BÓVEDA · GRATIS

# Claude *a fondo*, el manual completo

La referencia larga de la cuenta, para cuando ya usas Claude y quieres lo que hay debajo: modelos y precios, la app entera, Claude Code de arriba abajo, la API y los apéndices para consultar. No se lee de un tirón; se abre por donde haga falta.

**claudeprofessor.page**
Comprobado el 27 de agosto de 2026

@claudeprofessor

ÍNDICE · I

Este no es el manual con el que se empieza: para eso está **Claude de cero, en una tarde**. Este es el de después, cuando ya la usas y te falta lo de debajo. Los apartados son independientes y están pensados para abrirse sueltos.

> **Comprobado el 27 de agosto de 2026** en las páginas oficiales de Anthropic: documentación de la API, documentación de Claude Code, centro de ayuda y centro de privacidad. Los precios y los nombres de menú cambian; si algo no coincide, manda la fuente.

La segunda mitad es para quien construye algo y para quien administra Claude en un equipo. Los apéndices son tablas de consulta: no se leen, se buscan.

01 · TODO SE MIDE EN TOKENS

Antes de tocar un ajuste conviene entender la unidad de medida. Claude no lee letras ni palabras: parte el texto en **tokens**, trozos de unos cuatro caracteres. Todo lo demás —el precio, el límite de la conversación, lo que te queda de plan— está contado en esa moneda.

## Cuánto es un token

Como regla de bolsillo, **un token es más o menos 0,75 palabras** en inglés. En español sale peor parado: los acentos y las palabras largas se trocean más, así que cuenta un poco más de tokens para el mismo texto.

Se cobran las dos direcciones y a precios distintos. Los **tokens de entrada** son todo lo que le mandas: tu mensaje, el historial de la conversación, los archivos adjuntos y las instrucciones del sistema. Los **tokens de salida** son lo que escribe él, y valen entre cuatro y cinco veces más.

## Por qué te importa aunque no uses la API

Aunque pagues una cuota plana y no veas un solo número, el consumo interno se mide igual. Cada vez que sigues una conversación, se le vuelve a mandar entera: el turno veinte cuesta mucho más que el primero porque arrastra los diecinueve anteriores. De ahí sale el consejo que parece superstición y no lo es: **abre conversación nueva cuando cambies de tema**.

## El detalle que descoloca a todo el mundo

Los modelos de la generación 4.7 en adelante usan un troceador distinto que **produce alrededor de un 30% más de tokens para el mismo texto**. No es que gasten más de verdad: es que la unidad encogió, y los precios están calculados con eso ya dentro. Si comparas cifras de tokens entre generaciones, no estás comparando lo mismo.

---

Regla práctica: lo que ocupa contexto es lo que has pegado, no lo que has escrito. Un PDF de cincuenta páginas pesa más que todas tus preguntas de la semana.

La **ventana de contexto** es todo lo que Claude puede tener delante a la vez: tu conversación, los archivos, las instrucciones. No es memoria; es campo de visión. Cuando se llena, algo tiene que salir.

## Cuánto cabe hoy

Los tres modelos grandes —Opus 5, Sonnet 5 y Fable 5— trabajan con **un millón de tokens**. Haiku 4.5 se queda en 200.000. Traducido: un millón de tokens son **unas 555.000 palabras**, o dos millones y medio de caracteres. Un libro largo entero, con margen.

Ojo con esa conversión, que cambió: con el troceador anterior, ese mismo millón daba para unas 750.000 palabras. La ventana no encogió, cambió la regla de medir.

## La parte que no puedes usar

De la ventana se reserva un trozo para la respuesta, así que **la conversación más larga posible siempre es algo menor que la ventana entera**. El tope de salida de los modelos grandes es de 128.000 tokens por respuesta; Haiku 4.5 llega a 64.000.

## Qué pasa cuando se llena

En la web te avisa de que la conversación es demasiado larga y toca empezar otra. En Claude Code hay compactación: resume lo viejo y sigue. Y en los planes de pago, la longitud disponible depende del modelo: **los más nuevos llegan al millón, otros se quedan en 500.000 o 200.000**.

