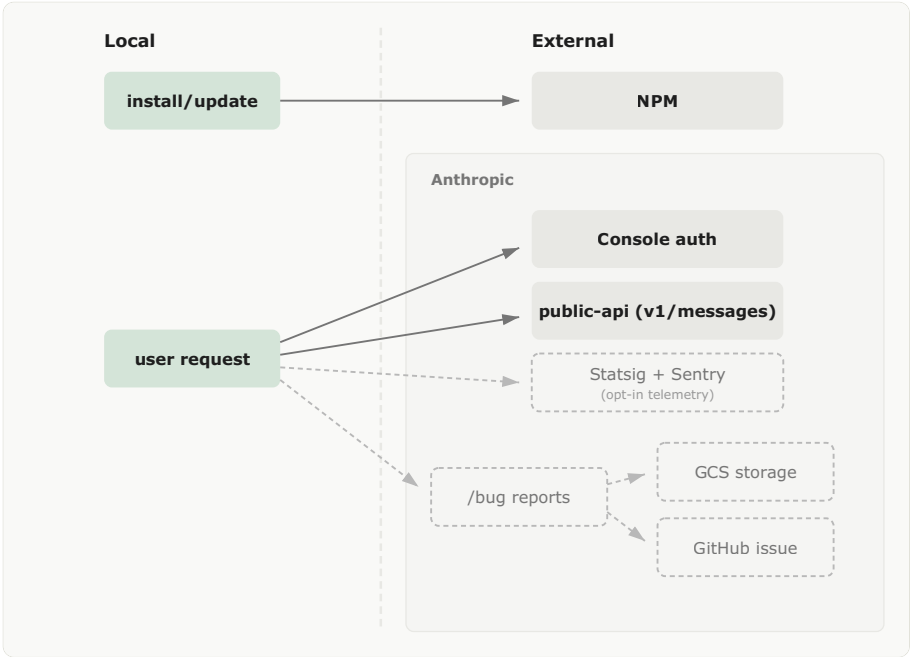


Diagrama que muestra las conexiones externas de Claude Code: instalar/actualizar se conecta a NPM, y las solicitudes del usuario se conectan a servicios de Anthropic incluyendo autenticación de consola, API pública, y opcionalmente Statsig, Sentry e informes de errores

Claude Code se instala desde [NPM](#). Claude Code se ejecuta localmente. Para interactuar con el LLM, Claude Code envía datos a través de la red. Estos datos incluyen todos los indicadores del usuario y salidas del modelo. Los datos se cifran en tránsito a través de TLS y no se cifran en reposo. Claude Code es compatible con la mayoría de VPN y proxies LLM populares.

Claude Code se construye sobre las API de Anthropic. Para obtener detalles sobre los controles de seguridad de nuestra API, incluidos nuestros procedimientos de registro de API, consulte los artefactos de cumplimiento ofrecidos en el [Anthropic Trust Center](#).

Cloud execution: Flujo de datos y dependencias

Cuando se utiliza [Claude Code en la web](#), las sesiones se ejecutan en máquinas virtuales administradas por Anthropic en lugar de localmente. En entornos en la nube:

- **Almacenamiento de código y datos:** Su repositorio se clona en una VM aislada. El código y los datos de sesión están sujetos a las políticas de retención y uso para su tipo de cuenta (consulte la sección Retención de datos anterior)
- **Credenciales:** La autenticación de GitHub se maneja a través de un proxy seguro; sus credenciales de GitHub nunca ingresan al sandbox
- **Tráfico de red:** Todo el tráfico saliente pasa a través de un proxy de seguridad para registro de auditoría y prevención de abuso
- **Datos de sesión:** Los indicadores, cambios de código y salidas siguen las mismas políticas de datos que el uso local de Claude Code

Para obtener detalles de seguridad sobre la ejecución en la nube, consulte [Security](#).

Servicios de telemetría

Claude Code se conecta desde las máquinas de los usuarios al servicio Statsig para registrar métricas operativas como latencia, confiabilidad y patrones de uso. Este registro no incluye ningún código o ruta de archivo. Los datos se cifran en tránsito usando TLS y en reposo usando cifrado AES de 256 bits. Lea más en la [documentación de seguridad de Statsig](#). Para optar por no participar en la telemetría de Statsig, establezca la variable de entorno `DISABLE_TELEMETRY`.

Claude Code se conecta desde las máquinas de los usuarios a Sentry para el registro de errores operativos. Los datos se cifran en tránsito usando TLS y en reposo usando cifrado AES de 256 bits. Lea más en la [documentación de seguridad de Sentry](#). Para optar por no participar en el registro de errores, establezca la variable de entorno `DISABLE_ERROR_REPORTING`.

Cuando los usuarios ejecutan el comando `/bug` , se envía una copia de su historial de conversación completo incluyendo código a Anthropic. Los datos se cifran en tránsito y en reposo. Opcionalmente, se crea un problema de Github en nuestro repositorio público. Para optar por no participar en los informes de errores, establezca la variable de entorno `DISABLE_BUG_COMMAND` .

Comportamientos predeterminados por proveedor de API

De forma predeterminada, deshabilitamos todo el tráfico no esencial (incluyendo informes de errores, telemetría, funcionalidad de informes de errores y encuestas de calidad de sesión) cuando se utiliza Bedrock, Vertex o Foundry. También puede optar por no participar en todos estos a la vez estableciendo la variable de entorno

`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` . Aquí están los comportamientos predeterminados completos:

Servicio	Claude API	Vertex API	Bedrock API	Foundry API
Statsig (Métricas)	Activado de forma predeterminada. <code>DISABLE_TELEMETRY=1</code> para activar.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_VERTEX</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_BEDROCK</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_FOUNDRY</code> debe ser 1.
Sentry (Errores)	Activado de forma predeterminada. <code>DISABLE_ERROR_REPORTING=1</code> para desactivar.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_VERTEX</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_BEDROCK</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_FOUNDRY</code> debe ser 1.
Claude API (informes / bug)	Activado de forma predeterminada. <code>DISABLE_BUG_COMMAND=1</code> para desactivar.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_VERTEX</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_BEDROCK</code> debe ser 1.	Desactivado de forma predeterminada. <code>CLAUDE_CODE_USE_FOUNDRY</code> debe ser 1.

Servi- cio	Claude API	Vertex API	Bedrock API	Foundry API
En- cuestas de cali- dad de sesión	Activado de forma pre- determinada. CLAUDE_CODE_DISABLE _FEEDBACK_SURVEY=1 para desactivar.	Desactivado de forma predeter- minada. CLAUDE_CODE_U SE_VERTEX debe ser 1.	Desactivado de forma predeter- minada. CLAUDE_CODE_U SE_BEDROCK debe ser 1.	Desactivado de forma predeter- minada. CLAUDE_CODE_U SE_FOUNDRY debe ser 1.

Todas las variables de entorno se pueden verificar en settings.json (leer más).

Legal y cumplimiento

Acuerdos legales, certificaciones de cumplimiento e información de seguridad para Claude Code.

Acuerdos legales

Licencia

Su uso de Claude Code está sujeto a:

- [Términos Comerciales](#) - para usuarios de Team, Enterprise y Claude API
- [Términos de Consumidor](#) - para usuarios de Free, Pro y Max

Acuerdos comerciales

Ya sea que esté utilizando la Claude API directamente (1P) o accediendo a través de AWS Bedrock o Google Vertex (3P), su acuerdo comercial existente se aplicará al uso de Claude Code, a menos que hayamos acordado mutuamente lo contrario.

Cumplimiento

Cumplimiento de atención médica (BAA)

Si un cliente tiene un Acuerdo de Asociado de Negocios (BAA) con nosotros y desea utilizar Claude Code, el BAA se extenderá automáticamente para cubrir Claude Code si el cliente ha ejecutado un BAA y tiene activada la Retención Cero de Datos (ZDR). El BAA será aplicable al tráfico de API de ese cliente que fluye a través de Claude Code.

Seguridad y confianza

Confianza y seguridad

Puede encontrar más información en el [Centro de Confianza de Anthropic](#) y [Hub de Transparencia](#).

Reporte de vulnerabilidades de seguridad

Anthropic gestiona nuestro programa de seguridad a través de HackerOne. [Utilice este formulario para reportar vulnerabilidades](#).

© Anthropic PBC. Todos los derechos reservados. El uso está sujeto a los Términos de Servicio de Anthropic aplicables.

Retención cero de datos

Obtenga información sobre la Retención Cero de Datos (ZDR) para Claude Code en Claude for Enterprise, incluido el alcance, las características deshabilitadas y cómo solicitar la habilitación.

La Retención Cero de Datos (ZDR) está disponible para Claude Code cuando se utiliza a través de Claude for Enterprise. Cuando ZDR está habilitado, los prompts y las respuestas del modelo generadas durante las sesiones de Claude Code se procesan en tiempo real y no se almacenan por Anthropic después de que se devuelve la respuesta, excepto cuando es necesario para cumplir con la ley o combatir el uso indebido.

ZDR en Claude for Enterprise proporciona a los clientes empresariales la capacidad de usar Claude Code con retención cero de datos y acceso a capacidades administrativas:

- Controles de costos por usuario
- Panel de [Analytics](#)
- [Configuración administrada por servidor](#)
- Registros de auditoría

ZDR para Claude Code en Claude for Enterprise se aplica solo a la plataforma directa de Anthropic. Para implementaciones de Claude en AWS Bedrock, Google Vertex AI o Microsoft Foundry, consulte las políticas de retención de datos de esas plataformas.

Alcance de ZDR

ZDR cubre la inferencia de Claude Code en Claude for Enterprise.

Warning:

ZDR se habilita por organización. Cada nueva organización requiere que ZDR sea habilitado por separado por su equipo de cuenta de Anthropic. ZDR no se aplica automáticamente a las nuevas organizaciones creadas bajo la misma cuenta. Póngase en contacto con su equipo de cuenta para habilitar ZDR para cualquier nueva organización.

Qué cubre ZDR

ZDR cubre las llamadas de inferencia del modelo realizadas a través de Claude Code en Claude for Enterprise. Cuando utiliza Claude Code en su terminal, los prompts que envía y las respuestas que genera Claude no se retienen por Anthropic. Esto se aplica independientemente de qué modelo de Claude se utilice.

Qué no cubre ZDR

ZDR no se extiende a lo siguiente, incluso para organizaciones con ZDR habilitado. Estas características siguen [políticas estándar de retención de datos](#):

Ca- rac- te- rísti- ca	Detalles
Chat en clau- de.ai	Las conversaciones de chat a través de la interfaz web de Claude for Enterprise no están cubiertas por ZDR.
Co- work	Las sesiones de Cowork no están cubiertas por ZDR.
Clau- de Code Anal- ytics	No almacena prompts o respuestas del modelo, pero recopila metadatos de productividad como correos electrónicos de cuenta y estadísticas de uso. Las métricas de contribución no están disponibles para organizaciones ZDR; el panel de analytics muestra solo métricas de uso.
Ges- tión de usua- rios y asien- tos	Los datos administrativos como correos electrónicos de cuenta y asignaciones de asientos se retienen bajo políticas estándar.

Característica	Detalles
Integraciones de terceros	Los datos procesados por herramientas de terceros, MCP servers u otras integraciones externas no están cubiertos por ZDR. Revise las prácticas de manejo de datos de esos servicios de forma independiente.

Características deshabilitadas bajo ZDR

Cuando ZDR está habilitado para una organización de Claude Code en Claude for Enterprise, ciertas características que requieren almacenar prompts o completaciones se deshabilitan automáticamente a nivel de backend:

Característica	Razón
Claude Code en la Web	Requiere almacenamiento del lado del servidor del historial de conversaciones.
Sesiones remotas desde la aplicación Desktop	Requiere datos de sesión persistentes que incluyen prompts y completaciones.
Envío de comentarios (/feedback)	Enviar comentarios envía datos de conversación a Anthropic.

Estas características se bloquean en el backend independientemente de la visualización del lado del cliente. Si ve una característica deshabilitada en la terminal de Claude Code durante el inicio, intentar usarla devuelve un error indicando que las políticas de la organización no permiten esa acción.

Las características futuras también pueden deshabilitarse si requieren almacenar prompts o completaciones.

Retención de datos para violaciones de políticas

Incluso con ZDR habilitado, Anthropic puede retener datos cuando sea requerido por ley o para abordar violaciones de la Política de Uso. Si una sesión se marca por una violación de política, Anthropic puede retener las entradas y salidas asociadas hasta 2 años, consistente con la política estándar de ZDR de Anthropic.

Solicitar ZDR

Para solicitar ZDR para Claude Code en Claude for Enterprise, póngase en contacto con su equipo de cuenta de Anthropic. Su equipo de cuenta presentará la solicitud internamente, y Anthropic revisará y habilitará ZDR en su organización después de confirmar la elegibilidad. Todas las acciones de habilitación se registran en auditoría.

Si actualmente está utilizando ZDR para Claude Code a través de claves API de pago por uso, puede hacer la transición a Claude for Enterprise para obtener acceso a características administrativas mientras mantiene ZDR para Claude Code. Póngase en contacto con su equipo de cuenta para coordinar la migración.

Part 10: Enterprise & Monitoring

Rastrear el uso del equipo con análisis

Ver métricas de uso de Claude Code, rastrear la adopción y medir la velocidad de ingeniería en el panel de análisis.

Claude Code proporciona paneles de análisis para ayudar a las organizaciones a comprender los patrones de uso de desarrolladores, rastrear métricas de contribución y medir cómo Claude Code impacta la velocidad de ingeniería. Acceda al panel para su plan:

Plan	URL del panel	Incluye	Más información
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	Métricas de uso, métricas de contribución con integración de GitHub, tabla de clasificación, exportación de datos	Detalles
API (Claude Console)	platform.claude.com/claude-code	Métricas de uso, seguimiento de gastos, información del equipo	Detalles

Acceder a análisis para Teams y Enterprise

Navegue a claude.ai/analytics/claude-code. Los administradores y propietarios pueden ver el panel.

El panel de Teams y Enterprise incluye:

- **Métricas de uso:** líneas de código aceptadas, tasa de aceptación de sugerencias, usuarios activos diarios y sesiones
- **Métricas de contribución:** PRs y líneas de código enviadas con asistencia de Claude Code, con [integración de GitHub](#)
- **Tabla de clasificación:** principales contribuyentes clasificados por uso de Claude Code

- **Exportación de datos:** descargar datos de contribución como CSV para informes personalizados

Habilitar métricas de contribución

Note:

Las métricas de contribución están en versión beta pública y disponibles en los planes Claude for Teams y Claude for Enterprise. Estas métricas solo cubren usuarios dentro de su organización de claude.ai. El uso a través de la API de Claude Console o integraciones de terceros no se incluye.

Los datos de uso y adopción están disponibles para todas las cuentas de Claude for Teams y Claude for Enterprise. Las métricas de contribución requieren configuración adicional para conectar su organización de GitHub.

Necesita el rol de propietario para configurar los ajustes de análisis. Un administrador de GitHub debe instalar la aplicación de GitHub.

Warning:

Las métricas de contribución no están disponibles para organizaciones con [Retención de datos cero](#) habilitada. El panel de análisis mostrará solo métricas de uso.

Step 1: Instalar la aplicación de GitHub

Un administrador de GitHub instala la aplicación Claude GitHub en la cuenta de GitHub de su organización en github.com/apps/claude.

Step 2: Habilitar análisis de Claude Code

Un propietario de Claude navega a claude.ai/admin-settings/claude-code y habilita la función de análisis de Claude Code.

Step 3: Habilitar análisis de GitHub

En la misma página, habilite el botón de alternancia “GitHub analytics”.

Step 4: Autenticarse con GitHub

Complete el flujo de autenticación de GitHub y seleccione qué organizaciones de GitHub incluir en el análisis.

Los datos generalmente aparecen dentro de 24 horas después de habilitar, con actualizaciones diarias. Si no aparecen datos, puede ver uno de estos mensajes:

- **“GitHub app required”**: instale la aplicación de GitHub para ver métricas de contribución
- **“Data processing in progress”**: vuelva a verificar en unos días y confirme que la aplicación de GitHub está instalada si los datos no aparecen

Las métricas de contribución admiten GitHub Cloud y GitHub Enterprise Server.

Revisar métricas de resumen

Note:

Estas métricas son deliberadamente conservadoras y representan una subestimación del impacto real de Claude Code. Solo se cuentan las líneas y PRs donde hay alta confianza en la participación de Claude Code.

El panel muestra estas métricas de resumen en la parte superior:

- **PRs with CC**: recuento total de solicitudes de extracción fusionadas que contienen al menos una línea de código escrita con Claude Code
- **Lines of code with CC**: líneas totales de código en todos los PRs fusionados que fueron escritas con asistencia de Claude Code. Solo se cuentan las “líneas efectivas”: líneas con más de 3 caracteres después de la normalización, excluyendo líneas vacías y líneas con solo corchetes o puntuación trivial.
- **PRs with Claude Code (%)**: porcentaje de todos los PRs fusionados que contienen código asistido por Claude Code
- **Suggestion accept rate**: porcentaje de veces que los usuarios aceptan las sugerencias de edición de código de Claude Code, incluido el uso de herramientas Edit, Write y NotebookEdit
- **Lines of code accepted**: líneas totales de código escritas por Claude Code que los usuarios han aceptado en sus sesiones. Esto excluye sugerencias rechazadas y no rastrea eliminaciones posteriores.

Explorar los gráficos

El panel incluye varios gráficos para visualizar tendencias a lo largo del tiempo.

Rastrear adopción

El gráfico de Adopción muestra tendencias de uso diario:

- **users:** usuarios activos diarios
- **sessions:** número de sesiones activas de Claude Code por día

Medir PRs por usuario

Este gráfico muestra la actividad de desarrolladores individuales a lo largo del tiempo:

- **PRs per user:** número total de PRs fusionados por día dividido por usuarios activos diarios
- **users:** usuarios activos diarios

Utilice esto para comprender cómo cambia la productividad individual a medida que aumenta la adopción de Claude Code.

Ver desglose de solicitudes de extracción

El gráfico de Pull requests muestra un desglose diario de PRs fusionados:

- **PRs with CC:** solicitudes de extracción que contienen código asistido por Claude Code
- **PRs without CC:** solicitudes de extracción sin código asistido por Claude Code

Cambie a la vista **Lines of code** para ver el mismo desglose por líneas de código en lugar de recuento de PR.

Encontrar principales contribuyentes

La Tabla de clasificación muestra los 10 principales usuarios clasificados por volumen de contribución. Alterne entre:

- **Pull requests:** muestra PRs con Claude Code vs Todos los PRs para cada usuario
- **Lines of code:** muestra líneas con Claude Code vs Todas las líneas para cada usuario

Haga clic en **Export all users** para descargar datos de contribución completos para todos los usuarios como archivo CSV. La exportación incluye todos los usuarios, no solo los 10 principales mostrados.

Atribución de PR

Cuando las métricas de contribución están habilitadas, Claude Code analiza las solicitudes de extracción fusionadas para determinar qué código fue escrito con asistencia de Claude Code. Esto se hace haciendo coincidir la actividad de sesión de Claude Code con el código en cada PR.

Criterios de etiquetado

Los PRs se etiquetan como “with Claude Code” si contienen al menos una línea de código escrita durante una sesión de Claude Code. El sistema utiliza coincidencia conservadora: solo el código donde hay alta confianza en la participación de Claude Code se cuenta como asistido.

Proceso de atribución

Cuando se fusiona una solicitud de extracción:

1. Se extraen las líneas agregadas del diff de PR
2. Se identifican las sesiones de Claude Code que editaron archivos coincidentes dentro de una ventana de tiempo
3. Las líneas de PR se comparan con la salida de Claude Code utilizando múltiples estrategias
4. Se calculan métricas para líneas asistidas por IA y líneas totales

Antes de la comparación, las líneas se normalizan: se recorta el espacio en blanco, se contraen múltiples espacios, se estandarizan las comillas y el texto se convierte a minúsculas.