---

Meter un millón de tokens es posible y a veces no es buena idea: cuanto más ruido hay delante, más cuesta encontrar la aguja. Contexto limpio gana a contexto grande.

Cuatro modelos vivos, y no son versiones mejores y peores del mismo: son tamaños distintos para trabajos distintos. Estos son los datos oficiales, con precio por millón de tokens de la API.

## 1 Claude Opus 5

El caballo de batalla para trabajo complejo y agentes de programación. Un millón de contexto, 128.000 de salida, razonamiento adaptativo. **5 $ de entrada y 25 $ de salida** por millón. Su conocimiento fiable llega hasta mayo de 2026.

## 2 Claude Sonnet 5

La mejor mezcla de velocidad e inteligencia, y el que sale a cuenta en volumen. Mismo millón de contexto y misma salida que Opus. **2 $ y 10 $** por millón: menos de la mitad.

## 3 Claude Haiku 4.5

El más rápido, con inteligencia casi de primera línea. Se queda en **200.000 de contexto** y 64.000 de salida, y no admite el ajuste de esfuerzo. **1 $ y 5 $** por millón.

## 4 Claude Fable 5

El más capaz que hay publicado, pensado para agentes de recorrido largo. Piensa siempre, sin interruptor. Es también el más lento y el más caro: **10 $ y 50 $** por millón.

La pregunta no es cuál es mejor, sino cuál es suficiente. Pagar Opus para clasificar correos es tirar dinero; poner Haiku a refactorizar un módulo es perder la tarde.

## El criterio corto

**Empieza por Opus 5 y baja si no hace falta.** Es la recomendación oficial de Anthropic para trabajo complejo, y en la práctica funciona: si el resultado te vale con Sonnet, cambias y ahorras más de la mitad.

## Dónde encaja cada uno

**Haiku 4.5** para tareas mecánicas y con prisa: clasificar, extraer un dato, resumir algo corto, rastrear un repositorio buscando dónde está una función. Cuidado con su contexto de 200.000: un documento gordo no le cabe.

**Sonnet 5** para casi todo el trabajo de producción: redactar, revisar, programar cosas acotadas, tareas repetidas muchas veces. Es el que usarías por defecto si el dinero lo pusieras tú.

**Opus 5** cuando el problema es de verdad difícil o largo: arquitectura, un bug que no se ve, una sesión de agente que va a durar horas y tomar decisiones sola.

**Fable 5** cuando la corrección importa más que el gasto y la tarea va a correr sin nadie mirando. Es caro a posta.

---

En Claude Code se cambia con el comando de modelo en mitad de la sesión, sin perder el hilo. No hay que decidirlo al principio.

Los modelos actuales pueden pensar antes de escribir. Ya no se configura con un presupuesto de tokens: **el modelo decide solo cuánto piensa**, y tú ajustas el listón con otra perilla.

## Razonamiento adaptativo

Opus 5, Sonnet 5 y Fable 5 usan **razonamiento adaptativo**: miran el problema y deciden cuánto merece la pena pensarlo. En Fable 5 está siempre encendido y no se puede apagar. Haiku 4.5 usa el modo antiguo, el de presupuesto fijo.

El sistema de antes —decirle «piensa con 10.000 tokens de margen»— está retirado. En los modelos nuevos ese parámetro **devuelve error**, no es que se ignore.

## El esfuerzo

Lo que sí controlas es el **esfuerzo**, en cinco escalones: bajo, medio, alto, muy alto y máximo. El valor por defecto es **alto**. Bajarlo hace que encadene menos llamadas a herramientas, escriba menos preámbulo y confirme más corto; subirlo hace lo contrario y cuesta más.

En Claude Code se cambia con el comando de esfuerzo. Merece la pena tocarlo: para tareas mecánicas, esfuerzo bajo hace el mismo trabajo por bastante menos.

---

Lo que Claude piensa no se te enseña en crudo nunca, en ningún modelo. Como mucho ves un resumen, y hay que pedirlo.

Precios oficiales de claude.com/pricing. Las cifras son en dólares y cambian; lo que no cambia tan rápido es qué desbloquea cada escalón, que es lo que de verdad decide.

## 0 $ Gratuito

Chat en web, iOS, Android y escritorio. Generación de código, búsqueda web, memoria, creación de archivos, extensiones de escritorio, conectores de Slack y Google Workspace, y pensamiento extendido. Da para probar de verdad, no es una demo.

## 17 $ Pro · al mes con pago anual

**200 $ por adelantado**, o **20 $ si pagas mes a mes**. Suma más uso, **Claude Code**, Cowork, Design, Science, proyectos ilimitados, acceso a Investigación, varios modelos y la integración con Microsoft 365. Es el salto que cambia algo.

## 100 $ Max · desde, al mes

En dos alturas: **5 veces o 20 veces** el uso de Pro. Además, topes de salida más altos, acceso temprano a funciones y **prioridad en horas de tráfico alto**. Se nota sobre todo si vives en Claude Code.

## 20 $ Team · por puesto y mes con pago anual

**25 $ mensual**, para equipos **de 2 a 150**. Trae Claude Code, Cowork, Design, Science, búsqueda de empresa, SSO y facturación central. El puesto premium cuesta **100 $ anual o 125 $ mensual** y da 5 veces más uso.

07 · LOS LÍMITES DE USO

Es la queja número uno de todo el mundo, y casi siempre nace de no saber cómo se cuenta. No se cuenta por mensajes al día.

> **Anthropic no publica las cifras.** No hay un número oficial de mensajes por plan, y quien te dé uno se lo está inventando o lo ha medido en su cuenta. Lo que sí está documentado es el mecanismo, y es lo que va aquí.

## Dos relojes a la vez

Hay una **ventana de sesión de cinco horas** y, por encima, un **límite semanal** que se aplica a todos los modelos juntos. El semanal se reinicia a una hora fija de la semana **asignada a tu cuenta**: no es el lunes a las cero horas para todos.

Eso explica el efecto raro de gastar mucho un martes y notarlo el jueves. Estás contra el reloj semanal, no contra el de la sesión.

## Dónde se mira de verdad

En Pro, Max, Team y Enterprise por puesto: **Ajustes, y dentro Uso**. Salen barras de progreso con lo consumido de la ventana de cinco horas y de la semana. Es el único dato fiable, porque es el tuyo.

## Y si lo agotas

Tres salidas y no hay más: esperar al reinicio, subir de plan o **comprar créditos de uso**. Los créditos son la novedad que casi nadie conoce: se pagan aparte y sirven para terminar lo que estabas haciendo sin cambiar de suscripción.

---

Lo que más gasta sin que se note: conversaciones larguísimas, adjuntos grandes y modelos grandes para tareas pequeñas. En ese orden.

Si vas a construir algo, el precio por millón de tokens es solo la mitad de la historia. Los descuentos son grandes y casi nadie los usa.

## La caché de prompts

Si mandas el mismo bloque de contexto una y otra vez —un documento, unas instrucciones largas—, se puede cachear. **Escribir en caché cuesta 1,25 veces** el precio de entrada si la guardas cinco minutos, o **2 veces** si la guardas una hora. Pero **leer de caché cuesta 0,1 veces**: una décima parte.

Traducido: con la caché de cinco minutos, **compensa a partir de la primera relectura**. Con la de una hora, a partir de la segunda.

## Los lotes

Si el trabajo no tiene prisa, la API de lotes lo procesa de forma asíncrona **con un 50% de descuento en entrada y salida**. Opus 5 pasa de 5 y 25 dólares a 2,50 y 12,50. Y los dos descuentos se acumulan: lote más caché.

## Lo que se paga aparte

La **búsqueda web** cuesta 10 $ por cada 1.000 búsquedas, más los tokens de lo que encuentre. La **descarga de páginas** no cuesta nada extra, solo los tokens del contenido. La **ejecución de código** trae 1.550 horas gratis al mes por organización y luego 0,05 $ la hora por contenedor.

---

El millón de contexto va a precio normal: una petición de 900.000 tokens se cobra al mismo precio por token que una de 9.000.

09 · DÓNDE ESTÁ CADA AJUSTE

Casi todo lo que la gente cree que Claude no hace, lo hace: está apagado o está en un menú que nunca ha abierto. Estas son las tres puertas, y conviene saberlas de memoria porque el resto del manual las nombra sin parar.