Las solicitudes de extracción fusionadas que contienen líneas asistidas por Claude Code se etiquetan como `claude-code-assisted` en GitHub.

Ventana de tiempo

Se consideran sesiones de 21 días antes a 2 días después de la fecha de fusión de PR para la coincidencia de atribución.

Archivos excluidos

Ciertos archivos se excluyen automáticamente del análisis porque se generan automáticamente:

- Archivos de bloqueo: `package-lock.json`, `yarn.lock`, `Cargo.lock` y similares
- Código generado: salidas de Protobuf, artefactos de compilación, archivos minificados
- Directorios de compilación: `dist/`, `build/`, `node_modules/`, `target/`
- Accesorios de prueba: instantáneas, cassettes, datos simulados
- Líneas con más de 1.000 caracteres, que probablemente sean minificadas o generadas

Notas de atribución

Tenga en cuenta estos detalles adicionales al interpretar datos de atribución:

- El código sustancialmente reescrito por desarrolladores, con más del 20% de diferencia, no se atribuye a Claude Code
- Las sesiones fuera de la ventana de 21 días no se consideran
- El algoritmo no considera la rama de origen o destino de PR al realizar la atribución

Obtener lo máximo de los análisis

Utilice métricas de contribución para demostrar ROI, identificar patrones de adopción y encontrar miembros del equipo que puedan ayudar a otros a comenzar.

Monitorear adopción

Rastreé el gráfico de Adopción y los recuentos de usuarios para identificar:

- Usuarios activos que pueden compartir mejores prácticas
- Tendencias generales de adopción en su organización
- Caídas en el uso que pueden indicar fricción o problemas

Medir ROI

Las métricas de contribución ayudan a responder “¿Vale la pena esta herramienta la inversión?” con datos de su propio código base:

- Rastreé cambios en PRs por usuario a lo largo del tiempo a medida que aumenta la adopción
- Compare PRs y líneas de código enviadas con y sin Claude Code
- Utilice junto con [métricas DORA](#), velocidad de sprint u otros KPI de ingeniería para comprender cambios por adoptar Claude Code

Identificar usuarios avanzados

La Tabla de clasificación le ayuda a encontrar miembros del equipo con alta adopción de Claude Code que pueden:

- Compartir técnicas de prompting y flujos de trabajo con el equipo
- Proporcionar comentarios sobre qué está funcionando bien
- Ayudar a incorporar nuevos usuarios

Acceder a datos mediante programación

Para consultar estos datos a través de GitHub, busque PRs etiquetados con `claude-code-assisted`.

Acceder a análisis para clientes de API

Los clientes de API que utilizan Claude Console pueden acceder a análisis en platform.claude.com/claude-code. Necesita el permiso UsageView para acceder al panel, que se otorga a los roles Developer, Billing, Admin, Owner y Primary Owner.

Note:

Las métricas de contribución con integración de GitHub no están disponibles actualmente para clientes de API. El panel de Console muestra solo métricas de uso y gastos.

El panel de Console muestra:

- **Lines of code accepted:** líneas totales de código escritas por Claude Code que los usuarios han aceptado en sus sesiones. Esto excluye sugerencias rechazadas y no rastrea eliminaciones posteriores.
- **Suggestion accept rate:** porcentaje de veces que los usuarios aceptan el uso de herramientas de edición de código, incluidas las herramientas Edit, Write y NotebookEdit.
- **Activity:** usuarios activos diarios y sesiones mostradas en un gráfico.
- **Spend:** costos diarios de API en dólares junto con el recuento de usuarios.

Ver información del equipo

La tabla de información del equipo muestra métricas por usuario:

- **Members:** todos los usuarios que se han autenticado en Claude Code. Los usuarios de clave API se muestran por identificador de clave, los usuarios de OAuth se muestran por dirección de correo electrónico.
- **Spend this month:** costos totales de API por usuario para el mes actual.
- **Lines this month:** total por usuario de líneas de código aceptadas para el mes actual.

Note:

Las cifras de gastos en el panel de Console son estimaciones para fines de análisis. Para costos reales, consulte su página de facturación.

Recursos relacionados

- [Monitoring with OpenTelemetry](#): exportar métricas y eventos en tiempo real a su pila de observabilidad
- [Manage costs effectively](#): establecer límites de gastos y optimizar el uso de tokens
- [Permissions](#): configurar roles y permisos

Monitoreo

Aprende cómo habilitar y configurar OpenTelemetry para Claude Code.

Claude Code admite métricas y eventos de OpenTelemetry (OTel) para monitoreo y observabilidad.

Todas las métricas son datos de series temporales exportados a través del protocolo estándar de métricas de OpenTelemetry, y los eventos se exportan a través del protocolo de registros/eventos de OpenTelemetry. Es responsabilidad del usuario asegurar que sus backends de métricas y registros estén correctamente configurados y que la granularidad de agregación cumpla con sus requisitos de monitoreo.

Inicio rápido

Configura OpenTelemetry usando variables de entorno:

```
## 1. Habilitar telemetría
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. Elegir exportadores (ambos son opcionales - configura solo lo que necesites)
export OTEL_METRICS_EXPORTER=otlp      # Opciones: otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp        # Opciones: otlp, console

## 3. Configurar punto final OTLP (para exportador OTLP)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. Establecer autenticación (si es requerida)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. Para depuración: reducir intervalos de exportación
export OTEL_METRIC_EXPORT_INTERVAL=10000 # 10 segundos (predeterminado: 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000    # 5 segundos (predeterminado: 5000ms)

## 6. Ejecutar Claude Code
claude
```

Note:

Los intervalos de exportación predeterminados son 60 segundos para métricas y 5 segundos para registros. Durante la configuración, es posible que desees usar intervalos más cortos para propósitos de depuración. Recuerda restablecer estos valores para uso en producción.

Para opciones de configuración completas, consulta la [especificación de OpenTelemetry](#).

Configuración del administrador

Los administradores pueden configurar los ajustes de OpenTelemetry para todos los usuarios a través del [archivo de configuración administrada](#). Esto permite el control centralizado de los ajustes de telemetría en toda una organización. Consulta la [precedencia de configuración](#) para obtener más información sobre cómo se aplican los ajustes.

Ejemplo de configuración de ajustes administrados:

```
{
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp",
    "OTEL_LOGS_EXPORTER": "otlp",
    "OTEL_EXPORTER_OTLP_PROTOCOL": "grpc",
    "OTEL_EXPORTER_OTLP_ENDPOINT": "http://collector.company.com:4317",
    "OTEL_EXPORTER_OTLP_HEADERS": "Authorization=Bearer company-token"
  }
}
```

Note:

Los ajustes administrados pueden distribuirse a través de MDM (Mobile Device Management) u otras soluciones de gestión de dispositivos. Las variables de entorno definidas en el archivo de configuración administrada tienen alta precedencia y no pueden ser anuladas por los usuarios.

Detalles de configuración

Variables de configuración comunes

Variable de Entorno	Descripción	Valores de Ejemplo
CLAUDE_CODE_ENABLE_TELEMETRY	Habilita la recopilación de telemetría (requerido)	1
OTEL_METRICS_EXPORTER	Tipo(s) de exportador de métricas (separados por comas)	console , otlp , prometheus
OTEL_LOGS_EXPORTER	Tipo(s) de exportador de registros/ eventos (separados por comas)	console , otlp
OTEL_EXPORTER_OTLP_PROTOCOL	Protocolo para exportador OTLP (todas las señales)	grpc , http/json , http/protobuf
OTEL_EXPORTER_OTLP_ENDPOINT	Punto final del recopilador OTLP (todas las señales)	http://localhost:4317
OTEL_EXPORTER_OTLP_METRICS_PROTOCOL	Protocolo para métricas (anula el general)	grpc , http/json , http/protobuf

Variable de Entorno	Descripción	Valores de Ejemplo
OTEL_EXPORTER_OTLP_METRICS_ENDPOINT	Punto final de métricas OTLP (anula el general)	http://localhost:4318/v1/metrics
OTEL_EXPORTER_OTLP_LOGS_PROTOCOL	Protocolo para registros (anula el general)	grpc , http/json , http/protobuf
OTEL_EXPORTER_OTLP_LOGS_ENDPOINT	Punto final de registros OTLP (anula el general)	http://localhost:4318/v1/logs
OTEL_EXPORTER_OTLP_HEADERS	Encabezados de autenticación para OTLP	Authorization=Bearer token
OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY	Clave de cliente para autenticación mTLS	Ruta al archivo de clave de cliente
OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE	Certificado de cliente para autenticación mTLS	Ruta al archivo de certificado de cliente
OTEL_METRIC_EXPORT_INTERVAL	Intervalo de exportación en milisegundos (predeterminado: 60000)	5000 , 60000
OTEL_LOGS_EXPORT_INTERVAL	Intervalo de exportación de registros en milisegundos (predeterminado: 5000)	1000 , 10000
OTEL_LOG_USER_PROMPTS	Habilitar registro del contenido del mensaje del usuario (predeterminado: deshabilitado)	1 para habilitar
CLAUDE_CODE_OTEL_HELPERS_HELPER_DEBOUNCE_MS	Intervalo para actualizar encabezados dinámicos (predeterminado: 1740000ms / 29 minutos)	900000

Control de cardinalidad de métricas

Las siguientes variables de entorno controlan qué atributos se incluyen en las métricas para gestionar la cardinalidad:

Variable de Entorno	Descripción	Valor Pre-determinado	Ejemplo para Des-habilitar
<code>OTEL_METRICS_INCLUDE_SESSION_ID</code>	Incluir atributo session.id en métricas	<code>true</code>	<code>false</code>
<code>OTEL_METRICS_INCLUDE_VERSION</code>	Incluir atributo app.version en métricas	<code>false</code>	<code>true</code>
<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code>	Incluir atributo user.account_uuid en métricas	<code>true</code>	<code>false</code>

Estas variables ayudan a controlar la cardinalidad de las métricas, lo que afecta los requisitos de almacenamiento y el rendimiento de las consultas en tu backend de métricas. Una cardinalidad más baja generalmente significa mejor rendimiento y costos de almacenamiento más bajos, pero datos menos granulares para el análisis.

Encabezados dinámicos

Para entornos empresariales que requieren autenticación dinámica, puedes configurar un script para generar encabezados dinámicamente:

Configuración de ajustes

Agrega a tu `.claude/settings.json` :

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

Requisitos del script

El script debe generar JSON válido con pares clave-valor de cadena que representen encabezados HTTP:

```
#!/bin/bash
## Ejemplo: Múltiples encabezados
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

Comportamiento de actualización

El script auxiliar de encabezados se ejecuta al inicio y periódicamente después para admitir la actualización de tokens. Por defecto, el script se ejecuta cada 29 minutos. Personaliza el intervalo con la variable de entorno

`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`.

Soporte de organización multi-equipo

Las organizaciones con múltiples equipos o departamentos pueden agregar atributos personalizados para distinguir entre diferentes grupos usando la variable de entorno

`OTEL_RESOURCE_ATTRIBUTES` :

```
## Agregar atributos personalizados para identificación de equipo
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

Estos atributos personalizados se incluirán en todas las métricas y eventos, permitiéndote:

- Filtrar métricas por equipo o departamento
- Rastrear costos por centro de costos
- Crear paneles específicos del equipo
- Configurar alertas para equipos específicos

Warning:

Requisitos de formato importantes para `OTEL_RESOURCE_ATTRIBUTES`:

La variable de entorno `OTEL_RESOURCE_ATTRIBUTES` sigue la [especificación W3C Baggage](#), que tiene requisitos de formato estrictos:

- **No se permiten espacios:** Los valores no pueden contener espacios. Por ejemplo, `user.organizationName=My Company` es inválido
- **Formato:** Debe ser pares clave=valor separados por comas:
`key1=value1,key2=value2`
- **Caracteres permitidos:** Solo caracteres US-ASCII excluyendo caracteres de control, espacios en blanco, comillas dobles, comas, puntos y comas, y barras invertidas

- **Caracteres especiales:** Los caracteres fuera del rango permitido deben estar codificados en porcentaje

Ejemplos:

```
## ❌ Inválido - contiene espacios
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅ Válido - usar guiones bajos o camelCase en su lugar
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅ Válido - codificar en porcentaje caracteres especiales si es necesario
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

Nota: envolver valores entre comillas no escapa espacios. Por ejemplo, `org.name="My Company"` resulta en el valor literal `"My Company"` (con comillas incluidas), no `My Company`.

Configuraciones de ejemplo

```

## Depuración de consola (intervalos de 1 segundo)
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console
export OTEL_METRIC_EXPORT_INTERVAL=1000

## OTLP/gRPC
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## Prometheus
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=prometheus

## Múltiples exportadores
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json

## Diferentes puntos finales/backends para métricas y registros
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_METRICS_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_METRICS_ENDPOINT=http://metrics.company.com:4318
export OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=http://logs.company.com:4317

## Solo métricas (sin eventos/registros)
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## Solo eventos/registros (sin métricas)
export CLAUDE_CODE_ENABLE_TELEMETRY=1

```

```
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

Métricas y eventos disponibles

Atributos estándar

Todas las métricas y eventos comparten estos atributos estándar:

Atributo	Descripción	Controlado Por
<code>session.id</code>	Identificador único de sesión	<code>OTEL_METRICS_INCLUDE_SESSION_ID</code> (predeterminado: true)
<code>app.version</code>	Versión actual de Claude Code	<code>OTEL_METRICS_INCLUDE_VERSION</code> (predeterminado: false)
<code>organization.id</code>	UUID de organización (cuando está autenticado)	Siempre incluido cuando está disponible
<code>user.account.uuid</code>	UUID de cuenta (cuando está autenticado)	<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code> (predeterminado: true)
<code>terminal.type</code>	Tipo de terminal (por ejemplo, <code>iTerm.app</code> , <code>vscode</code> , <code>cursor</code> , <code>tmux</code>)	Siempre incluido cuando se detecta

Métricas

Claude Code exporta las siguientes métricas:

Nombre de Métrica	Descripción	Unidad
<code>claude_code.session.count</code>	Recuento de sesiones CLI iniciadas	count
<code>claude_code.lines_of_code.count</code>	Recuento de líneas de código modificadas	count
<code>claude_code.pull_request.count</code>	Número de solicitudes de extracción creadas	count

Nombre de Métrica	Descripción	Unidad
<code>claude_code.commit.count</code>	Número de confirmaciones de git creadas	count
<code>claude_code.cost.usage</code>	Costo de la sesión de Claude Code	USD
<code>claude_code.token.usage</code>	Número de tokens utilizados	tokens
<code>claude_code.code_edit_tool.decision</code>	Recuento de decisiones de permisos de herramienta de edición de código	count
<code>claude_code.active_time.total</code>	Tiempo activo total en segundos	s

Detalles de métricas

Contador de sesión

Se incrementa al inicio de cada sesión.

Atributos:

- Todos los [atributos estándar](#)

Contador de líneas de código

Se incrementa cuando se agrega o se elimina código.

Atributos:

- Todos los [atributos estándar](#)
- `type` : (`"added"` , `"removed"`)

Contador de solicitud de extracción

Se incrementa al crear solicitudes de extracción a través de Claude Code.

Atributos:

- Todos los [atributos estándar](#)

Contador de confirmación

Se incrementa al crear confirmaciones de git a través de Claude Code.

Atributos:

- Todos los [atributos estándar](#)

Contador de costo

Se incrementa después de cada solicitud de API.

Atributos:

- Todos los [atributos estándar](#)
- `model` : Identificador de modelo (por ejemplo, “claude-sonnet-4-5-20250929”)

Contador de tokens

Se incrementa después de cada solicitud de API.

Atributos:

- Todos los [atributos estándar](#)
- `type` : (`"input"` , `"output"` , `"cacheRead"` , `"cacheCreation"`)
- `model` : Identificador de modelo (por ejemplo, “claude-sonnet-4-5-20250929”)

Contador de decisión de herramienta de edición de código

Se incrementa cuando el usuario acepta o rechaza el uso de herramientas Edit, Write o NotebookEdit.

Atributos:

- Todos los [atributos estándar](#)
- `tool` : Nombre de la herramienta (`"Edit"` , `"Write"` , `"NotebookEdit"`)
- `decision` : Decisión del usuario (`"accept"` , `"reject"`)
- `language` : Lenguaje de programación del archivo editado (por ejemplo, `"TypeScript"` , `"Python"` , `"JavaScript"` , `"Markdown"`). Devuelve `"unknown"` para extensiones de archivo no reconocidas.

Contador de tiempo activo

Rastrea el tiempo real dedicado a usar activamente Claude Code (no tiempo inactivo). Esta métrica se incrementa durante interacciones del usuario como escribir mensajes o recibir respuestas.

Atributos:

- Todos los [atributos estándar](#)

Eventos

Claude Code exporta los siguientes eventos a través de registros/eventos de OpenTelemetry (cuando `OTEL_LOGS_EXPORTER` está configurado):

Evento de mensaje del usuario

Se registra cuando un usuario envía un mensaje.

Nombre del Evento: `claude_code.user_prompt`

Atributos:

- Todos los [atributos estándar](#)
- `event.name` : `"user_prompt"`
- `event.timestamp` : Marca de tiempo ISO 8601
- `prompt_length` : Longitud del mensaje
- `prompt` : Contenido del mensaje (redactado por defecto, habilitar con `OTEL_LOG_USER_PROMPTS=1`)

Evento de resultado de herramienta

Se registra cuando una herramienta completa la ejecución.

Nombre del Evento: `claude_code.tool_result`

Atributos:

- Todos los [atributos estándar](#)
- `event.name` : `"tool_result"`
- `event.timestamp` : Marca de tiempo ISO 8601
- `tool_name` : Nombre de la herramienta
- `success` : `"true"` o `"false"`
- `duration_ms` : Tiempo de ejecución en milisegundos
- `error` : Mensaje de error (si falló)
- `decision` : Ya sea `"accept"` o `"reject"`
- `source` : Fuente de decisión - `"config"` , `"user_permanent"` , `"user_temporary"` , `"user_abort"` , o `"user_reject"`
- `tool_parameters` : Cadena JSON que contiene parámetros específicos de la herramienta (cuando está disponible)
 - Para herramienta Bash: incluye `bash_command` , `full_command` , `timeout` , `description` , `sandbox`

Evento de solicitud de API

Se registra para cada solicitud de API a Claude.

Nombre del Evento: `claude_code.api_request`

Atributos:

- Todos los [atributos estándar](#)
- `event.name` : `"api_request"`
- `event.timestamp` : Marca de tiempo ISO 8601
- `model` : Modelo utilizado (por ejemplo, "claude-sonnet-4-5-20250929")
- `cost_usd` : Costo estimado en USD
- `duration_ms` : Duración de la solicitud en milisegundos
- `input_tokens` : Número de tokens de entrada
- `output_tokens` : Número de tokens de salida
- `cache_read_tokens` : Número de tokens leídos del caché
- `cache_creation_tokens` : Número de tokens utilizados para la creación del caché

Evento de error de API

Se registra cuando una solicitud de API a Claude falla.

Nombre del Evento: `claude_code.api_error`

Atributos:

- Todos los [atributos estándar](#)
- `event.name` : `"api_error"`
- `event.timestamp` : Marca de tiempo ISO 8601
- `model` : Modelo utilizado (por ejemplo, "claude-sonnet-4-5-20250929")
- `error` : Mensaje de error
- `status_code` : Código de estado HTTP (si aplica)
- `duration_ms` : Duración de la solicitud en milisegundos
- `attempt` : Número de intento (para solicitudes reintentadas)

Evento de decisión de herramienta

Se registra cuando se toma una decisión de permiso de herramienta (aceptar/rechazar).

Nombre del Evento: `claude_code.tool_decision`

Atributos:

- Todos los [atributos estándar](#)
- `event.name` : "tool_decision"
- `event.timestamp` : Marca de tiempo ISO 8601
- `tool_name` : Nombre de la herramienta (por ejemplo, "Read", "Edit", "Write", "NotebookEdit")
- `decision` : Ya sea "accept" o "reject"
- `source` : Fuente de decisión - "config" , "user_permanent" , "user_temporary" , "user_abort" , o "user_reject"

Interpretación de datos de métricas y eventos

Las métricas exportadas por Claude Code proporcionan información valiosa sobre patrones de uso y productividad. Aquí hay algunas visualizaciones y análisis comunes que puedes crear:

Monitoreo de uso

Métrica	Oportunidad de Análisis
<code>claude_code.token.usage</code>	Desglosar por <code>type</code> (entrada/salida), usuario, equipo o modelo
<code>claude_code.session.count</code>	Rastrear adopción y compromiso a lo largo del tiempo
<code>claude_code.lines_of_code.count</code>	Medir productividad rastreando adiciones/eliminaciones de código
<code>claude_code.commit.count</code> & <code>claude_code.pull_request.count</code>	Entender el impacto en los flujos de trabajo de desarrollo

Monitoreo de costos

La métrica `claude_code.cost.usage` ayuda con:

- Rastrear tendencias de uso entre equipos o individuos
- Identificar sesiones de alto uso para optimización

Note:

Las métricas de costo son aproximaciones. Para datos de facturación oficiales, consulta tu proveedor de API (Claude Console, AWS Bedrock o Google Cloud Vertex).

Alertas y segmentación

Alertas comunes a considerar:

- Picos de costo
- Consumo inusual de tokens
- Alto volumen de sesiones de usuarios específicos

Todas las métricas pueden segmentarse por `user.account_uuid`, `organization.id`, `session.id`, `model` y `app.version`.

Análisis de eventos

Los datos de eventos proporcionan información detallada sobre las interacciones de Claude Code:

Patrones de Uso de Herramientas: analizar eventos de resultado de herramientas para identificar:

- Herramientas más utilizadas frecuentemente
- Tasas de éxito de herramientas
- Tiempos de ejecución promedio de herramientas
- Patrones de error por tipo de herramienta

Monitoreo de Rendimiento: rastrear duraciones de solicitudes de API y tiempos de ejecución de herramientas para identificar cuellos de botella de rendimiento.

Consideraciones de backend

Tu elección de backends de métricas y registros determina los tipos de análisis que puedes realizar:

Para métricas

- **Bases de datos de series temporales (por ejemplo, Prometheus):** Cálculos de tasa, métricas agregadas
- **Almacenes columnares (por ejemplo, ClickHouse):** Consultas complejas, análisis de usuario único
- **Plataformas de observabilidad completas (por ejemplo, Honeycomb, Data-dog):** Consultas avanzadas, visualización, alertas

Para eventos/registros

- **Sistemas de agregación de registros (por ejemplo, Elasticsearch, Loki):** Búsqueda de texto completo, análisis de registros
- **Almacenes columnares (por ejemplo, ClickHouse):** Análisis de eventos estructurados
- **Plataformas de observabilidad completas (por ejemplo, Honeycomb, Datadog):** Correlación entre métricas y eventos

Para organizaciones que requieren métricas de Usuarios Activos Diarios/Semanales/Mensuales (DAU/WAU/MAU), considera backends que admitan consultas de valores únicos eficientes.

Información del servicio

Todas las métricas y eventos se exportan con los siguientes atributos de recurso:

- `service.name` : `claude-code`
- `service.version` : Versión actual de Claude Code
- `os.type` : Tipo de sistema operativo (por ejemplo, `linux` , `darwin` , `windows`)
- `os.version` : Cadena de versión del sistema operativo
- `host.arch` : Arquitectura del host (por ejemplo, `amd64` , `arm64`)
- `wsl.version` : Número de versión de WSL (solo presente cuando se ejecuta en Windows Subsystem for Linux)
- Nombre del Medidor: `com.anthropic.claude_code`

Recursos de medición de ROI

Para una guía completa sobre cómo medir el retorno de inversión para Claude Code, incluyendo configuración de telemetría, análisis de costos, métricas de productividad e informes automatizados, consulta la [Guía de Medición de ROI de Claude Code](#). Este repositorio proporciona configuraciones de Docker Compose listas para usar, configuraciones de Prometheus y OpenTelemetry, y plantillas para generar informes de productividad integrados con herramientas como Linear.

Consideraciones de seguridad/privacidad

- La telemetría es opcional y requiere configuración explícita
- La información sensible como claves de API o contenidos de archivos nunca se incluye en métricas o eventos

- El contenido del mensaje del usuario se redacta por defecto - solo se registra la longitud del mensaje. Para habilitar el registro de mensajes del usuario, establece

`OTEL_LOG_USER_PROMPTS=1`

Monitoreo de Claude Code en Amazon Bedrock

Para orientación detallada sobre monitoreo de uso de Claude Code para Amazon Bedrock, consulta [Implementación de Monitoreo de Claude Code \(Bedrock\)](#).

Descripción general de implementación empresarial

Aprende cómo Claude Code puede integrarse con varios servicios de terceros e infraestructura para cumplir con los requisitos de implementación empresarial.

Esta página proporciona una descripción general de las opciones de implementación disponibles y te ayuda a elegir la configuración correcta para tu organización.

Comparación de proveedores

Característica Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

Regiones [Países](#) compatibles Múltiples [regiones](#) de AWS Múltiples [regiones](#) de GCP Múltiples [regiones](#) de Azure

Almacenamiento en caché de indicaciones Habilitado de forma predeterminada Habilitado de forma predeterminada Habilitado de forma predeterminada Habilitado de forma predeterminada

Autenticación Clave API Clave API o credenciales de AWS Credenciales de GCP Clave API o Microsoft Entra ID

Seguimiento de costos Panel de control AWS Cost Explorer Facturación de GCP Administración de costos de Azure

Características empresariales Equipos, monitoreo de uso Políticas de IAM, CloudTrail Roles de IAM, registros de auditoría en la nube Políticas de RBAC, Azure Monitor

Proveedores de nube

- [Amazon Bedrock](#) Utiliza modelos Claude a través de la infraestructura de AWS con autenticación basada en clave API o IAM y monitoreo nativo de AWS
- [Google Vertex AI](#) Accede a modelos Claude a través de Google Cloud Platform con seguridad y cumplimiento de nivel empresarial
- [Microsoft Foundry](#) Accede a Claude a través de Azure con autenticación de clave API o Microsoft Entra ID y facturación de Azure

Infraestructura corporativa

- [Red empresarial](#) Configura Claude Code para trabajar con los servidores proxy de tu organización y requisitos de SSL/TLS
- [Puerta de enlace LLM](#) Implementa acceso centralizado a modelos con seguimiento de uso, presupuesto y registro de auditoría

Descripción general de la configuración

Claude Code admite opciones de configuración flexible que te permiten combinar diferentes proveedores e infraestructura:

Note:

Comprende la diferencia entre:

- **Proxy corporativo:** Un proxy HTTP/HTTPS para enrutar tráfico (establecido a través de `HTTPS_PROXY` o `HTTP_PROXY`)
- **Puerta de enlace LLM:** Un servicio que maneja la autenticación y proporciona puntos finales compatibles con proveedores (establecido a través de `ANTHROPIC_BASE_URL` , `ANTHROPIC_BEDROCK_BASE_URL` , o `ANTHROPIC_VERTEX_BASE_URL`)

Ambas configuraciones se pueden usar en conjunto.

Uso de Bedrock con proxy corporativo

Enruta el tráfico de Bedrock a través de un proxy HTTP/HTTPS corporativo:

```
## Habilitar Bedrock
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## Configurar proxy corporativo
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Uso de Bedrock con puerta de enlace LLM

Utiliza un servicio de puerta de enlace que proporciona puntos finales compatibles con Bedrock:


```
## Habilitar Bedrock
export CLAUDE_CODE_USE_BEDROCK=1

## Configurar puerta de enlace LLM
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1 # Si la puerta de enlace maneja la
autenticación de AWS
```

Uso de Foundry con proxy corporativo

Enruta el tráfico de Azure a través de un proxy HTTP/HTTPS corporativo:

```
## Habilitar Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key # O omitir para autenticación de
Entra ID

## Configurar proxy corporativo
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Uso de Foundry con puerta de enlace LLM

Utiliza un servicio de puerta de enlace que proporciona puntos finales compatibles con Azure:

```
## Habilitar Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## Configurar puerta de enlace LLM
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1 # Si la puerta de enlace maneja la
autenticación de Azure
```

Uso de Vertex AI con proxy corporativo

Enruta el tráfico de Vertex AI a través de un proxy HTTP/HTTPS corporativo:

```
## Habilitar Vertex
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## Configurar proxy corporativo
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Uso de Vertex AI con puerta de enlace LLM

Combina modelos de Google Vertex AI con una puerta de enlace LLM para gestión centralizada:

```
## Habilitar Vertex
export CLAUDE_CODE_USE_VERTEX=1

## Configurar puerta de enlace LLM
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1 # Si la puerta de enlace maneja la
autenticación de GCP
```

Configuración de autenticación

Claude Code utiliza `ANTHROPIC_AUTH_TOKEN` para el encabezado `Authorization` cuando sea necesario. Las banderas `SKIP_AUTH` (`CLAUDE_CODE_SKIP_BEDROCK_AUTH` , `CLAUDE_CODE_SKIP_VERTEX_AUTH`) se utilizan en escenarios de puerta de enlace LLM donde la puerta de enlace maneja la autenticación del proveedor.

Elegir la configuración de implementación correcta

Considera estos factores al seleccionar tu enfoque de implementación:

Acceso directo al proveedor

Mejor para organizaciones que:

- Desean la configuración más simple
- Tienen infraestructura existente de AWS o GCP
- Necesitan monitoreo nativo del proveedor y cumplimiento

Proxy corporativo

Mejor para organizaciones que:

- Tienen requisitos de proxy corporativo existentes
- Necesitan monitoreo de tráfico y cumplimiento
- Deben enrutar todo el tráfico a través de rutas de red específicas

Puerta de enlace LLM

Mejor para organizaciones que:

- Necesitan seguimiento de uso entre equipos
- Desean cambiar dinámicamente entre modelos
- Requieren limitación de velocidad personalizada o presupuestos
- Necesitan gestión centralizada de autenticación

Depuración

Al depurar tu implementación:

- Utiliza el [comando de barra](#) `claude /status`. Este comando proporciona observabilidad en cualquier autenticación, proxy y configuración de URL aplicados.
- Establece la variable de entorno `export ANTHROPIC_LOG=debug` para registrar solicitudes.

Mejores prácticas para organizaciones

1. Invertir en documentación y memoria

Recomendamos encarecidamente invertir en documentación para que Claude Code comprenda tu base de código. Las organizaciones pueden implementar archivos CLAUDE.md en múltiples niveles:

- **En toda la organización:** Implementa en directorios del sistema como `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS) para estándares de toda la empresa
- **A nivel de repositorio:** Crea archivos `CLAUDE.md` en raíces de repositorio que contengan arquitectura del proyecto, comandos de compilación y directrices de contribución. Verifica estos en el control de código fuente para que todos los usuarios se beneficien

[Aprende más.](#)

2. Simplificar la implementación

Si tienes un entorno de desarrollo personalizado, encontramos que crear una forma “de un clic” para instalar Claude Code es clave para aumentar la adopción en toda una organización.

3. Comenzar con uso guiado

Anima a los nuevos usuarios a probar Claude Code para preguntas sobre la base de código, o en correcciones de errores más pequeñas o solicitudes de características. Pide a Claude Code que haga un plan. Verifica las sugerencias de Claude y proporciona retroalimentación si se desvía. Con el tiempo, a medida que los usuarios comprendan mejor este nuevo paradigma, serán más efectivos al permitir que Claude Code funcione de manera más autónoma.

4. Configurar políticas de seguridad

Los equipos de seguridad pueden configurar permisos administrados para lo que Claude Code puede y no puede hacer, lo que no puede ser sobrescrito por la configuración local. [Aprende más.](#)

5. Aprovechar MCP para integraciones

MCP es una excelente manera de dar a Claude Code más información, como conectarse a sistemas de gestión de tickets o registros de errores. Recomendamos que un equipo central configure servidores MCP y verifique una configuración `.mcp.json` en la base de código para que todos los usuarios se beneficien. [Aprende más.](#)

En Anthropic, confiamos en Claude Code para potenciar el desarrollo en todas las bases de código de Anthropic. Esperamos que disfrutes usando Claude Code tanto como nosotros.

Próximos pasos

- [Configurar Amazon Bedrock](#) para implementación nativa de AWS
- [Configurar Google Vertex AI](#) para implementación de GCP
- [Configurar Microsoft Foundry](#) para implementación de Azure
- [Configurar red empresarial](#) para requisitos de red
- [Implementar puerta de enlace LLM](#) para gestión empresarial
- [Configuración](#) para opciones de configuración y variables de entorno

Configurar la configuración administrada por servidor (beta pública)

Configure centralmente Claude Code para su organización a través de configuración entregada por servidor, sin requerir infraestructura de administración de dispositivos.

La configuración administrada por servidor permite a los administradores configurar centralmente Claude Code a través de una interfaz basada en web en Claude.ai. Los clientes de Claude Code reciben automáticamente estas configuraciones cuando los usuarios se autentican con sus credenciales organizacionales.

Este enfoque está diseñado para organizaciones que no tienen infraestructura de administración de dispositivos implementada, o que necesitan administrar configuraciones para usuarios en dispositivos no administrados.

Note:

La configuración administrada por servidor está en beta pública y disponible para clientes de [Claude for Teams](#) y [Claude for Enterprise](#). Las características pueden evolucionar antes de la disponibilidad general.

Requisitos

Para usar la configuración administrada por servidor, necesita:

- Plan Claude for Teams o Claude for Enterprise
- Claude Code versión 2.1.38 o posterior para Claude for Teams, o versión 2.1.30 o posterior para Claude for Enterprise
- Acceso de red a api.anthropic.com

Elegir entre configuración administrada por servidor y administrada por punto final

Claude Code admite dos enfoques para la configuración centralizada. La configuración administrada por servidor entrega la configuración desde los servidores de Anthropic. La [configuración administrada por punto final](#) se implementa directamente en dispositivos a través de políticas nativas del sistema operativo (preferencias administradas de macOS, registro de Windows) o archivos de configuración administrados.

Enfoque	Mejor para	Modelo de seguridad
Configuración administrada por servidor	Organizaciones sin MDM, o usuarios en dispositivos no administrados	Configuración entregada desde los servidores de Anthropic en el momento de la autenticación
Configuración administrada por punto final	Organizaciones con MDM o administración de puntos finales	Configuración implementada en dispositivos a través de perfiles de configuración MDM, políticas de registro o archivos de configuración administrados

Si sus dispositivos están inscritos en una solución MDM o de administración de puntos finales, la configuración administrada por punto final proporciona garantías de seguridad más sólidas porque el archivo de configuración puede protegerse de la modificación del usuario a nivel del sistema operativo.

Configurar la configuración administrada por servidor

Step 1: Abrir la consola de administración

En [Claude.ai](#), navegue a **Admin Settings > Claude Code > Managed settings**.

Step 2: Definir su configuración

Agregue su configuración como JSON. Todas las [configuraciones disponibles en settings.json](#) son compatibles, incluidas las [configuraciones solo administradas](#) como `disableBypassPermissionsMode`.

Este ejemplo aplica una lista de denegación de permisos e impide que los usuarios omitan permisos:

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: Guardar e implementar

Guarde sus cambios. Los clientes de Claude Code reciben la configuración actualizada en su próximo inicio o ciclo de sondeo por hora.

Verificar la entrega de configuración

Para confirmar que la configuración se está aplicando, pida a un usuario que reinicie Claude Code. Si la configuración incluye configuraciones que activan el [diálogo de aprobación de seguridad](#), el usuario ve un mensaje que describe la configuración administrada al inicio. También puede verificar que las reglas de permisos administrados estén activas haciendo que un usuario ejecute `/permissions` para ver sus reglas de permisos efectivas.

Control de acceso

Los siguientes roles pueden administrar la configuración administrada por servidor:

- **Propietario principal**
- **Propietario**

Restrinja el acceso al personal de confianza, ya que los cambios de configuración se aplican a todos los usuarios de la organización.

Limitaciones actuales

La configuración administrada por servidor tiene las siguientes limitaciones durante el período beta:

- La configuración se aplica uniformemente a todos los usuarios de la organización. Las configuraciones por grupo aún no son compatibles.

- Las [configuraciones de servidor MCP](#) no se pueden distribuir a través de la configuración administrada por servidor.

Entrega de configuración

Precedencia de configuración

La configuración administrada por servidor y la [configuración administrada por punto final](#) ocupan el nivel más alto en la [jerarquía de configuración](#) de Claude Code. Ningún otro nivel de configuración puede anularlas, incluidos los argumentos de línea de comandos. Cuando ambas están presentes, la configuración administrada por servidor tiene prioridad y la configuración administrada por punto final no se utiliza.

Comportamiento de obtención y almacenamiento en caché

Claude Code obtiene la configuración de los servidores de Anthropic al inicio y sondea actualizaciones cada hora durante sesiones activas.

Primer lanzamiento sin configuración en caché:

- Claude Code obtiene la configuración de forma asincrónica
- Si la obtención falla, Claude Code continúa sin configuración administrada
- Hay una breve ventana antes de que se cargue la configuración donde las restricciones aún no se aplican

Lanzamientos posteriores con configuración en caché:

- La configuración en caché se aplica inmediatamente al inicio
- Claude Code obtiene configuración nueva en segundo plano
- La configuración en caché persiste a través de fallos de red

Claude Code aplica actualizaciones de configuración automáticamente sin reinicio, excepto para configuraciones avanzadas como la configuración de OpenTelemetry, que requieren un reinicio completo para tomar efecto.

Díálogos de aprobación de seguridad

Ciertas configuraciones que podrían presentar riesgos de seguridad requieren aprobación explícita del usuario antes de ser aplicadas:

- **Configuraciones de comandos de shell:** configuraciones que ejecutan comandos de shell
- **Variables de entorno personalizadas:** variables que no están en la lista de permitidos conocida y segura

- **Configuraciones de hooks:** cualquier definición de hook

Cuando estas configuraciones están presentes, los usuarios ven un diálogo de seguridad que explica qué se está configurando. Los usuarios deben aprobar para continuar. Si un usuario rechaza la configuración, Claude Code se cierra.

Note:

En modo no interactivo con la bandera `-p`, Claude Code omite los diálogos de seguridad y aplica la configuración sin aprobación del usuario.

Disponibilidad de plataforma

La configuración administrada por servidor requiere una conexión directa a `api.anthropic.com` y no está disponible cuando se utilizan proveedores de modelos de terceros:

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- Puntos finales de API personalizados a través de `ANTHROPIC_BASE_URL` o [puertas de enlace LLM](#)

Registro de auditoría

Los eventos del registro de auditoría para cambios de configuración están disponibles a través de la API de cumplimiento o exportación del registro de auditoría. Póngase en contacto con su equipo de cuenta de Anthropic para obtener acceso.

Los eventos de auditoría incluyen el tipo de acción realizada, la cuenta y el dispositivo que realizó la acción, y referencias a los valores anteriores y nuevos.

Consideraciones de seguridad

La configuración administrada por servidor proporciona aplicación de políticas centralizada, pero funciona como un control del lado del cliente. En dispositivos no administrados, los usuarios con acceso de administrador o sudo pueden modificar el binario de Claude Code, el sistema de archivos o la configuración de red.

Escenario	Comportamiento
El usuario edita el archivo de configuración en caché	El archivo manipulado se aplica al inicio, pero la configuración correcta se restaura en la siguiente obtención del servidor
El usuario elimina el archivo de configuración en caché	Ocurre el comportamiento del primer lanzamiento: la configuración se obtiene de forma asincrónica con una breve ventana no aplicada
La API no está disponible	La configuración en caché se aplica si está disponible, de lo contrario, la configuración administrada no se aplica hasta la siguiente obtención exitosa
El usuario se autentica con una organización diferente	La configuración no se entrega para cuentas fuera de la organización administrada
El usuario establece un <code>ANTHROPIC_BASE_URL</code> no predeterminado	La configuración administrada por servidor se omite cuando se utilizan proveedores de API de terceros

Para detectar cambios de configuración en tiempo de ejecución, use [hooks](#) `ConfigChange` para registrar modificaciones o bloquear cambios no autorizados antes de que surtan efecto.

Para garantías de aplicación más sólidas, use la [configuración administrada por punto final](#) en dispositivos inscritos en una solución MDM.

Ver también

Páginas relacionadas para administrar la configuración de Claude Code:

- [Settings](#): referencia de configuración completa que incluye todas las configuraciones disponibles
- [Endpoint-managed settings](#): configuración administrada implementada en dispositivos por TI
- [Authentication](#): configurar el acceso de usuarios a Claude Code
- [Security](#): salvaguardas de seguridad y mejores prácticas

Part 11: Cloud Providers

Claude Code en Amazon Bedrock

Aprenda a configurar Claude Code a través de Amazon Bedrock, incluyendo configuración, configuración de IAM y solución de problemas.

Requisitos previos

Antes de configurar Claude Code con Bedrock, asegúrese de tener:

- Una cuenta de AWS con acceso a Bedrock habilitado
- Acceso a los modelos Claude deseados (por ejemplo, Claude Sonnet 4.6) en Bedrock
- AWS CLI instalado y configurado (opcional - solo se necesita si no tiene otro mecanismo para obtener credenciales)
- Permisos de IAM apropiados

Note:

Si está implementando Claude Code para múltiples usuarios, [fije las versiones de su modelo](#) para evitar problemas cuando Anthropic lance nuevos modelos.

Configuración

1. Envíe los detalles del caso de uso

Los usuarios por primera vez de modelos de Anthropic deben enviar detalles del caso de uso antes de invocar un modelo. Esto se realiza una vez por cuenta.

1. Asegúrese de tener los permisos de IAM correctos (vea más sobre eso a continuación)
2. Navegue a la [consola de Amazon Bedrock](#)
3. Seleccione **Chat/Text playground**
4. Elija cualquier modelo de Anthropic y se le pedirá que complete el formulario de caso de uso

2. Configure las credenciales de AWS

Claude Code utiliza la cadena de credenciales predeterminada del SDK de AWS. Configure sus credenciales utilizando uno de estos métodos:

Opción A: Configuración de AWS CLI