## 1 Ajustes, dentro Capacidades

Se llega pulsando **tus iniciales, abajo a la izquierda**. Aquí vive el interruptor de **Ejecución de código y creación de archivos**, que es el que enciende de golpe los artefactos y los archivos de verdad. En el móvil: iniciales en la barra lateral, Ajustes, Capacidades.

## 2 Ajustes, dentro Memoria

Aquí se enciende o se apaga la memoria, se leen todos los temas que Claude ha guardado sobre ti, se editan uno a uno y se decide qué hacer con los temas sensibles. Casi nadie ha entrado nunca.

## 3 Ajustes, dentro Privacidad

El interruptor **Ayuda a mejorar nuestros modelos de IA** y el resto de la configuración de datos. Es el ajuste con más consecuencias de los tres y el que está peor señalizado.

Un **proyecto** es un espacio de trabajo cerrado, con su propio historial de conversaciones y su propia base de conocimiento. Es lo primero que deberías montar si usas Claude para algo que se repite, y lo que casi nadie monta.

## Qué cambia respecto a un chat suelto

Tres cosas. Las conversaciones del proyecto viven aparte y no se mezclan con el resto. Los archivos que subes están disponibles en **todas** las conversaciones de dentro, sin volver a adjuntarlos. Y las **instrucciones del proyecto** se aplican a todas ellas sin que las repitas.

## Las instrucciones son la mitad del valor

Son un campo de texto libre donde defines cómo quieres que te responda dentro de ese proyecto: el tono, desde qué punto de vista profesional, qué dar por sabido, qué no volver a preguntarte. Escrito una vez, sirve para siempre.

El error típico es dejarlas vacías y seguir explicando el contexto en cada chat. Si te oyes a ti mismo repitiendo una frase por tercera vez, esa frase va aquí.

## Cuántos puedes tener

En la cuenta **gratuita, cinco proyectos** como máximo. En los planes de pago son ilimitados, y además cambia lo que pasa por dentro con la base de conocimiento, que es el apartado siguiente.

---

En Team y Enterprise un proyecto se comparte con personas concretas o con toda la organización, y se les da permiso de ver o de editar.

Los archivos que subes a un proyecto no son adjuntos: son la base de conocimiento a la que Claude vuelve cada vez. Y aquí está la diferencia técnica más grande entre pagar y no pagar, que no aparece en ninguna tabla de precios.

## Qué pasa cuando no cabe

Si el material que has subido se acerca al límite de contexto, en las cuentas de pago entra en juego la **recuperación aumentada**: en vez de meterlo todo delante, Claude busca dentro y trae solo los trozos que hacen falta para lo que le has preguntado.

El efecto es que la capacidad del proyecto **se multiplica hasta por diez**. En la cuenta gratuita tienes conocimiento de proyecto básico, sin esa ampliación.

## Qué subir y qué no

Sube lo que sea referencia estable: guías de estilo, documentación, contratos tipo, ejemplos de trabajos tuyos bien hechos, glosarios. No subas lo que cambia cada semana, porque tendrás que acordarte de reemplazarlo y no te vas a acordar.

El caso que más rinde es el más aburrido: **tres o cuatro ejemplos de tu propio trabajo**. Es la forma más barata de que escriba como tú sin explicarle cómo escribes.

## Y los archivos, además, se ejecutan

Los archivos del proyecto también están accesibles desde el entorno de computación de Claude sin salir del contexto. Traducido: puede abrir tu hoja de cálculo y trabajar con ella de verdad, no solo leerla.

---

Un proyecto bien montado se nota en la primera respuesta. Si no se nota, es que las instrucciones están vacías.

Claude puede acordarse de ti entre conversaciones. Suena a magia y es una lista de notas que puedes abrir, leer, corregir y borrar. Que se pueda auditar es justo lo que la hace fiable.

## Cómo guarda, que no es como crees

No resume la conversación al terminar. **Va guardando temas sueltos mientras habláis**, y los actualiza. Si le dices que la entrega se ha movido al día 12, la siguiente conversación ya arranca con el día 12, no con el anterior.