```
aws configure
```

Opción B: Variables de entorno (clave de acceso)

```
export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-access-key
export AWS_SESSION_TOKEN=your-session-token
```

Opción C: Variables de entorno (perfil SSO)

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

Opción D: Credenciales de la consola de administración de AWS

```
aws login
```

[Obtenga más información](#) sobre `aws login`.

Opción E: Claves de API de Bedrock

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Las claves de API de Bedrock proporcionan un método de autenticación más simple sin necesidad de credenciales completas de AWS. [Obtenga más información sobre las claves de API de Bedrock](#).

Configuración avanzada de credenciales

Claude Code admite la actualización automática de credenciales para AWS SSO y proveedores de identidad corporativos. Agregue estas configuraciones a su archivo de configuración de Claude Code (vea [Configuración](#) para ubicaciones de archivos).

Cuando Claude Code detecta que sus credenciales de AWS han expirado (ya sea localmente según su marca de tiempo o cuando Bedrock devuelve un error de credencial), ejecutará automáticamente sus comandos `awsAuthRefresh` y/o `awsCredentialExport` configurados para obtener nuevas credenciales antes de reintentar la solicitud.

Configuración de ejemplo

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

Configuración explicada

awsAuthRefresh : Utilice esto para comandos que modifiquen el directorio `.aws`, como actualizar credenciales, caché de SSO o archivos de configuración. La salida del comando se muestra al usuario, pero la entrada interactiva no es compatible. Esto funciona bien para flujos de SSO basados en navegador donde la CLI muestra una URL o código y usted completa la autenticación en el navegador.

awsCredentialExport : Solo use esto si no puede modificar `.aws` y debe devolver credenciales directamente. La salida se captura silenciosamente y no se muestra al usuario. El comando debe generar JSON en este formato:

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Configure Claude Code

Establezca las siguientes variables de entorno para habilitar Bedrock:

```
## Enable Bedrock integration
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1 # or your preferred region

## Optional: Override the region for the small/fast model (Haiku)
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Al habilitar Bedrock para Claude Code, tenga en cuenta lo siguiente:

- `AWS_REGION` es una variable de entorno requerida. Claude Code no lee desde el archivo de configuración `.aws` para esta configuración.
- Cuando se usa Bedrock, los comandos `/login` y `/logout` están deshabilitados ya que la autenticación se maneja a través de credenciales de AWS.
- Puede usar archivos de configuración para variables de entorno como `AWS_PROFILE` que no desea filtrar a otros procesos. Vea [Configuración](#) para más información.

4. Fije las versiones del modelo

Warning:

Fije versiones de modelo específicas para cada implementación. Si utiliza alias de modelo (`sonnet` , `opus` , `haiku`) sin fijar, Claude Code puede intentar utilizar una versión de modelo más nueva que no está disponible en su cuenta de Bedrock, rompiendo usuarios existentes cuando Anthropic lanza actualizaciones.

Establezca estas variables de entorno en IDs de modelo de Bedrock específicos:

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'
export ANTHROPIC_DEFAULT_SONNET_MODEL='us.anthropic.claude-sonnet-4-6'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'
```

Estas variables utilizan IDs de perfil de inferencia entre regiones (con el prefijo `us.`). Si utiliza un prefijo de región diferente o perfiles de inferencia de aplicación, ajuste en consecuencia. Para IDs de modelo actuales y heredados, vea [Descripción general de modelos](#). Vea [Configuración de modelo](#) para la lista completa de variables de entorno.

Claude Code utiliza estos modelos predeterminados cuando no se establecen variables de fijación:

Tipo de modelo	Valor predeterminado
Modelo principal	<code>global.anthropic.claude-sonnet-4-6</code>
Modelo pequeño/rápido	<code>us.anthropic.claude-haiku-4-5-20251001-v1:0</code>

Para personalizar modelos aún más, utilice uno de estos métodos:

```
## Using inference profile ID
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## Using application inference profile ARN
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
inference-profile/your-model-id'

## Optional: Disable prompt caching if needed
export DISABLE_PROMPT_CACHING=1
```

Note:

[Prompt caching](#) puede no estar disponible en todas las regiones.

Asigne cada versión de modelo a un perfil de inferencia

Las variables de entorno `ANTHROPIC_DEFAULT_*_MODEL` configuran un perfil de inferencia por familia de modelo. Si su organización necesita exponer varias versiones de la misma familia en el selector `/model`, cada una enrutada a su propio ARN de perfil de inferencia de aplicación, utilice la configuración `modelOverrides` en su [archivo de configuración](#) en su lugar.

Este ejemplo asigna tres versiones de Opus a ARN distintos para que los usuarios puedan cambiar entre ellas sin eludir los perfiles de inferencia de su organización:

```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/opus-46-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-opus-4-1-20250805": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-41-prod"
  }
}
```

Cuando un usuario selecciona una de estas versiones en `/model`, Claude Code llama a Bedrock con el ARN asignado. Las versiones sin una anulación se reversionan al ID de modelo de Bedrock integrado o a cualquier perfil de inferencia coincidente descubierto al inicio. Vea [Anular IDs de modelo por versión](#) para detalles sobre cómo las anulaciones interactúan con `availableModels` y otras configuraciones de modelo.

Configuración de IAM

Cree una política de IAM con los permisos requeridos para Claude Code:


```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowModelAndInferenceProfileAccess",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream",
        "bedrock:ListInferenceProfiles"
      ],
      "Resource": [
        "arn:aws:bedrock:*:*:inference-profile/*",
        "arn:aws:bedrock:*:*:application-inference-profile/*",
        "arn:aws:bedrock:*:*:foundation-model/*"
      ]
    },
    {
      "Sid": "AllowMarketplaceSubscription",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "bedrock.amazonaws.com"
        }
      }
    }
  ]
}

```

Para permisos más restrictivos, puede limitar el Resource a ARN de perfil de inferencia específicos.

Para más detalles, vea [documentación de IAM de Bedrock](#).

Note:

Cree una cuenta de AWS dedicada para Claude Code para simplificar el seguimiento de costos y el control de acceso.

AWS Guardrails

[Amazon Bedrock Guardrails](#) le permite implementar filtrado de contenido para Claude Code. Cree un Guardrail en la [consola de Amazon Bedrock](#), publique una versión, luego agregue los encabezados de Guardrail a su [archivo de configuración](#). Habilite la inferencia entre regiones en su Guardrail si está utilizando perfiles de inferencia entre regiones.

Configuración de ejemplo:

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

Solución de problemas

Si encuentra problemas de región:

- Verifique la disponibilidad del modelo: `aws bedrock list-inference-profiles --region your-region`
- Cambie a una región compatible: `export AWS_REGION=us-east-1`
- Considere usar perfiles de inferencia para acceso entre regiones

Si recibe un error “on-demand throughput isn’t supported”:

- Especifique el modelo como un ID de [perfil de inferencia](#)

Claude Code utiliza la [API de Invoke](#) de Bedrock y no admite la API de Converse.

Recursos adicionales

- [Documentación de Bedrock](#)
- [Precios de Bedrock](#)
- [Perfiles de inferencia de Bedrock](#)

- [Claude Code en Amazon Bedrock: Guía de configuración rápida](#)
- [Implementación de monitoreo de Claude Code \(Bedrock\)](#)

Claude Code en Google Vertex AI

Aprenda a configurar Claude Code a través de Google Vertex AI, incluyendo configuración, configuración de IAM y solución de problemas.

Requisitos previos

Antes de configurar Claude Code con Vertex AI, asegúrese de tener:

- Una cuenta de Google Cloud Platform (GCP) con facturación habilitada
- Un proyecto de GCP con la API de Vertex AI habilitada
- Acceso a los modelos Claude deseados (por ejemplo, Claude Sonnet 4.5)
- Google Cloud SDK (`gccloud`) instalado y configurado
- Cuota asignada en la región de GCP deseada

Configuración de región

Claude Code se puede utilizar tanto con [puntos finales globales](#) como regionales de Vertex AI.

Note:

Vertex AI puede no admitir los modelos predeterminados de Claude Code en todas las regiones. Es posible que deba cambiar a una [región o modelo compatible](#).

Note:

Vertex AI puede no admitir los modelos predeterminados de Claude Code en puntos finales globales. Es posible que deba cambiar a un punto final regional o a un [modelo compatible](#).

Configuración

1. Habilitar la API de Vertex AI

Habilite la API de Vertex AI en su proyecto de GCP:

```
## Establezca su ID de proyecto
gcloud config set project YOUR-PROJECT-ID

## Habilitar la API de Vertex AI
gcloud services enable aiplatform.googleapis.com
```

2. Solicitar acceso al modelo

Solicite acceso a los modelos Claude en Vertex AI:

1. Navegue al [Vertex AI Model Garden](#)
2. Busque modelos “Claude”
3. Solicite acceso a los modelos Claude deseados (por ejemplo, Claude Sonnet 4.5)
4. Espere la aprobación (puede tomar 24-48 horas)

3. Configurar credenciales de GCP

Claude Code utiliza la autenticación estándar de Google Cloud.

Para obtener más información, consulte la [documentación de autenticación de Google Cloud](#).

Note:

Al autenticarse, Claude Code utilizará automáticamente el ID de proyecto de la variable de entorno `ANTHROPIC_VERTEX_PROJECT_ID`. Para anular esto, establezca una de estas variables de entorno: `GCPLOUD_PROJECT`, `GOOGLE_CLOUD_PROJECT` o `GOOGLE_APPLICATION_CREDENTIALS`.

4. Configurar Claude Code

Establezca las siguientes variables de entorno:

```
## Habilitar integración de Vertex AI
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## Opcional: Deshabilitar almacenamiento en caché de solicitudes si es necesario
export DISABLE_PROMPT_CACHING=1

## Cuando CLOUD_ML_REGION=global, anule la región para modelos no compatibles
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## Opcional: Anular regiones para otros modelos específicos
export VERTEX_REGION_CLAUDE_3_5_SONNET=us-east5
export VERTEX_REGION_CLAUDE_3_7_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_0_OPUS=europe-west1
export VERTEX_REGION_CLAUDE_4_0_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_1_OPUS=europe-west1
```

Note:

[Prompt caching](#) se admite automáticamente cuando especifica la bandera efímera `cache_control`. Para deshabilitarlo, establezca `DISABLE_PROMPT_CACHING=1`. Para límites de velocidad mejorados, póngase en contacto con el soporte de Google Cloud.

Note:

Al utilizar Vertex AI, los comandos `/login` y `/logout` están deshabilitados ya que la autenticación se maneja a través de credenciales de Google Cloud.

5. Configuración del modelo

Claude Code utiliza estos modelos predeterminados para Vertex AI:

Tipo de modelo	Valor predeterminado
Modelo principal	claude-sonnet-4-5@20250929
Modelo pequeño/rápido	claude-haiku-4-5@20251001

Note:

Para usuarios de Vertex AI, Claude Code no se actualizará automáticamente de Haiku 3.5 a Haiku 4.5. Para cambiar manualmente a un modelo Haiku más nuevo, establezca la variable de entorno `ANTHROPIC_DEFAULT_HAIKU_MODEL` en el nombre completo del modelo (por ejemplo, `claude-haiku-4-5@20251001`).

Para personalizar modelos:

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'  
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

Configuración de IAM

Asigne los permisos de IAM requeridos:

El rol `roles/aiplatform.user` incluye los permisos requeridos:

- `aiplatform.endpoints.predict` - Requerido para la invocación de modelos y conteo de tokens

Para permisos más restrictivos, cree un rol personalizado solo con los permisos anteriores.

Para obtener más detalles, consulte la [documentación de IAM de Vertex](#).

Note:

Recomendamos crear un proyecto de GCP dedicado para Claude Code para simplificar el seguimiento de costos y el control de acceso.

Ventana de contexto de 1M de tokens

Claude Sonnet 4 y Sonnet 4.5 admiten la [ventana de contexto de 1M de tokens](#) en Vertex AI.

Note:

La ventana de contexto de 1M de tokens está actualmente en versión beta. Para utilizar la ventana de contexto extendida, incluya el encabezado beta `context-1m-2025-08-07` en sus solicitudes de Vertex AI.

Solución de problemas

Si encuentra problemas de cuota:

- Verifique las cuotas actuales o solicite un aumento de cuota a través de [Cloud Console](#)

Si encuentra errores 404 “modelo no encontrado”:

- Confirme que el modelo está habilitado en [Model Garden](#)
- Verifique que tenga acceso a la región especificada
- Si utiliza `CLOUD_ML_REGION=globa1`, verifique que sus modelos admitan puntos finales globales en [Model Garden](#) en “Características compatibles”. Para modelos que no admiten puntos finales globales, ya sea:
 - Especifique un modelo compatible a través de `ANTHROPIC_MODEL` o `ANTHROPIC_SMALL_FAST_MODEL`, o
 - Establezca un punto final regional usando variables de entorno `VERTEX_REGION_<MODEL_NAME>`

Si encuentra errores 429:

- Para puntos finales regionales, asegúrese de que el modelo principal y el modelo pequeño/rápido sean compatibles en su región seleccionada
- Considere cambiar a `CLOUD_ML_REGION=globa1` para una mejor disponibilidad

Recursos adicionales

- [Documentación de Vertex AI](#)
- [Precios de Vertex AI](#)
- [Cuotas y límites de Vertex AI](#)

Claude Code en Microsoft Foundry

Aprende a configurar Claude Code a través de Microsoft Foundry, incluyendo configuración, instalación y solución de problemas.

Requisitos previos

Antes de configurar Claude Code con Microsoft Foundry, asegúrate de que tienes:

- Una suscripción de Azure con acceso a Microsoft Foundry
- Permisos RBAC para crear recursos e implementaciones de Microsoft Foundry
- Azure CLI instalado y configurado (opcional - solo necesario si no tienes otro mecanismo para obtener credenciales)

Configuración

1. Aprovisionar recurso de Microsoft Foundry

Primero, crea un recurso de Claude en Azure:

1. Navega al [portal de Microsoft Foundry](#)
2. Crea un nuevo recurso, anotando el nombre de tu recurso
3. Crea implementaciones para los modelos de Claude:
 - Claude Opus
 - Claude Sonnet
 - Claude Haiku

2. Configurar credenciales de Azure

Claude Code admite dos métodos de autenticación para Microsoft Foundry. Elige el método que mejor se ajuste a tus requisitos de seguridad.

Opción A: Autenticación por clave API

1. Navega a tu recurso en el portal de Microsoft Foundry
2. Ve a la sección **Endpoints and keys** (Puntos finales y claves)
3. Copia **API Key** (Clave API)
4. Establece la variable de entorno:

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

Opción B: Autenticación de Microsoft Entra ID

Cuando `ANTHROPIC_FOUNDRY_API_KEY` no está configurado, Claude Code utiliza automáticamente la [cadena de credenciales predeterminada](#) del SDK de Azure. Esto admite una variedad de métodos para autenticar cargas de trabajo locales y remotas.

En entornos locales, comúnmente puedes usar Azure CLI:

```
az login
```

Note:

Cuando se usa Microsoft Foundry, los comandos `/login` y `/logout` están deshabilitados ya que la autenticación se maneja a través de credenciales de Azure.

3. Configurar Claude Code

Establece las siguientes variables de entorno para habilitar Microsoft Foundry. Ten en cuenta que los nombres de tus implementaciones se establecen como identificadores de modelo en Claude Code (puede ser opcional si usas nombres de implementación sugeridos).

```
## Enable Microsoft Foundry integration
export CLAUDE_CODE_USE_FOUNDRY=1

## Azure resource name (replace {resource} with your resource name)
export ANTHROPIC_FOUNDRY_RESOURCE={resource}
## Or provide the full base URL:
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## Set models to your resource's deployment names
export ANTHROPIC_DEFAULT_SONNET_MODEL='claude-sonnet-4-5'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5'
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-1'
```

Para más detalles sobre las opciones de configuración de modelos, consulta [Configuración de modelos](#).

Configuración de RBAC de Azure

Los roles predeterminados `Azure AI User` y `Cognitive Services User` incluyen todos los permisos necesarios para invocar modelos de Claude.

Para permisos más restrictivos, crea un rol personalizado con lo siguiente:

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

Para más detalles, consulta la [documentación de RBAC de Microsoft Foundry](#).

Solución de problemas

Si recibes un error “Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed”:

- Configura Entra ID en el entorno, o establece `ANTHROPIC_FOUNDRY_API_KEY`.

Recursos adicionales

- [Documentación de Microsoft Foundry](#)
- [Modelos de Microsoft Foundry](#)
- [Precios de Microsoft Foundry](#)

Configuración de la puerta de enlace LLM

Aprende cómo configurar Claude Code para trabajar con soluciones de puerta de enlace LLM. Cubre requisitos de puerta de enlace, configuración de autenticación, selección de modelos y configuración de puntos finales específicos del proveedor.

Las puertas de enlace LLM proporcionan una capa proxy centralizada entre Claude Code y los proveedores de modelos, a menudo proporcionando:

- **Autenticación centralizada** - Punto único para la gestión de claves API
- **Seguimiento de uso** - Monitorea el uso en equipos y proyectos
- **Controles de costos** - Implementa presupuestos y límites de velocidad
- **Registro de auditoría** - Rastrea todas las interacciones del modelo para cumplimiento normativo
- **Enrutamiento de modelos** - Cambia entre proveedores sin cambios de código

Requisitos de la puerta de enlace

Para que una puerta de enlace LLM funcione con Claude Code, debe cumplir con los siguientes requisitos:

Formato de API

La puerta de enlace debe exponer a los clientes al menos uno de los siguientes formatos de API:

1. **Anthropic Messages:** `/v1/messages` , `/v1/messages/count_tokens`
 - Debe reenviar encabezados de solicitud: `anthropic-beta` , `anthropic-version`
2. **Bedrock InvokeModel:** `/invoke` , `/invoke-with-response-stream`
 - Debe preservar campos del cuerpo de la solicitud: `anthropic_beta` , `anthropic_version`
3. **Vertex rawPredict:** `:rawPredict` , `:streamRawPredict` , `/count-tokens:rawPredict`
 - Debe reenviar encabezados de solicitud: `anthropic-beta` , `anthropic-version`

El incumplimiento de reenvío de encabezados o la preservación de campos del cuerpo puede resultar en funcionalidad reducida o incapacidad de usar características de Claude Code.

Note:

Claude Code determina qué características habilitar en función del formato de API. Al usar el formato Anthropic Messages con Bedrock o Vertex, es posible que necesites establecer la variable de entorno `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1`.

Configuración

Selección de modelo

Por defecto, Claude Code utilizará nombres de modelo estándar para el formato de API seleccionado.

Si has configurado nombres de modelo personalizados en tu puerta de enlace, utiliza las variables de entorno documentadas en [Configuración de modelo](#) para que coincidan con tus nombres personalizados.

Configuración de LiteLLM

Note:

LiteLLM es un servicio proxy de terceros. Anthropic no respalda, mantiene ni audita la seguridad o funcionalidad de LiteLLM. Esta guía se proporciona con fines informativos y puede quedar obsoleta. Úsala bajo tu propio criterio.

Requisitos previos

- Claude Code actualizado a la última versión
- Servidor Proxy de LiteLLM implementado y accesible
- Acceso a modelos Claude a través de tu proveedor elegido

Configuración básica de LiteLLM

Configura Claude Code:

Métodos de autenticación

Clave API estática

Método más simple usando una clave API fija:

```
## Establecer en el entorno
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## 0 en la configuración de Claude Code
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

Este valor se enviará como encabezado `Authorization`.

Clave API dinámica con ayudante

Para claves rotativas o autenticación por usuario:

1. Crea un script ayudante de clave API:

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## Ejemplo: Obtener clave del almacén
vault kv get -field=api_key secret/litellm/claude-code

## Ejemplo: Generar token JWT
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. Configura la configuración de Claude Code para usar el ayudante:

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. Establece el intervalo de actualización de token:

```
## Actualizar cada hora (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

Este valor se enviará como encabezados `Authorization` y `X-API-Key`. El `apiKeyHelper` tiene menor precedencia que `ANTHROPIC_AUTH_TOKEN` o `ANTHROPIC_API_KEY`.

Punto final unificado (recomendado)

Usando el [punto final de formato Anthropic](#) de LiteLLM:

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

Beneficios del punto final unificado sobre puntos finales de paso directo:

- Equilibrio de carga
- Alternativas
- Soporte consistente para seguimiento de costos y seguimiento de usuario final

Puntos finales de paso directo específicos del proveedor (alternativa)

API de Claude a través de LiteLLM

Usando [punto final de paso directo](#):

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

Amazon Bedrock a través de LiteLLM

Usando [punto final de paso directo](#):

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

Google Vertex AI a través de LiteLLM

Usando [punto final de paso directo](#):

```
export ANTHROPIC_VERTEX_BASE_URL=https://litellm-server:4000/vertex_ai/v1
export ANTHROPIC_VERTEX_PROJECT_ID=your-gcp-project-id
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
```

Para obtener información más detallada, consulta la [documentación de LiteLLM](#).

Recursos adicionales

- [Documentación de LiteLLM](#)
- [Configuración de Claude Code](#)
- [Configuración de red empresarial](#)
- [Descripción general de integraciones de terceros](#)

Configuración de red empresarial

Configure Claude Code para entornos empresariales con servidores proxy, Autoridades de Certificación (CA) personalizadas y autenticación mutua de Seguridad de la Capa de Transporte (mTLS).

Claude Code admite varias configuraciones de red y seguridad empresarial a través de variables de entorno. Esto incluye enrutar el tráfico a través de servidores proxy corporativos, confiar en Autoridades de Certificación (CA) personalizadas y autenticarse con certificados de Seguridad de la Capa de Transporte mutua (mTLS) para mayor seguridad.

Note:

Todas las variables de entorno que se muestran en esta página también se pueden configurar en `settings.json`.

Configuración de proxy

Variables de entorno

Claude Code respeta las variables de entorno proxy estándar:

```
## Proxy HTTPS (recomendado)
export HTTPS_PROXY=https://proxy.example.com:8080

## Proxy HTTP (si HTTPS no está disponible)
export HTTP_PROXY=http://proxy.example.com:8080

## Omitir proxy para solicitudes específicas - formato separado por espacios
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"

## Omitir proxy para solicitudes específicas - formato separado por comas
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"

## Omitir proxy para todas las solicitudes
export NO_PROXY="*"

```

Note:

Claude Code no admite proxies SOCKS.

Autenticación básica

Si su proxy requiere autenticación básica, incluya las credenciales en la URL del proxy:

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

Warning:

Evite codificar contraseñas en scripts. Utilice variables de entorno o almacenamiento seguro de credenciales en su lugar.

Tip:

Para proxies que requieren autenticación avanzada (NTLM, Kerberos, etc.), considere utilizar un servicio LLM Gateway que admita su método de autenticación.

Certificados CA personalizados

Si su entorno empresarial utiliza CA personalizadas para conexiones HTTPS (ya sea a través de un proxy o acceso directo a la API), configure Claude Code para confiar en ellas:

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

Autenticación mTLS

Para entornos empresariales que requieren autenticación de certificado de cliente:

```
## Certificado de cliente para autenticación
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## Clave privada del cliente
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## Opcional: Frase de contraseña para clave privada cifrada
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

Requisitos de acceso a la red

Claude Code requiere acceso a las siguientes URL:

- `api.anthropic.com` : Puntos finales de la API de Claude
- `claude.ai` : autenticación para cuentas de claude.ai
- `platform.claude.com` : autenticación para cuentas de Anthropic Console

Asegúrese de que estas URL estén en la lista blanca en su configuración de proxy y reglas de firewall. Esto es especialmente importante cuando se utiliza Claude Code en entornos de red restringidos o en contenedores.

Recursos adicionales

- [Configuración de Claude Code](#)
- [Referencia de variables de entorno](#)
- [Guía de solución de problemas](#)

Part 12: Environment Setup

Contenedores de desarrollo

Aprende sobre el contenedor de desarrollo de Claude Code para equipos que necesitan entornos consistentes y seguros.

La [configuración de devcontainer](#) de referencia y el [Dockerfile](#) asociado ofrecen un contenedor de desarrollo preconfigurado que puedes usar tal como está, o personalizar según tus necesidades. Este devcontainer funciona con la extensión [Dev Containers](#) de Visual Studio Code y herramientas similares.

Las medidas de seguridad mejoradas del contenedor (aislamiento y reglas de firewall) te permiten ejecutar `claude --dangerously-skip-permissions` para omitir solicitudes de permisos para operación desatendida.

Warning:

Aunque el devcontainer proporciona protecciones sustanciales, ningún sistema es completamente inmune a todos los ataques. Cuando se ejecuta con `--dangerously-skip-permissions`, los devcontainers no previenen que un proyecto malicioso exfiltre cualquier cosa accesible en el devcontainer, incluyendo credenciales de Claude Code. Recomendamos usar devcontainers solo cuando se desarrolla con repositorios de confianza. Siempre mantén buenas prácticas de seguridad y monitorea las actividades de Claude.

Características clave

- **Node.js listo para producción:** Construido sobre Node.js 20 con dependencias de desarrollo esenciales
- **Seguridad por diseño:** Firewall personalizado que restringe el acceso de red solo a servicios necesarios
- **Herramientas amigables para desarrolladores:** Incluye git, ZSH con mejoras de productividad, fzf y más
- **Integración perfecta con VS Code:** Extensiones preconfiguradas y configuraciones optimizadas
- **Persistencia de sesión:** Preserva el historial de comandos y configuraciones entre reinicios del contenedor

- **Funciona en todas partes:** Compatible con entornos de desarrollo en macOS, Windows y Linux

Comenzar en 4 pasos

1. Instala VS Code y la extensión Remote - Containers
2. Clona el repositorio de [implementación de referencia de Claude Code](#)
3. Abre el repositorio en VS Code
4. Cuando se te solicite, haz clic en “Reopen in Container” (o usa Command Palette: Cmd+Shift+P → “Remote-Containers: Reopen in Container”)

Desglose de configuración

La configuración de devcontainer consta de tres componentes principales:

- **[devcontainer.json](#):** Controla la configuración del contenedor, extensiones y montajes de volumen
- **[Dockerfile](#):** Define la imagen del contenedor y las herramientas instaladas
- **[init-firewall.sh](#):** Establece las reglas de seguridad de red

Características de seguridad

El contenedor implementa un enfoque de seguridad multicapa con su configuración de firewall:

- **Control de acceso preciso:** Restringe las conexiones salientes solo a dominios en la lista blanca (registro npm, GitHub, API de Claude, etc.)
- **Conexiones salientes permitidas:** El firewall permite conexiones DNS y SSH salientes
- **Política de negación por defecto:** Bloquea todo otro acceso de red externo
- **Verificación de inicio:** Valida las reglas del firewall cuando se inicializa el contenedor
- **Aislamiento:** Crea un entorno de desarrollo seguro separado de tu sistema principal

Opciones de personalización

La configuración de devcontainer está diseñada para ser adaptable a tus necesidades:

- Añade o elimina extensiones de VS Code según tu flujo de trabajo
- Modifica asignaciones de recursos para diferentes entornos de hardware
- Ajusta permisos de acceso de red

- Personaliza configuraciones de shell y herramientas de desarrollador

Casos de uso de ejemplo

Trabajo seguro con clientes

Usa devcontainers para aislar diferentes proyectos de clientes, asegurando que el código y las credenciales nunca se mezclen entre entornos.

Incorporación de equipo

Los nuevos miembros del equipo pueden obtener un entorno de desarrollo completamente configurado en minutos, con todas las herramientas y configuraciones necesarias preinstaladas.

Entornos CI/CD consistentes

Refleja tu configuración de devcontainer en canalizaciones CI/CD para asegurar que los entornos de desarrollo y producción coincidan.

Recursos relacionados

- [Documentación de devcontainers de VS Code](#)
- [Mejores prácticas de seguridad de Claude Code](#)
- [Configuración de red empresarial](#)

Optimiza tu configuración de terminal

Claude Code funciona mejor cuando tu terminal está correctamente configurada. Sigue estas directrices para optimizar tu experiencia.

Temas y apariencia

Claude no puede controlar el tema de tu terminal. Eso se maneja por tu aplicación de terminal. Puedes hacer coincidir el tema de Claude Code con tu terminal en cualquier momento a través del comando `/config`.

Para personalización adicional de la interfaz de Claude Code en sí, puedes configurar una [línea de estado personalizada](#) para mostrar información contextual como el modelo actual, directorio de trabajo o rama de git en la parte inferior de tu terminal.

Saltos de línea

Tienes varias opciones para ingresar saltos de línea en Claude Code:

- **Escape rápido:** Escribe `\` seguido de Enter para crear una nueva línea
- **Shift+Enter:** Funciona de forma nativa en iTerm2, WezTerm, Ghostty y Kitty
- **Atajo de teclado:** Configura un enlace de tecla para insertar una nueva línea en otras terminales

Configura Shift+Enter para otras terminales

Ejecuta `/terminal-setup` dentro de Claude Code para configurar automáticamente Shift+Enter para VS Code, Alacritty, Zed y Warp.

Note:

El comando `/terminal-setup` solo es visible en terminales que requieren configuración manual. Si estás usando iTerm2, WezTerm, Ghostty o Kitty, no verás este comando porque Shift+Enter ya funciona de forma nativa.

Configura Option+Enter (VS Code, iTerm2 o macOS Terminal.app)

Para Mac Terminal.app:

1. Abre Configuración → Perfiles → Teclado
2. Marca “Usar Opción como tecla Meta”

Para iTerm2 y terminal de VS Code:

1. Abre Configuración → Perfiles → Teclas
2. En General, establece la tecla Opción Izquierda/Derecha en “Esc+”

Configuración de notificaciones

Nunca te pierdas cuando Claude completa una tarea con la configuración adecuada de notificaciones:

Notificaciones del sistema iTerm 2

Para alertas de iTerm 2 cuando se completan tareas:

1. Abre Preferencias de iTerm 2
2. Navega a Perfiles → Terminal
3. Habilita “Silenciar campana” y Filtrar Alertas → “Enviar alertas generadas por secuencia de escape”
4. Establece tu retraso de notificación preferido

Ten en cuenta que estas notificaciones son específicas de iTerm 2 y no están disponibles en la Terminal predeterminada de macOS.

Ganchos de notificación personalizados

Para manejo avanzado de notificaciones, puedes crear [ganchos de notificación](#) para ejecutar tu propia lógica.

Manejo de entradas grandes

Cuando trabajes con código extenso o instrucciones largas:

- **Evita pegar directamente:** Claude Code puede tener dificultades con contenido pegado muy largo
- **Usa flujos de trabajo basados en archivos:** Escribe contenido en un archivo y pide a Claude que lo lea
- **Ten en cuenta las limitaciones de VS Code:** La terminal de VS Code es particularmente propensa a truncar pegados largos

Modo Vim

Claude Code admite un subconjunto de enlaces de teclas Vim que se pueden habilitar con `/vim` o configurar a través de `/config`.

El subconjunto admitido incluye:

- Cambio de modo: **Esc** (a NORMAL), **i / I**, **a / A**, **o / O** (a INSERT)
- Navegación: **h / j / k / l**, **w / e / b**, **0 / \$ / ^**, **gg / G**, **f / F / t / T** con repetición **;** / **,**
- Edición: **x**, **dw / de / db / dd / D**, **cw / ce / cb / cc / C**, **.** (repetir)
- Copiar/pegar: **yy / Y**, **yw / ye / yb**, **p / P**
- Objetos de texto: **iw / aw**, **iW / aW**, **i" / a"**, **i' / a'**, **i(/ a(**, **i[/ a[**, **i{ / a{**
- Indentación: **>> / <<**
- Operaciones de línea: **J** (unir líneas)

Consulta [Modo interactivo](#) para la referencia completa.

Personaliza tu línea de estado

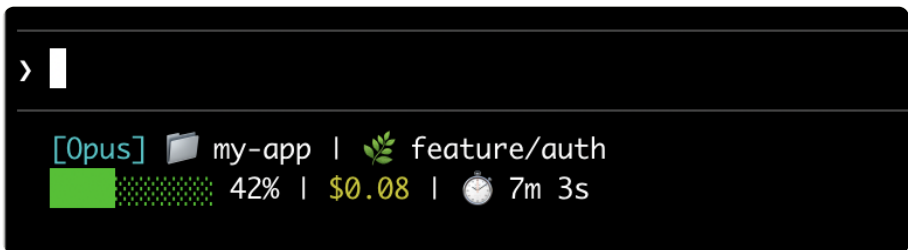
Configura una barra de estado personalizada para monitorear el uso de la ventana de contexto, costos y estado de git en Claude Code

La línea de estado es una barra personalizable en la parte inferior de Claude Code que ejecuta cualquier script de shell que configures. Recibe datos de sesión JSON en stdin y muestra lo que tu script imprime, dándote una vista persistente y de un vistazo del uso de contexto, costos, estado de git, o cualquier otra cosa que desees rastrear.

Las líneas de estado son útiles cuando:

- Deseas monitorear el uso de la ventana de contexto mientras trabajas
- Necesitas rastrear los costos de la sesión
- Trabajas en múltiples sesiones y necesitas distinguirlas
- Deseas que la rama de git y el estado siempre sean visibles

Aquí hay un ejemplo de una [línea de estado de múltiples líneas](#) que muestra información de git en la primera línea y una barra de contexto codificada por colores en la segunda.



Una línea de estado de múltiples líneas que muestra el nombre del modelo, directorio, rama de git en la primera línea, y una barra de progreso de uso de contexto con costo y duración en la segunda línea

Esta página te guía a través de [configurar una línea de estado básica](#), explica [cómo fluyen los datos](#) desde Claude Code a tu script, enumera [todos los campos que puedes mostrar](#), y proporciona [ejemplos listos para usar](#) para patrones comunes como estado de git, seguimiento de costos y barras de progreso.

Configura una línea de estado

Usa el [comando](#) `/statusline` para que Claude Code genere un script para ti, o [crea manualmente un script](#) y agrégalo a tu configuración.

Usa el comando `/statusline`

El comando `/statusline` acepta instrucciones en lenguaje natural que describen lo que deseas mostrar. Claude Code genera un archivo de script en `~/.claude/` y actualiza tu configuración automáticamente:

```
/statusline show model name and context percentage with a progress bar
```

Configura manualmente una línea de estado

Agrega un campo `statusLine` a tu configuración de usuario (`~/.claude/settings.json`, donde `~` es tu directorio de inicio) o [configuración del proyecto](#). Establece `type` en `"command"` y apunta `command` a una ruta de script o un comando de shell en línea. Para un tutorial completo sobre cómo crear un script, consulta [Construye una línea de estado paso a paso](#).

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

El campo `command` se ejecuta en un shell, por lo que también puedes usar comandos en línea en lugar de un archivo de script. Este ejemplo usa `jq` para analizar la entrada JSON y mostrar el nombre del modelo y el porcentaje de contexto:

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\"[\\(\\.model\\.display_name)] \\(\\.context_window\\.used_percent  
age // 0)% context\\\"'"
  }
}
```

El campo `padding` opcional agrega espaciado horizontal adicional (en caracteres) al contenido de la línea de estado. Por defecto es `0`. Este relleno se suma al espaciado integrado de la interfaz, por lo que controla la indentación relativa en lugar de la distancia absoluta desde el borde de la terminal.

Desactiva la línea de estado

Ejecuta `/statusline` y pídele que elimine o borre tu línea de estado (por ejemplo, `/statusline delete`, `/statusline clear`, `/statusline remove it`). También puedes eliminar manualmente el campo `statusLine` de tu `settings.json`.

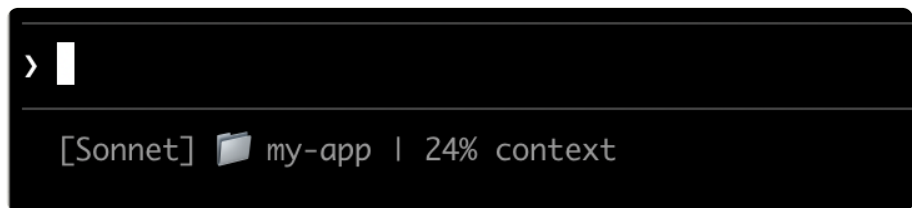
Construye una línea de estado paso a paso

Este tutorial muestra lo que está sucediendo bajo el capó creando manualmente una línea de estado que muestra el modelo actual, el directorio de trabajo y el porcentaje de uso de la ventana de contexto.

Note:

Ejecutar `/statusline` con una descripción de lo que deseas configura todo esto automáticamente para ti.

Estos ejemplos usan scripts de Bash, que funcionan en macOS y Linux. En Windows, consulta [Configuración de Windows](#) para ejemplos de PowerShell y Git Bash.



Una línea de estado que muestra el nombre del modelo, directorio y porcentaje de contexto

Step 1: Crea un script que lea JSON e imprima salida

Claude Code envía datos JSON a tu script a través de stdin. Este script usa `jq`, un analizador JSON de línea de comandos que es posible que necesites instalar, para extraer el nombre del modelo, el directorio y el porcentaje de contexto, luego imprime una línea formateada.

Guarda esto en `~/.claude/statusline.sh` (donde `~` es tu directorio de inicio, como `/Users/username` en macOS o `/home/username` en Linux):

```
#!/bin/bash

## Read JSON data that Claude Code sends to stdin
input=$(cat)

## Extract fields using jq
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## The "// 0" provides a fallback if the field is null
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Output the status line - ${DIR##*/} extracts just the folder name
echo "[${MODEL}] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: Hazlo ejecutable

Marca el script como ejecutable para que tu shell pueda ejecutarlo:

```
chmod +x ~/.claude/statusline.sh
```

Step 3: Agrega a la configuración

Dile a Claude Code que ejecute tu script como la línea de estado. Agrega esta configuración a `~/.claude/settings.json`, que establece `type` en `"command"` (lo que significa “ejecuta este comando de shell”) y apunta `command` a tu script:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

Tu línea de estado aparece en la parte inferior de la interfaz. La configuración se recarga automáticamente, pero los cambios no aparecerán hasta tu próxima interacción con Claude Code.