## Qué se apunta

Contexto de trabajo: a qué te dedicas, en qué proyectos andas, tus responsabilidades, cómo prefieres que te escriban, tus manías técnicas y cómo enfocas el código. Es una memoria profesional, no un diario.

Por defecto **deja fuera lo sensible** —salud, religión, política y similares—. Si te interesa que sí lo guarde para algo concreto, se activa a mano en los ajustes.

## Cada proyecto tiene la suya

Un proyecto no comparte memoria con los demás ni con los chats de fuera: tiene su propio espacio y su propio resumen. Lo que aprende trabajando en un cliente no se le escapa en el chat de otro.

## Quién la tiene encendida

En **gratuito, Pro y Max viene encendida** de fábrica. En **Team y Enterprise viene apagada** y tiene que encenderla el dueño de la organización antes de que nadie pueda usarla.

---

Funciona además entre Chat y Cowork cuando este corre en la nube, así que lo que aprende en un sitio aparece en el otro.

13 · CORREGIR LO QUE RECUERDA

La memoria se equivoca. Guarda como permanente algo que dijiste de pasada, o se queda con una versión vieja de tu trabajo. Se arregla en dos minutos y casi nadie lo hace porque no sabe que se puede.

## El procedimiento

Abre **Ajustes, Memoria**. Ahí está la lista completa de temas que ha guardado sobre ti. Pulsa cualquiera para leerlo entero: verás exactamente la frase con la que te tiene fichado.

Con el **icono de editar** lo cambias, y con **Eliminar** lo borras. El cambio se aplica a las conversaciones siguientes, no a las que ya existen.

## Cuándo merece la pena entrar

Cuando notes que arrastra una suposición vieja: te habla de un proyecto que cerraste, da por hecho una herramienta que ya no usas, o insiste en un formato que pediste una vez. Eso no se arregla explicándoselo otra vez en el chat; se arregla borrando la nota.

## El uso que no se le da

También se puede **escribir a propósito**. Dile «acuérdate de que reviso todo antes de publicar y nunca quiero que dé por bueno un dato sin fuente» y esa frase se queda de guardia en todas las conversaciones futuras. Es lo más parecido a configurarlo sin tocar un ajuste.

---

Si trabajas con datos de clientes, entra una vez al mes. Es la única forma de saber qué se ha quedado guardado.

Un **artefacto** es contenido que Claude saca a una ventana aparte, fuera del hilo del chat. Deja de ser texto que has de copiar y pasa a ser una cosa que existe: un documento, un diagrama, una web de una página, una calculadora que funciona.

## Cuándo aparece uno

El criterio está publicado y es más concreto de lo que parece: cuando el contenido es **significativo y se sostiene solo, normalmente por encima de quince líneas**, y es algo que vas a querer editar, reutilizar o volver a mirar más tarde.

Por eso un párrafo de respuesta no sale en artefacto y una plantilla de correo sí. Si quieres forzarlo, pídelo: «sácamelo como artefacto» funciona.

## De qué tipos hay

Documentos en Markdown o texto plano, fragmentos de código, **páginas web de un solo archivo HTML**, imágenes SVG, diagramas y **componentes interactivos de React**. El último es el que sorprende: cosas que se pulsan y responden.

## Cómo se enciende

No es un ajuste propio. Vive dentro de **Ajustes, Capacidades, Ejecución de código y creación de archivos**. Si tienes Team o Enterprise, es el dueño quien lo enciende en Ajustes de la organización, Capacidades.

---

Para editar un artefacto de Markdown sin volver a pedirlo entero: selecciona el trozo y usa **Editar con Claude**. Cambia solo eso.

Los artefactos no se quedan atrapados en la conversación donde nacieron. Tienen su propio cajón y su propio botón de publicar, y esa es la parte que convierte la función en algo utilizable.

## El cajón

En la barra lateral hay una sección de **Artefactos** con todo lo que has creado. Es donde vuelves a buscar aquella plantilla de hace tres semanas sin acordarte de en qué chat la hiciste.