Cómo funcionan las líneas de estado

Claude Code ejecuta tu script y canaliza [datos de sesión JSON](#) a través de stdin. Tu script lee el JSON, extrae lo que necesita e imprime texto a stdout. Claude Code muestra lo que tu script imprime.

Cuándo se actualiza

Tu script se ejecuta después de cada nuevo mensaje del asistente, cuando cambia el modo de permiso, o cuando se activa/desactiva el modo vim. Las actualizaciones se debounce en 300ms, lo que significa que los cambios rápidos se agrupan y tu script se ejecuta una vez que las cosas se estabilizan. Si una nueva actualización se activa mientras tu script aún se está ejecutando, la ejecución en vuelo se cancela. Si editas tu script, los cambios no aparecerán hasta que tu próxima interacción con Claude Code active una actualización.

Lo que tu script puede generar

- **Múltiples líneas:** cada declaración `echo` o `print` se muestra como una fila separada. Consulta el [ejemplo de múltiples líneas](#).
- **Colores:** usa [códigos de escape ANSI](#) como `\033[32m` para verde (la terminal debe admitirlos). Consulta el [ejemplo de estado de git](#).
- **Enlaces:** usa [secuencias de escape OSC 8](#) para hacer que el texto sea clickeable (Cmd+clíc en macOS, Ctrl+clíc en Windows/Linux). Requiere una terminal que admita hipervínculos como iTerm2, Kitty o WezTerm. Consulta el [ejemplo de enlaces clickeables](#).

Note:

La línea de estado se ejecuta localmente y no consume tokens de API. Se oculta temporalmente durante ciertas interacciones de la interfaz, incluidas sugerencias de autocompletado, el menú de ayuda y solicitudes de permiso.

Datos disponibles

Claude Code envía los siguientes campos JSON a tu script a través de stdin:

Campo	Descripción
<code>model.id</code> , <code>model.display_name</code>	Identificador del modelo actual y nombre para mostrar

Campo	Descripción
<code>cwd</code> , <code>workspace.current_dir</code>	Directorio de trabajo actual. Ambos campos contienen el mismo valor; <code>workspace.current_dir</code> es preferido para consistencia con <code>workspace.project_dir</code> .
<code>workspace.project_dir</code>	Directorio donde se lanzó Claude Code, que puede diferir de <code>cwd</code> si el directorio de trabajo cambia durante una sesión
<code>cost.total_cost_usd</code>	Costo total de la sesión en USD
<code>cost.total_duration_ms</code>	Tiempo total transcurrido desde que comenzó la sesión, en milisegundos
<code>cost.total_api_duration_ms</code>	Tiempo total dedicado a esperar respuestas de API en milisegundos
<code>cost.total_lines_added</code> , <code>cost.total_lines_removed</code>	Líneas de código cambiadas
<code>context_window.total_input_tokens</code> , <code>context_window.total_output_tokens</code>	Conteos de tokens acumulativos en toda la sesión
<code>context_window.context_window_size</code>	Tamaño máximo de la ventana de contexto en tokens. 200000 por defecto, o 1000000 para modelos con contexto extendido.
<code>context_window.used_percentage</code>	Porcentaje precalculado de ventana de contexto utilizada
<code>context_window.remaining_percentage</code>	Porcentaje precalculado de ventana de contexto restante
<code>context_window.current_usage</code>	Conteos de tokens de la última llamada a API, descritos en campos de ventana de contexto
<code>exceeds_200k_tokens</code>	Si el conteo total de tokens (tokens de entrada, caché y salida combinados) de la respuesta de API más reciente excede 200k. Este es un umbral fijo independientemente del tamaño real de la ventana de contexto.
<code>session_id</code>	Identificador único de sesión

Campo	Descripción
<code>transcript_path</code>	Ruta al archivo de transcripción de conversación
<code>version</code>	Versión de Claude Code
<code>output_style.name</code>	Nombre del estilo de salida actual
<code>vim.mode</code>	Modo vim actual (<code>NORMAL</code> o <code>INSERT</code>) cuando el modo vim está habilitado
<code>agent.name</code>	Nombre del agente cuando se ejecuta con la bandera <code>--agent</code> o configuración de agente configurada
<code>worktree.name</code>	Nombre del worktree activo. Presente solo durante sesiones <code>--worktree</code>
<code>worktree.path</code>	Ruta absoluta al directorio del worktree
<code>worktree.branch</code>	Nombre de rama de Git para el worktree (por ejemplo, <code>"worktree-my-feature"</code>). Ausente para worktrees basados en hooks
<code>worktree.original_cwd</code>	El directorio en el que estaba Claude antes de entrar en el worktree
<code>worktree.original_branch</code>	Rama de Git extraída antes de entrar en el worktree. Ausente para worktrees basados en hooks

Tu comando de línea de estado recibe esta estructura JSON a través de stdin:


```

{
  "cwd": "/current/working/directory",
  "session_id": "abc123...",
  "transcript_path": "/path/to/transcript.jsonl",
  "model": {
    "id": "claude-opus-4-6",
    "display_name": "Opus"
  },
  "workspace": {
    "current_dir": "/current/working/directory",
    "project_dir": "/original/project/directory"
  },
  "version": "1.0.80",
  "output_style": {
    "name": "default"
  },
  "cost": {
    "total_cost_usd": 0.01234,
    "total_duration_ms": 45000,
    "total_api_duration_ms": 2300,
    "total_lines_added": 156,
    "total_lines_removed": 23
  },
  "context_window": {
    "total_input_tokens": 15234,
    "total_output_tokens": 4521,
    "context_window_size": 200000,
    "used_percentage": 8,
    "remaining_percentage": 92,
    "current_usage": {
      "input_tokens": 8500,
      "output_tokens": 1200,
      "cache_creation_input_tokens": 5000,
      "cache_read_input_tokens": 2000
    }
  },
  "exceeds_200k_tokens": false,
  "vim": {

```

```

    "mode": "NORMAL"
  },
  "agent": {
    "name": "security-reviewer"
  },
  "worktree": {
    "name": "my-feature",
    "path": "/path/to/.claude/worktrees/my-feature",
    "branch": "worktree-my-feature",
    "original_cwd": "/path/to/project",
    "original_branch": "main"
  }
}

```

Campos que pueden estar ausentes (no presentes en JSON):

- `vim` : aparece solo cuando el modo vim está habilitado
- `agent` : aparece solo cuando se ejecuta con la bandera `--agent` o configuración de agente configurada
- `worktree` : aparece solo durante sesiones `--worktree`. Cuando está presente, `branch` y `original_branch` también pueden estar ausentes para worktrees basados en hooks

Campos que pueden ser `null` :

- `context_window.current_usage` : `null` antes de la primera llamada a API en una sesión
- `context_window.used_percentage` , `context_window.remaining_percentage` : pueden ser `null` al principio de la sesión

Maneja campos faltantes con acceso condicional y valores nulos con valores predeterminados de respaldo en tus scripts.

Campos de ventana de contexto

El objeto `context_window` proporciona dos formas de rastrear el uso de contexto:

- **Totales acumulativos** (`total_input_tokens` , `total_output_tokens`): suma de todos los tokens en toda la sesión, útil para rastrear el consumo total
- **Uso actual** (`current_usage`): conteos de tokens de la llamada a API más reciente, úsalo para un porcentaje de contexto preciso ya que refleja el estado real del contexto

El objeto `current_usage` contiene:

- `input_tokens` : tokens de entrada en contexto actual
- `output_tokens` : tokens de salida generados
- `cache_creation_input_tokens` : tokens escritos en caché
- `cache_read_input_tokens` : tokens leídos del caché

El campo `used_percentage` se calcula solo a partir de tokens de entrada: `input_tokens + cache_creation_input_tokens + cache_read_input_tokens` . No incluye `output_tokens` .

Si calculas el porcentaje de contexto manualmente desde `current_usage` , usa la misma fórmula de solo entrada para coincidir con `used_percentage` .

El objeto `current_usage` es `null` antes de la primera llamada a API en una sesión.

Ejemplos

Estos ejemplos muestran patrones comunes de línea de estado. Para usar cualquier ejemplo:

1. Guarda el script en un archivo como `~/.claude/statusline.sh` (o `.py` / `.js`)
2. Hazlo ejecutable: `chmod +x ~/.claude/statusline.sh`
3. Agrega la ruta a tu [configuración](#)

Los ejemplos de Bash usan `jq` para analizar JSON. Python y Node.js tienen análisis JSON integrado.

Uso de ventana de contexto

Muestra el modelo actual y el uso de la ventana de contexto con una barra de progreso visual. Cada script lee JSON desde stdin, extrae el campo `used_percentage` y construye una barra de 10 caracteres donde los bloques rellenos (█) representan el uso:



Una línea de estado que muestra el nombre del modelo y una barra de progreso con porcentaje

```
#!/bin/bash

## Read all of stdin into a variable
input=$(cat)

## Extract fields with jq, "// 0" provides fallback for null
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Build progress bar: printf creates spaces, tr replaces with blocks
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""

[ "$FILLED" -gt 0 ] && BAR=$(printf "%${FILLED}s" | tr ' ' '█')
[ "$EMPTY" -gt 0 ] && BAR="${BAR}$(printf "%${EMPTY}s" | tr ' ' '░')"

echo "[${MODEL}] $BAR ${PCT%}"
```

```
#!/usr/bin/env python3
import json, sys

## json.load reads and parses stdin in one step
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0" handles null values
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## String multiplication builds the bar
filled = pct * 10 // 100
bar = '█' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

```
#!/usr/bin/env node
// Node.js reads stdin asynchronously with events
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  // Optional chaining (?) safely handles null fields
  const pct = Math.floor(data.context_window?.used_percentage || 0);

  // String.repeat() builds the bar
  const filled = Math.floor(pct * 10 / 100);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

  console.log(`[${model}] ${bar} ${pct}%`);
});
```

Estado de git con colores

Muestra la rama de git con indicadores codificados por colores para archivos preparados y modificados. Este script usa [códigos de escape ANSI](#) para colores de terminal: `\033[32m` es verde, `\033[33m` es amarillo, y `\033[0m` se reinicia al predeterminado.

```
> █
```

```
[Sonnet] 📁 my-app | 🌿 feature/auth +3~2
```

Una línea de estado que muestra modelo, directorio, rama de git e indicadores codificados por colores para archivos preparados y modificados

Cada script verifica si el directorio actual es un repositorio de git, cuenta archivos preparados y modificados, y muestra indicadores codificados por colores:

```
#!/bin/bash

input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

GREEN='\033[32m'
YELLOW='\033[33m'
RESET='\033[0m'

if git rev-parse --git-dir > /dev/null 2>&1; then
    BRANCH=$(git branch --show-current 2>/dev/null)
    STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
    MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')

    GIT_STATUS=""
    [ "$STAGED" -gt 0 ] && GIT_STATUS="${GREEN}+${STAGED}${RESET}"
    [ "$MODIFIED" -gt 0 ] && GIT_STATUS="${GIT_STATUS}${YELLOW}~${MODIFIED}${RESET}"

    echo -e "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH $GIT_STATUS"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

GREEN, YELLOW, RESET = '\033[32m', '\033[33m', '\033[0m'

try:
    subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.D
EVNULL)
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=Tru
e).strip()
    staged_output = subprocess.check_output(['git', 'diff', '--cached', '--
numstat'], text=True).strip()
    modified_output = subprocess.check_output(['git', 'diff', '--numstat'], text=T
rue).strip()
    staged = len(staged_output.split('\n')) if staged_output else 0
    modified = len(modified_output.split('\n')) if modified_output else 0

    git_status = f"{GREEN}+{staged}{RESET}" if staged else ""
    git_status += f"{YELLOW}~{modified}{RESET}" if modified else ""

    print(f"[{model}] 📁 {directory} | 🌿 {branch} {git_status}")
except:
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);

  const GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RESET = '\x1b[0m';

  try {
    execSync('git rev-parse --git-dir', { stdio: 'ignore' });
    const branch = execSync('git branch --show-current', { encoding:
'utf8' }).trim();
    const staged = execSync('git diff --cached --numstat', { encoding: 'utf8'
}).trim().split('\n').filter(Boolean).length;
    const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;

    let gitStatus = staged ? `${GREEN}+${staged}${RESET}` : '';
    gitStatus += modified ? `${YELLOW}~${modified}${RESET}` : '';

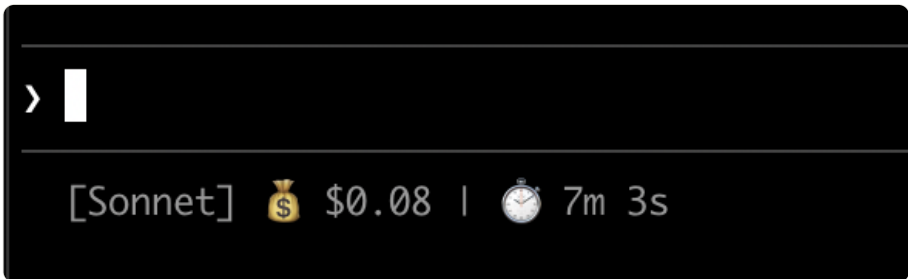
    console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} ${gitStatus}`);
  } catch {
    console.log(`[${model}] 📁 ${dir}`);
  }
});
```

Seguimiento de costos y duración

Rastrea los costos de API de tu sesión y el tiempo transcurrido. El campo

`cost.total_cost_usd` acumula el costo de todas las llamadas a API en la sesión actual. El campo `cost.total_duration_ms` mide el tiempo total transcurrido desde que comenzó la sesión, mientras que `cost.total_api_duration_ms` rastrea solo el tiempo dedicado a esperar respuestas de API.

Cada script formatea el costo como moneda y convierte milisegundos a minutos y segundos:



Una línea de estado que muestra el nombre del modelo, costo de sesión y duración

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

COST_FMT=$(printf '=%.2f' "$COST")
DURATION_SEC=$((DURATION_MS / 1000))
MINS=$((DURATION_SEC / 60))
SECS=$((DURATION_SEC % 60))

echo "[$MODEL] 💰 $COST_FMT | 🕒 ${MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

duration_sec = duration_ms // 1000
mins, secs = duration_sec // 60, duration_sec % 60

print(f"[{model}] 💰 ${cost:.2f} | ⌚ {mins}m {secs}s")
```

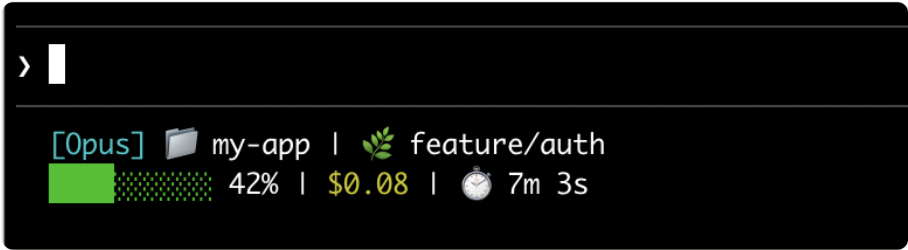
```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const cost = data.cost?.total_cost_usd || 0;
  const durationMs = data.cost?.total_duration_ms || 0;

  const durationSec = Math.floor(durationMs / 1000);
  const mins = Math.floor(durationSec / 60);
  const secs = durationSec % 60;

  console.log(`[${model}] 💰 $$${cost.toFixed(2)} | ⌚ ${mins}m ${secs}s`);
});
```

Muestra múltiples líneas

Tu script puede generar múltiples líneas para crear una pantalla más rica. Cada declaración `echo` produce una fila separada en el área de estado.



Una línea de estado de múltiples líneas que muestra el nombre del modelo, directorio, rama de git en la primera línea, y una barra de progreso de uso de contexto con costo y duración en la segunda línea

Este ejemplo combina varias técnicas: colores basados en umbrales (verde por debajo del 70%, amarillo 70-89%, rojo 90%+), una barra de progreso e información de rama de git. Cada declaración `print` o `echo` crea una fila separada:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[
0m'

## Pick bar color based on context usage
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
BAR=$(printf "%${FILLED}s" | tr ' ' '█')$(printf "%${EMPTY}s" | tr ' ' '░')

MINS=$((DURATION_MS / 60000)); SECS=$((DURATION_MS % 60000 / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | 🌿 $(git branch --show-
current 2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} 📁 ${DIR##*/}${BRANCH}"
COST_FMT=$(printf '=%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⌚ $
{MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m',
'\033[0m'

bar_color = RED if pct ≥ 90 else YELLOW if pct ≥ 70 else GREEN
filled = pct // 10
bar = '█' * filled + '░' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=True,
stderr=subprocess.DEVNULL).strip()
    branch = f" | 🌿 {branch}" if branch else ""
except:
    branch = ""

print(f"{CYAN}[{model}]{RESET} 📁 {directory}{branch}")
print(f"{bar_color}{bar}{RESET} {pct}% | {YELLOW}${cost:.2f}{RESET} | ⌚ {mins}m
{secs}s")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);
  const cost = data.cost?.total_cost_usd || 0;
  const pct = Math.floor(data.context_window?.used_percentage || 0);
  const durationMs = data.cost?.total_duration_ms || 0;

  const CYAN = '\x1b[36m', GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RED = '\x1b[31m', RESET = '\x1b[0m';

  const barColor = pct ≥ 90 ? RED : pct ≥ 70 ? YELLOW : GREEN;
  const filled = Math.floor(pct / 10);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

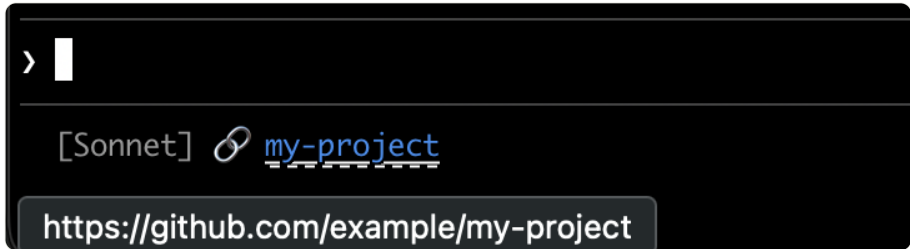
  const mins = Math.floor(durationMs / 60000);
  const secs = Math.floor((durationMs % 60000) / 1000);

  let branch = '';
  try {
    branch = execSync('git branch --show-current', { encoding: 'utf8', stdio:
['pipe', 'pipe', 'ignore'] }).trim();
    branch = branch ? ` | 🌿 ${branch}` : '';
  } catch {}

  console.log(`${CYAN}${model}${RESET} 📁 ${dir}${branch}`);
  console.log(`${barColor}${bar}${RESET} ${pct}% | ${YELLOW}${cost.toFixed(2)}${RESET} | 🕒 ${mins}m ${secs}s`);
});
```

Enlaces clickeables

Este ejemplo crea un enlace clickeable a tu repositorio de GitHub. Lee la URL remota de git, convierte el formato SSH a HTTPS con `sed`, y envuelve el nombre del repositorio en códigos de escape OSC 8. Mantén presionado Cmd (macOS) o Ctrl (Windows/Linux) y haz clic para abrir el enlace en tu navegador.



Una línea de estado que muestra un enlace clickeable a un repositorio de GitHub

Cada script obtiene la URL remota de git, convierte el formato SSH a HTTPS, y envuelve el nombre del repositorio en códigos de escape OSC 8. La versión de Bash usa `printf '%b'` que interpreta escapes de barra invertida de manera más confiable que `echo -e` en diferentes shells:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## Convert git SSH URL to HTTPS
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/github.com\/' | sed 's\/.git$\/')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 format: \e]8;;URL\a then TEXT then \e]8;;\a
    # printf %b interprets escape sequences reliably across shells
    printf '%b' "[${MODEL}]  \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[${MODEL}]"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## Get git remote URL
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # Convert SSH to HTTPS format
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 escape sequences
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] 🔗 {link}")
except:
    print(f"[{model}]")
```



```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;

  try {
    let remote = execSync('git remote get-url origin', { encoding: 'utf8', stdio: ['pipe', 'pipe', 'ignore'] }).trim();
    // Convert SSH to HTTPS format
    remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replace(/\.git$/, '');
    const repoName = path.basename(remote);
    // OSC 8 escape sequences
    const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
    console.log(`[${model}] 🔗 ${link}`);
  } catch {
    console.log(`[${model}]`);
  }
});
```

Cachea operaciones costosas

Tu script de línea de estado se ejecuta frecuentemente durante sesiones activas. Comandos como `git status` o `git diff` pueden ser lentos, especialmente en repositorios grandes. Este ejemplo cachea información de git en un archivo temporal y solo la actualiza cada 5 segundos.

Usa un nombre de archivo de caché estable y fijo como `/tmp/statusline-git-cache`. Cada invocación de línea de estado se ejecuta como un nuevo proceso, por lo que identificadores basados en procesos como `$$`, `os.getpid()`, o `process.pid` producen un valor diferente cada vez y el caché nunca se reutiliza.

Cada script verifica si el archivo de caché falta o es más antiguo que 5 segundos antes de ejecutar comandos de git:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

CACHE_FILE="/tmp/statusline-git-cache"
CACHE_MAX_AGE=5 # seconds

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -f %m is macOS, stat -c %Y is Linux
    [ (($((date +%s)) - $(stat -f %m "$CACHE_FILE" 2>/dev/null || stat -c %Y "$CACHE_FILE" 2>/dev/null || echo 0))) -gt $CACHE_MAX_AGE ]
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[ $MODEL ] 📁 ${DIR##*/} | 🌿 $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[ $MODEL ] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

CACHE_FILE = "/tmp/statusline-git-cache"
CACHE_MAX_AGE = 5 # seconds

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],
text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--numstat'],
text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'],
text=True).strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
```

```
print(f"[{model}] 📁 {directory} | 🌿 {branch} +{staged} ~{modified}")  
else:  
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node

const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);

  const CACHE_FILE = '/tmp/statusline-git-cache';
  const CACHE_MAX_AGE = 5; // seconds

  const cacheIsStale = () => {
    if (!fs.existsSync(CACHE_FILE)) return true;
    return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 >
    CACHE_MAX_AGE;
  };

  if (cacheIsStale()) {
    try {
      execSync('git rev-parse --git-dir', { stdio: 'ignore' });
      const branch = execSync('git branch --show-current', { encoding: 'utf8'
' }).trim();
      const staged = execSync('git diff --cached --numstat', { encoding: 'ut
f8' }).trim().split('\n').filter(Boolean).length;
      const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;
      fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
    } catch {
      fs.writeFileSync(CACHE_FILE, '||');
    }
  }

  const [branch, staged, modified] = fs.readFileSync(CACHE_FILE,
'utf8').trim().split('|');
```

```

    if (branch) {
        console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} +${staged} ~${modified}
`);
    } else {
        console.log(`[${model}] 📁 ${dir}`);
    }
}
});

```

Configuración de Windows

En Windows, Claude Code ejecuta comandos de línea de estado a través de Git Bash. Puedes invocar PowerShell desde ese shell:

```

{
  "statusLine": {
    "type": "command",
    "command": "powershell -NoProfile -File C:/Users/username/.claude/
statusline.ps1"
  }
}

```

```

$input_json = $input | Out-String | ConvertFrom-Json
$cwd = $input_json.cwd
$model = $input_json.model.display_name
$used = $input_json.context_window.used_percentage
$dirname = Split-Path $cwd -Leaf

if ($used) {
    Write-Host "$dirname [$model] ctx: $used%"
} else {
    Write-Host "$dirname [$model]"
}

```

O ejecuta un script de Bash directamente:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/ .claude/statusline.sh"
  }
}
```

```
#!/usr/bin/env bash
input=$(cat)
cwd=$(echo "$input" | grep -o ' "cwd": "[^"]*" ' | cut -d'"' -f4)
model=$(echo "$input" | grep -o ' "display_name": "[^"]*" ' | cut -d'"' -f4)
dirname="${cwd##*/[\\]}"
echo "$dirname [$model]"
```

Consejos

- **Prueba con entrada simulada:** `echo '{"model": {"display_name": "Opus"}, "context_window": {"used_percentage": 25}}' | ./statusline.sh`
- **Mantén la salida corta:** la barra de estado tiene un ancho limitado, por lo que la salida larga puede truncarse o ajustarse de manera incómoda
- **Cachea operaciones lentas:** tu script se ejecuta frecuentemente durante sesiones activas, por lo que comandos como `git status` pueden causar retrasos. Consulta el [ejemplo de caché](#) para saber cómo manejar esto.

Proyectos comunitarios como [ccstatusline](#) y [starship-claude](#) proporcionan configuraciones preconstruidas con temas y características adicionales.

Solución de problemas

La línea de estado no aparece

- Verifica que tu script sea ejecutable: `chmod +x ~/ .claude/statusline.sh`
- Comprueba que tu script genere salida a stdout, no stderr
- Ejecuta tu script manualmente para verificar que produce salida
- Si `disableAllHooks` está establecido en `true` en tu configuración, la línea de estado también está deshabilitada. Elimina esta configuración o establécela en `false` para volver a habilitarla.

- Ejecuta `claude --debug` para registrar el código de salida y stderr de la primera invocación de línea de estado en una sesión
- Pídele a Claude que lea tu archivo de configuración y ejecute el comando `statusLine` directamente para exponer errores

La línea de estado muestra `--` o valores vacíos

- Los campos pueden ser `null` antes de que se complete la primera respuesta de API
- Maneja valores nulos en tu script con valores predeterminados de respaldo como `// 0` en jq
- Reinicia Claude Code si los valores permanecen vacíos después de múltiples mensajes

El porcentaje de contexto muestra valores inesperados

- Usa `used_percentage` para un estado de contexto preciso en lugar de totales acumulativos
- Los `total_input_tokens` y `total_output_tokens` son acumulativos en toda la sesión y pueden exceder el tamaño de la ventana de contexto
- El porcentaje de contexto puede diferir de la salida `/context` debido a cuándo se calcula cada uno

Los enlaces OSC 8 no son clickeables

- Verifica que tu terminal admita hipervínculos OSC 8 (iTerm2, Kitty, WezTerm)
- Terminal.app no admite enlaces clickeables
- Las sesiones SSH y tmux pueden eliminar secuencias OSC dependiendo de la configuración
- Si las secuencias de escape aparecen como texto literal como `\e]8;;`, usa `printf '%b'` en lugar de `echo -e` para un manejo de escape más confiable

Problemas de visualización con secuencias de escape

- Las secuencias de escape complejas (colores ANSI, enlaces OSC 8) pueden ocasionalmente causar salida garbled si se superponen con otras actualizaciones de la interfaz
- Si ves texto corrupto, intenta simplificar tu script a salida de texto plano
- Las líneas de estado de múltiples líneas con códigos de escape son más propensas a problemas de renderizado que el texto plano de una sola línea

Errores de script o bloqueos

- Los scripts que salen con códigos distintos de cero o no producen salida hacen que la línea de estado se quede en blanco

- Los scripts lentos bloquean la línea de estado de actualizar hasta que se completen. Mantén los scripts rápidos para evitar salida obsoleta.
- Si una nueva actualización se activa mientras un script lento se está ejecutando, el script en vuelo se cancela
- Prueba tu script de forma independiente con entrada simulada antes de configurarlo

Las notificaciones comparten la fila de la línea de estado

- Las notificaciones del sistema como errores de servidor MCP, actualizaciones automáticas y advertencias de tokens se muestran en el lado derecho de la misma fila que tu línea de estado
- Habilitar el modo verbose agrega un contador de tokens a esta área
- En terminales estrechas, estas notificaciones pueden truncar tu salida de línea de estado

Part 13: Troubleshooting & Changelog

Solución de problemas

Descubre soluciones para problemas comunes con la instalación y el uso de Claude Code.

Solucionar problemas de instalación

Tip:

Si prefieres evitar la terminal por completo, la [aplicación de escritorio Claude Code](#) te permite instalar y usar Claude Code a través de una interfaz gráfica. Descárgala para [macOS](#) o [Windows](#) y comienza a codificar sin ninguna configuración de línea de comandos.

Encuentra el mensaje de error o síntoma que estás viendo:

Lo que ves	Solución
<code>command not found: claude</code> o <code>'claude' is not recognized</code>	Corrige tu PATH
<code>syntax error near</code> <code>unexpected token '<'</code>	El script de instalación devuelve HTML
<code>curl: (56) Failure writing</code> <code>output to destination</code>	Descarga el script primero, luego ejecútalo
<code>Killed</code> durante la instalación en Linux	Añade espacio de intercambio para servidores con poca memoria
<code>TLS connect error</code> o <code>SSL/TLS secure channel</code>	Actualiza los certificados CA

Lo que ves	Solución
<code>Failed to fetch version</code> o no se puede alcanzar el servidor de descarga	Verifica la conectividad de red y la configuración del proxy
<code>irm is not recognized</code> o <code>&& is not valid</code>	Usa el comando correcto para tu shell
Claude Code on Windows requires <code>git-bash</code>	Instala o configura Git Bash
Error loading shared library	Variante binaria incorrecta para tu sistema
<code>Illegal instruction</code> en Linux	Desajuste de arquitectura
<code>dyld: cannot load</code> o <code>Abort trap</code> en macOS	Incompatibilidad binaria
<code>Invoke-Expression: Missing argument in parameter list</code>	El script de instalación devuelve HTML
<code>App unavailable in region</code>	Claude Code no está disponible en tu país. Consulta países compatibles .
<code>unable to get local issuer certificate</code>	Configura certificados CA corporativos
<code>OAuth error</code> o <code>403 Forbidden</code>	Corrige la autenticación

Si tu problema no está listado, trabaja a través de estos pasos de diagnóstico.

Depurar problemas de instalación

Verifica la conectividad de red

El instalador descarga desde `storage.googleapis.com`. Verifica que puedas alcanzarlo:

```
curl -sI https://storage.googleapis.com
```

Si esto falla, tu red puede estar bloqueando la conexión. Las causas comunes incluyen:

- Firewalls corporativos o proxies bloqueando Google Cloud Storage
- Restricciones de red regional: intenta usar una VPN o una red alternativa
- Problemas de TLS/SSL: actualiza los certificados CA de tu sistema, o verifica si `HTTPS_PROXY` está configurado

Si estás detrás de un proxy corporativo, establece `HTTPS_PROXY` y `HTTP_PROXY` a la dirección de tu proxy antes de instalar. Pregunta a tu equipo de TI por la URL del proxy si no la conoces, o verifica la configuración del proxy en tu navegador.

Este ejemplo establece ambas variables de proxy, luego ejecuta el instalador a través de tu proxy:

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

Verifica tu PATH

Si la instalación fue exitosa pero obtienes un error `command not found` o `not recognized` al ejecutar `claude`, el directorio de instalación no está en tu PATH. Tu shell busca programas en directorios listados en PATH, y el instalador coloca `claude` en `~/.local/bin/claude` en macOS/Linux o `%USERPROFILE%\local\bin\claude.exe` en Windows.

Verifica si el directorio de instalación está en tu PATH listando tus entradas de PATH y filtrando por `local/bin`:

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

Si no hay salida, el directorio falta. Añádalo a tu configuración de shell:

```
## Zsh (macOS por defecto)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux por defecto)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Alternativamente, cierra y reabre tu terminal.

Verifica que la corrección funcionó:

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\bin'
```

Si no hay salida, añade el directorio de instalación a tu PATH de usuario:

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\local\bin", 'User')
```

Reinicia tu terminal para que el cambio surta efecto.

Verifica que la corrección funcionó:

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

Si no hay salida, abre Configuración del Sistema, ve a Variables de Entorno, y añade `%USERPROFILE%\local\bin` a tu variable PATH de usuario. Reinicia tu terminal.

Verifica que la corrección funcionó:

```
claude --version
```

Verifica instalaciones conflictivas

Múltiples instalaciones de Claude Code pueden causar desajustes de versión o comportamiento inesperado. Verifica qué está instalado:

macOS/Linux

Lista todos los binarios `claude` encontrados en tu PATH:

```
which -a claude
```

Verifica si están presentes las versiones del instalador nativo y npm:

```
ls -la ~/.local/bin/claude
```

```
ls -la ~/.claude/local/
```

```
npm -g ls @anthropic-ai/claude-code 2>/dev/null
```

Windows PowerShell

```
where.exe claude
```

```
Test-Path "$env:LOCALAPPDATA\Claude Code\claude.exe"
```

Si encuentras múltiples instalaciones, mantén solo una. La instalación nativa en `~/.local/bin/claude` es recomendada. Elimina cualquier instalación adicional:

Desinstala una instalación global de npm:

```
npm uninstall -g @anthropic-ai/claude-code
```

Elimina una instalación de Homebrew en macOS:

```
brew uninstall --cask claude-code
```

Verifica permisos de directorio

El instalador necesita acceso de escritura a `~/.local/bin/` y `~/.claude/`. Si la instalación falla con errores de permisos, verifica si estos directorios son escribibles:

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

Si alguno de los directorios no es escribible, crea el directorio de instalación y establece tu usuario como propietario:

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

Verifica que el binario funciona

Si `claude` está instalado pero se bloquea o cuelga al iniciar, ejecuta estas comprobaciones para reducir la causa.

Confirma que el binario existe y es ejecutable:

```
ls -la $(which claude)
```

En Linux, verifica si faltan bibliotecas compartidas. Si `ldd` muestra bibliotecas faltantes, es posible que necesites instalar paquetes del sistema. En Alpine Linux y otras distribuciones basadas en musl, consulta [Configuración de Alpine Linux](#).

```
ldd $(which claude) | grep "not found"
```

Ejecuta una comprobación rápida de cordura de que el binario puede ejecutarse:

```
claude --version
```

Problemas comunes de instalación

Estos son los problemas de instalación más frecuentemente encontrados y sus soluciones.

El script de instalación devuelve HTML en lugar de un script de shell

Al ejecutar el comando de instalación, puedes ver uno de estos errores:

```
bash: line 1: syntax error near unexpected token `<'  
bash: line 1: `<!DOCTYPE html>'
```

En PowerShell, el mismo problema aparece como:

```
Invoke-Expression: Missing argument in parameter list.
```

Esto significa que la URL de instalación devolvió una página HTML en lugar del script de instalación. Si la página HTML dice “App unavailable in region”, Claude Code no está disponible en tu país. Consulta [países compatibles](#).

De lo contrario, esto puede ocurrir debido a problemas de red, enrutamiento regional, o una interrupción temporal del servicio.

Soluciones:

1. Usa un método de instalación alternativo:

En macOS o Linux, instala a través de Homebrew:

```
brew install --cask claude-code
```

En Windows, instala a través de WinGet:

```
winget install Anthropic.ClaudeCode
```

1. **Reintentar después de unos minutos:** el problema suele ser temporal. Espera e intenta el comando original nuevamente.

command not found: claude después de la instalación

La instalación terminó pero **claude** no funciona. El error exacto varía según la plataforma:

Plataforma	Mensaje de error
macOS	<code>zsh: command not found: claude</code>
Linux	<code>bash: claude: command not found</code>
Windows CMD	<code>'claude' is not recognized as an internal or external command</code>
PowerShell	<code>claude : The term 'claude' is not recognized as the name of a cmdlet</code>

Esto significa que el directorio de instalación no está en la ruta de búsqueda de tu shell. Consulta [Verifica tu PATH](#) para la corrección en cada plataforma.

`curl: (56) Failure writing output to destination`

El comando `curl ... | bash` descarga el script y lo pasa directamente a Bash para su ejecución usando una tubería (`|`). Este error significa que la conexión se interrumpió antes de que el script terminara de descargarse. Las causas comunes incluyen interrupciones de red, la descarga siendo bloqueada a mitad de camino, o límites de recursos del sistema.

Soluciones:

1. **Verifica la estabilidad de la red:** Los binarios de Claude Code se alojan en Google Cloud Storage. Prueba que puedas alcanzarlo:

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

Si el comando se completa silenciosamente, tu conexión está bien y el problema probablemente es intermitente. Reintentar el comando de instalación. Si ves un error, tu red puede estar bloqueando la descarga.

1. **Intenta un método de instalación alternativo:**

En macOS o Linux:

```
brew install --cask claude-code
```

En Windows:

```
winget install Anthropic.ClaudeCode
```

Errores de conexión TLS o SSL

Errores como `curl: (35) TLS connect error, schannel: next InitializeSecurityContext failed`, o el `Could not establish trust relationship for the SSL/TLS secure channel` de PowerShell indican fallos en el apretón de manos TLS.

Soluciones:

1. Actualiza tus certificados CA del sistema:

En Ubuntu/Debian:

```
sudo apt-get update && sudo apt-get install ca-certificates
```

En macOS a través de Homebrew:

```
brew install ca-certificates
```

1. En Windows, habilita TLS 1.2 en PowerShell antes de ejecutar el instalador:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12  
irm https://claude.ai/install.ps1 | iex
```

1. Verifica la interferencia de proxy o firewall: los proxies corporativos que realizan inspección TLS pueden causar estos errores, incluyendo `unable to get local issuer certificate`. Establece `NODE_EXTRA_CA_CERTS` a tu paquete de certificados CA corporativos:

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

Pregunta a tu equipo de TI por el archivo de certificado si no lo tienes. También puedes intentar en una conexión directa para confirmar que el proxy es la causa.

`Failed to fetch version from storage.googleapis.com`

El instalador no pudo alcanzar el servidor de descarga. Esto típicamente significa que `storage.googleapis.com` está bloqueado en tu red.

Soluciones:

1. Prueba la conectividad directamente:

```
curl -sI https://storage.googleapis.com
```

1. **Si estás detrás de un proxy**, establece `HTTPS_PROXY` para que el instalador pueda enrutarse a través de él. Consulta [configuración de proxy](#) para detalles.

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **Si estás en una red restringida**, intenta una red diferente o VPN, o usa un método de instalación alternativo:

En macOS o Linux:

```
brew install --cask claude-code
```

En Windows:

```
winget install Anthropic.ClaudeCode
```

Windows: `irm` o `&&` no reconocido

Si ves `'irm' is not recognized` o `The token '&&' is not valid`, estás ejecutando el comando incorrecto para tu shell.

- **`irm` no reconocido:** estás en CMD, no en PowerShell. Tienes dos opciones:
Abre PowerShell buscando “PowerShell” en el menú Inicio, luego ejecuta el comando de instalación original:

```
irm https://claude.ai/install.ps1 | iex
```

O quédate en CMD y usa el instalador de CMD en su lugar:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del
install.cmd
```

- **&& no válido:** estás en PowerShell pero ejecutaste el comando del instalador de CMD. Usa el instalador de PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Instalación eliminada en servidores Linux con poca memoria

Si ves **Killed** durante la instalación en un VPS o instancia en la nube:

```
Setting up Claude Code...
Installing Claude Code native build latest...
bash: line 142: 34803 Killed      "$binary_path" install ${TARGET:+"$TARGET"}
```

El asesino de OOM de Linux terminó el proceso porque el sistema se quedó sin memoria. Claude Code requiere al menos 4 GB de RAM disponible.

Soluciones:

1. **Añade espacio de intercambio** si tu servidor tiene RAM limitada. El intercambio usa espacio en disco como memoria de desbordamiento, permitiendo que la instalación se complete incluso con RAM física baja.

Crea un archivo de intercambio de 2 GB y habilítalo:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

Luego reintentar la instalación:

```
curl -fsSL https://claude.ai/install.sh | bash
```

1. **Cierra otros procesos** para liberar memoria antes de instalar.

2. **Usa una instancia más grande** si es posible. Claude Code requiere al menos 4 GB de RAM.

La instalación se cuelga en Docker

Al instalar Claude Code en un contenedor Docker, instalar como root en `/` puede causar cuelgues.

Soluciones:

1. **Establece un directorio de trabajo** antes de ejecutar el instalador. Cuando se ejecuta desde `/`, el instalador escanea todo el sistema de archivos, lo que causa un uso excesivo de memoria. Establecer `WORKDIR` limita el escaneo a un directorio pequeño:

```
WORKDIR /tmp
RUN curl -fsSL https://claude.ai/install.sh | bash
```

1. **Aumenta los límites de memoria de Docker** si usas Docker Desktop:

```
docker build --memory=4g .
```

Windows: Claude Desktop anula el comando CLI `claude`

Si instalaste una versión anterior de Claude Desktop, puede registrar un `Claude.exe` en el directorio `WindowsApps` que toma prioridad en PATH sobre Claude Code CLI. Ejecutar `claude` abre la aplicación de escritorio en lugar de la CLI.

Actualiza Claude Desktop a la última versión para corregir este problema.

Windows: “Claude Code on Windows requires git-bash”

Claude Code en Windows nativo necesita [Git para Windows](#), que incluye Git Bash.

Si Git no está instalado, descárgalo e instálalo desde git-scm.com/downloads/win. Durante la configuración, selecciona “Add to PATH.” Reinicia tu terminal después de instalar.

Si Git ya está instalado pero Claude Code aún no puede encontrarlo, establece la ruta en tu [archivo settings.json](#):

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Si tu Git está instalado en otro lugar, encuentra la ruta ejecutando `where.exe git` en PowerShell y usa la ruta `bin\\bash.exe` de ese directorio.

Linux: variante binaria incorrecta instalada (desajuste musl/glibc)

Si ves errores sobre bibliotecas compartidas faltantes como `libstdc++.so.6` o `libgcc_s.so.1` después de la instalación, el instalador puede haber descargado la variante binaria incorrecta para tu sistema.

```
Error loading shared library libstdc++.so.6: No such file or directory
```

Esto puede ocurrir en sistemas basados en glibc que tienen paquetes de compilación cruzada musl instalados, causando que el instalador detecte incorrectamente el sistema como musl.

Soluciones:

1. Verifica qué libc usa tu sistema:

```
ldd /bin/ls | head -1
```

Si muestra `linux-vdso.so` o referencias a `/lib/x86_64-linux-gnu/`, estás en glibc. Si muestra `musl`, estás en musl.

1. **Si estás en glibc pero obtuviste el binario musl**, elimina la instalación y reinstala. También puedes descargar manualmente el binario correcto desde el bucket de GCS en <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json>. Abre un [problema en GitHub](#) con la salida de `ldd /bin/ls` y `ls /lib/libc.musl*`.

2. **Si realmente estás en musl** (Alpine Linux), instala los paquetes requeridos:

```
apk add libgcc libstdc++ ripgrep
```

Illegal instruction en Linux

Si el instalador imprime **Illegal instruction** en lugar del mensaje **Killed** de OOM, el binario descargado no coincide con la arquitectura de tu CPU. Esto ocurre comúnmente en servidores ARM que reciben un binario x86, o en CPUs más antiguas que carecen de conjuntos de instrucciones requeridos.

```
bash: line 142: 2238232 Illegal instruction   "$binary_path" install ${TARGET:+"$TARGET"}
```

Soluciones:

1. Verifica tu arquitectura:

```
uname -m
```

x86_64 significa 64-bit Intel/AMD, **aarch64** significa ARM64. Si el binario no coincide, [abre un problema en GitHub](#) con la salida.

1. Intenta un método de instalación alternativo mientras se resuelve el problema de arquitectura:

```
brew install --cask claude-code
```

dyld: cannot load en macOS

Si ves **dyld: cannot load** o **Abort trap: 6** durante la instalación, el binario es incompatible con tu versión de macOS o hardware.

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

Soluciones:

1. **Verifica tu versión de macOS:** Claude Code requiere macOS 13.0 o posterior. Abre el menú Apple y selecciona About This Mac para verificar tu versión.
2. **Actualiza macOS** si estás en una versión anterior. El binario usa comandos de carga que las versiones anteriores de macOS no soportan.
3. **Intenta Homebrew** como método de instalación alternativo:

```
brew install --cask claude-code
```

Problemas de instalación en Windows: errores en WSL

Podrías encontrar los siguientes problemas en WSL:

Problemas de detección de SO/plataforma: si recibes un error durante la instalación, WSL puede estar usando `npm` de Windows. Intenta:

- Ejecuta `npm config set os linux` antes de la instalación
- Instala con `npm install -g @anthropic-ai/claude-code --force --no-os-check`. No uses `sudo`.

Errores de Node no encontrado: si ves `exec: node: not found` al ejecutar `claude`, tu entorno WSL puede estar usando una instalación de Windows de Node.js. Puedes confirmar esto con `which npm` y `which node`, que deberían apuntar a rutas de Linux que comienzan con `/usr/` en lugar de `/mnt/c/`. Para corregir esto, intenta instalar Node a través del gestor de paquetes de tu distribución de Linux o a través de `nvm`.

Conflictos de versión de nvm: si tienes `nvm` instalado tanto en WSL como en Windows, puedes experimentar conflictos de versión al cambiar versiones de Node en WSL. Esto ocurre porque WSL importa el `PATH` de Windows por defecto, causando que `npm/nvm` de Windows tenga prioridad sobre la instalación de WSL.

Puedes identificar este problema por:

- Ejecutar `which npm` y `which node` - si apuntan a rutas de Windows (comenzando con `/mnt/c/`), se están usando versiones de Windows
- Experimentar funcionalidad rota después de cambiar versiones de Node con `nvm` en WSL

Para resolver este problema, corrige tu `PATH` de Linux para asegurar que las versiones de `node/npm` de Linux tengan prioridad:

Solución principal: Asegúrate de que `nvm` se carga correctamente en tu shell

La causa más común es que `nvm` no se carga en shells no interactivos. Añade lo siguiente a tu archivo de configuración de shell (`~/.bashrc`, `~/.zshrc`, etc.):


```
## Carga nvm si existe
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

O ejecuta directamente en tu sesión actual:

```
source ~/.nvm/nvm.sh
```

Alternativa: Ajusta el orden de PATH

Si nvm se carga correctamente pero las rutas de Windows aún tienen prioridad, puedes prepender explícitamente tus rutas de Linux a PATH en tu configuración de shell:

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

Warning:

Evita desactivar la importación de PATH de Windows a través de `appendWindowsPath = false` ya que esto rompe la capacidad de llamar ejecutables de Windows desde WSL. De manera similar, evita desinstalar Node.js de Windows si lo usas para desarrollo de Windows.

Configuración de sandbox WSL2

[Sandboxing](#) es compatible en WSL2 pero requiere instalar paquetes adicionales. Si ves un error como “Sandbox requires socat and bubblewrap” al ejecutar `/sandbox`, instala las dependencias:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 no soporta sandboxing. Si ves “Sandboxing requires WSL2”, necesitas actualizar a WSL2 o ejecutar Claude Code sin sandboxing.

Errores de permisos durante la instalación

Si el instalador nativo falla con errores de permisos, el directorio de destino puede no ser escribible. Consulta [Verifica permisos de directorio](#).

Si instalaste previamente con npm y estás teniendo errores específicos de permisos de npm, cambia al instalador nativo:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Permisos y autenticación

Estas secciones abordan fallos de inicio de sesión, problemas de tokens, y comportamiento de solicitudes de permisos.

Solicitudes de permisos repetidas

Si te encuentras aprobando repetidamente los mismos comandos, puedes permitir que herramientas específicas se ejecuten sin aprobación usando el comando `/permissions`. Consulta [documentación de Permisos](#).

Problemas de autenticación

Si estás experimentando problemas de autenticación:

1. Ejecuta `/logout` para cerrar sesión completamente
2. Cierra Claude Code
3. Reinicia con `claude` y completa el proceso de autenticación nuevamente

Si el navegador no se abre automáticamente durante el inicio de sesión, presiona `c` para copiar la URL de OAuth a tu portapapeles, luego pégala en tu navegador manualmente.

Error OAuth: Código inválido

Si ves `OAuth error: Invalid code. Please make sure the full code was copied`, el código de inicio de sesión expiró o fue truncado durante la copia-pegar.

Soluciones:

- Presiona Enter para reintentar y completa el inicio de sesión rápidamente después de que se abra el navegador
- Escribe `c` para copiar la URL completa si el navegador no se abre automáticamente
- Si usas una sesión remota/SSH, el navegador puede abrirse en la máquina incorrecta. Copia la URL mostrada en la terminal y ábrela en tu navegador local en su lugar.

403 Forbidden después del inicio de sesión

Si ves `API Error: 403 {"error":{"type":"forbidden"},"message":"Request not allowed"}}` después de iniciar sesión:

- **Usuarios de Claude Pro/Max:** verifica que tu suscripción esté activa en claude.ai/settings
- **Usuarios de Console:** confirma que tu cuenta tiene el rol “Claude Code” o “Developer” asignado por tu administrador
- **Detrás de un proxy:** los proxies corporativos pueden interferir con solicitudes de API. Consulta [configuración de red](#) para configuración de proxy.

El inicio de sesión OAuth falla en WSL2

El inicio de sesión basado en navegador en WSL2 puede fallar si WSL no puede abrir tu navegador de Windows. Establece la variable de entorno `BROWSER` :

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

O copia la URL manualmente: cuando aparezca la solicitud de inicio de sesión, presiona `C` para copiar la URL de OAuth, luego pégala en tu navegador de Windows.

“Not logged in” o token expirado

Si Claude Code te solicita iniciar sesión nuevamente después de una sesión, tu token de OAuth puede haber expirado.

Ejecuta `/login` para re-autenticarte. Si esto ocurre frecuentemente, verifica que tu reloj del sistema sea preciso, ya que la validación de tokens depende de marcas de tiempo correctas.

Ubicaciones de archivos de configuración

Claude Code almacena la configuración en varias ubicaciones:

Archivo	Propósito
<code>~/.claude/settings.json</code>	Configuración de usuario (permisos, hooks, anulaciones de modelo)
<code>.claude/settings.json</code>	Configuración de proyecto (registrada en control de fuente)

Archivo	Propósito
<code>.claude/ settings.local.json</code>	Configuración de proyecto local (no confirmada)
<code>~/.claude.json</code>	Estado global (tema, OAuth, servidores MCP)
<code>.mcp.json</code>	Servidores MCP de proyecto (registrados en control de fuente)
<code>managed-mcp.json</code>	Servidores MCP administrados
Configuración adminis- trada	Configuración administrada (administrada por servidor, políticas MDM/nivel de SO, o basada en archivos)

En Windows, `~` se refiere a tu directorio de inicio de usuario, como `C:\Users\TuNombre`.

Para detalles sobre la configuración de estos archivos, consulta [Configuración](#) y [MCP](#).

Restablecimiento de configuración

Para restablecer Claude Code a la configuración predeterminada, puedes eliminar los archivos de configuración:

```
## Restablece toda la configuración de usuario y estado
rm ~/.claude.json
rm -rf ~/.claude/

## Restablece la configuración específica del proyecto
rm -rf .claude/
rm .mcp.json
```

Warning:

Esto eliminará toda tu configuración, configuraciones de servidores MCP, e historial de sesiones.

Rendimiento y estabilidad

Estas secciones cubren problemas relacionados con el uso de recursos, capacidad de respuesta, y comportamiento de búsqueda.

Alto uso de CPU o memoria

Claude Code está diseñado para funcionar con la mayoría de entornos de desarrollo, pero puede consumir recursos significativos al procesar bases de código grandes. Si estás experimentando problemas de rendimiento:

1. Usa `/compact` regularmente para reducir el tamaño del contexto
2. Cierra y reinicia Claude Code entre tareas principales
3. Considera añadir directorios de compilación grandes a tu archivo `.gitignore`

El comando se cuelga o congela

Si Claude Code parece no responder:

1. Presiona Ctrl+C para intentar cancelar la operación actual
2. Si no responde, es posible que necesites cerrar la terminal y reiniciar

Problemas de búsqueda y descubrimiento

Si la herramienta Search, menciones `@file`, agentes personalizados, y skills personalizados no funcionan, instala el sistema `ripgrep`:

```
## macOS (Homebrew)
brew install ripgrep

## Windows (winget)
winget install BurntSushi.ripgrep.MSVC

## Ubuntu/Debian
sudo apt install ripgrep

## Alpine Linux
apk add ripgrep

## Arch Linux
pacman -S ripgrep
```

Luego establece `USE_BUILTIN_RIPGREP=0` en tu [entorno](#).

Resultados de búsqueda lentos o incompletos en WSL

Las penalizaciones de rendimiento de lectura de disco al [trabajar entre sistemas de archivos en WSL](#) pueden resultar en menos coincidencias de las esperadas al usar Claude Code en WSL. La búsqueda aún funciona, pero devuelve menos resultados que en un sistema de archivos nativo.

Note:

`/doctor` mostrará Search como OK en este caso.

Soluciones:

1. **Envía búsquedas más específicas:** reduce el número de archivos buscados especificando directorios o tipos de archivo: “Search for JWT validation logic in the auth-service package” o “Find use of md5 hash in JS files”.
2. **Mueve el proyecto al sistema de archivos de Linux:** si es posible, asegúrate de que tu proyecto esté ubicado en el sistema de archivos de Linux (`/home/`) en lugar del sistema de archivos de Windows (`/mnt/c/`).
3. **Usa Windows nativo en su lugar:** considera ejecutar Claude Code de forma nativa en Windows en lugar de a través de WSL, para mejor rendimiento del sistema de archivos.

Problemas de integración de IDE

Si Claude Code no se conecta a tu IDE o se comporta inesperadamente dentro de una terminal de IDE, intenta las soluciones a continuación.

IDE de JetBrains no detectado en WSL2

Si estás usando Claude Code en WSL2 con IDEs de JetBrains y obteniendo errores “No available IDEs detected”, esto probablemente se debe a la configuración de red de WSL2 o Windows Firewall bloqueando la conexión.

Modos de red de WSL2

WSL2 usa red NAT por defecto, lo que puede prevenir la detección de IDE. Tienes dos opciones:

Opción 1: Configura Windows Firewall (recomendado)

1. Encuentra tu dirección IP de WSL2:

```
wsl hostname -I
## Salida de ejemplo: 172.21.123.45
```

1. Abre PowerShell como Administrador y crea una regla de firewall:

```
New-NetFirewallRule -DisplayName "Allow WSL2 Internal Traffic" -Direction Inbound
-Protocol TCP -Action Allow -RemoteAddress 172.21.0.0/16 -LocalAddress 172.21.0.0/16
```

Ajusta el rango de IP basado en tu subred de WSL2 del paso 1.

1. Reinicia tanto tu IDE como Claude Code

Opción 2: Cambia a red reflejada

Añade a `.wslconfig` en tu directorio de usuario de Windows:

```
[wsl2]
networkingMode=mirrored
```

Luego reinicia WSL con `wsl --shutdown` desde PowerShell.

Note:

Estos problemas de red solo afectan a WSL2. WSL1 usa la red del host directamente y no requiere estas configuraciones.

Para consejos de configuración adicionales de JetBrains, consulta la [guía de IDE de JetBrains](#).

Reporta problemas de integración de IDE en Windows

Si estás experimentando problemas de integración de IDE en Windows, [crea un problema](#) con la siguiente información:

- Tipo de entorno: Windows nativo (Git Bash) o WSL1/WSL2
- Modo de red de WSL, si aplica: NAT o reflejado
- Nombre y versión del IDE
- Versión de extensión/plugin de Claude Code
- Tipo de shell: Bash, Zsh, PowerShell, etc.

La tecla Escape no funciona en terminales de IDE de JetBrains

Si estás usando Claude Code en terminales de JetBrains y la tecla **Esc** no interrumpe el agente como se espera, esto probablemente se debe a un choque de atajos de teclado con los atajos de teclado predeterminados de JetBrains.

Para corregir este problema:

1. Ve a Settings → Tools → Terminal
2. Ya sea:
 - Desmarca “Move focus to the editor with Escape”, o
 - Haz clic en “Configure terminal keybindings” y elimina el atajo “Switch focus to Editor”
3. Aplica los cambios

Esto permite que la tecla **Esc** interrumpa correctamente las operaciones de Claude Code.

Problemas de formato de markdown

Claude Code a veces genera archivos markdown con etiquetas de lenguaje faltantes en cercas de código, lo que puede afectar el resaltado de sintaxis y la legibilidad en GitHub, editores, y herramientas de documentación.

Etiquetas de lenguaje faltantes en bloques de código

Si notas bloques de código como este en markdown generado:

```
```  

function example() {
 return "hello";
}
```text
```

En lugar de bloques correctamente etiquetados como:

```
```javascript  
function example() {
 return "hello";
}
```text
```


Soluciones:

1. **Pide a Claude que añada etiquetas de lenguaje:** solicita “Add appropriate language tags to all code blocks in this markdown file.”
2. **Usa hooks de post-procesamiento:** configura hooks de formato automático para detectar y añadir etiquetas de lenguaje faltantes. Consulta [Auto-format code after edits](#) para un ejemplo de un hook de formato PostToolUse.
3. **Verificación manual:** después de generar archivos markdown, revísalos para el formato correcto de bloques de código y solicita correcciones si es necesario.

Espaciado e formato inconsistentes

Si el markdown generado tiene líneas en blanco excesivas o espaciado inconsistente:

Soluciones:

1. **Solicita correcciones de formato:** pide a Claude que “Fix spacing and formatting issues in this markdown file.”
2. **Usa herramientas de formato:** configura hooks para ejecutar formateadores de markdown como `prettier` o scripts de formato personalizados en archivos markdown generados.
3. **Especifica preferencias de formato:** incluye requisitos de formato en tus solicitudes o archivos de [memoria](#) del proyecto.

Reduce problemas de formato de markdown

Para minimizar problemas de formato:

- **Sé explícito en solicitudes:** pide “properly formatted markdown with language-tagged code blocks”
- **Usa convenciones de proyecto:** documenta tu estilo de markdown preferido en `CLAUDE.md`
- **Configura hooks de validación:** usa hooks de post-procesamiento para verificar y corregir automáticamente problemas comunes de formato

Obtén más ayuda

Si estás experimentando problemas no cubiertos aquí:

1. Usa el comando `/bug` dentro de Claude Code para reportar problemas directamente a Anthropic
2. Verifica el [repositorio de GitHub](#) para problemas conocidos

3. Ejecuta `/doctor` para diagnosticar problemas. Verifica:
 - Tipo de instalación, versión, y funcionalidad de búsqueda
 - Estado de actualización automática y versiones disponibles
 - Archivos de configuración inválidos (JSON malformado, tipos incorrectos)
 - Errores de configuración de servidores MCP
 - Problemas de configuración de atajos de teclado
 - Advertencias de uso de contexto (archivos CLAUDE.md grandes, alto uso de tokens de MCP, reglas de permisos inalcanzables)
 - Errores de carga de plugins y agentes
4. Pregunta a Claude directamente sobre sus capacidades y características - Claude tiene acceso integrado a su documentación