## Publicar

Cualquier artefacto tiene un botón de **Publicar** que lo saca del chat y lo deja accesible. Es la vía rápida para enseñarle algo a alguien sin mandarle una captura ni un archivo.

## El almacenamiento persistente

En los planes de pago, un artefacto puede guardar estado entre visitas, con un tope de **20 MB por artefacto**. La letra pequeña importa: la entrada es **solo texto**, nada de imágenes, archivos ni datos binarios.

Eso acota bien para qué sirve. Una lista, una encuesta, un contador, un documento que la gente edita. No un gestor de fotos.

---

Antes de publicar nada, léelo entero. Un artefacto publicado es contenido que sale de tu cuenta, y ya no lo controlas igual.

16 · EJECUCIÓN DE CÓDIGO Y ARCHIVOS

Con este interruptor, Claude deja de describirte la tabla y te la entrega hecha. Ejecuta código en un entorno aislado y devuelve el archivo terminado. Está en todos los planes y mucha gente no lo tiene encendido.

## Qué fabrica

Hojas de cálculo de Excel, presentaciones de PowerPoint, documentos de Word y **PDF**. Además saca visualizaciones de datos como imágenes PNG y ejecuta Python para analizar lo que le des.

La diferencia con pedirle «hazme una tabla» es que aquí no te describe la tabla: te devuelve el .xlsx con las fórmulas dentro.

## El límite que hay que saber

**30 MB por archivo**, y vale igual para lo que subes y para lo que te descargas. Si tu fuente pesa más, hay que partirla antes.

## Quién lo tiene

Todos los planes: gratuito, Pro, Max, Team y Enterprise, en web, escritorio y móvil. En **gratuito, Pro y Max viene encendido de fábrica y con acceso a red activado**. En Team y Enterprise viene encendido pero el dueño puede apagarlo o limitar la red para toda la organización.

---

Se enciende en Ajustes, Capacidades. Es el mismo interruptor que da los artefactos y las skills: uno solo, tres funciones.

17 · EL AVISO QUE SÍ IMPORTA

Cuando Claude puede ejecutar código y salir a la red, alguien puede aprovecharlo. El mecanismo se llama **inyección de prompt** y funciona así.

> Esto no es una advertencia genérica de las que se ponen por si acaso. Es lo que Anthropic documenta como riesgo real de la ejecución de código, y merece leerse dos veces.

## El ataque

Es posible que un actor malicioso **añada instrucciones de forma poco visible en archivos externos o en páginas web** para engañar a Claude y conseguir que ejecute código dañino o que **saque datos de tu contexto hacia terceros**.

## Qué hay puesto de barrera

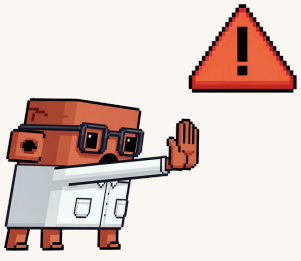
El entorno está **aislado y en caja de arena**, y hay un **clasificador de inyección de prompt** que intenta detectar la manipulación. Además, el acceso a red se puede desactivar del todo, y las organizaciones pueden dejar en lista blanca solo los dominios que aprueben.

## La recomendación oficial

Empezar **con el acceso a red desactivado**, habilitar los gestores de paquetes que hagan falta y solo después ir ampliando dominios. Si trabajas con material sensible, esa es la postura por defecto, no la paranoica.

---

La regla que resume todo: lo que viene de fuera es dato, nunca orden. Vale para Claude y vale para ti.



18 · MODO INCÓGNITO

Una conversación que no deja rastro en tu cuenta. Está en todos los planes y se abre con un icono que la mayoría no ha visto nunca.

## Cómo se abre

Al empezar un chat nuevo **fuera de un proyecto**, pulsa el **icono del fantasma, arriba a la derecha**. Sabrás que estás dentro porque la ventana se pone con **borde negro** y aparece la etiqueta **Chat incógnito** arriba a la izquierda. Se cierra con la equis.

## Qué hace y qué no

No se guarda en tu historial, no alimenta la memoria, **no se usa para entrenar** y no aparece en los resúmenes mensuales. A cambio, tampoco **puede leer** lo que la memoria ya sabe de ti: entra en frío.

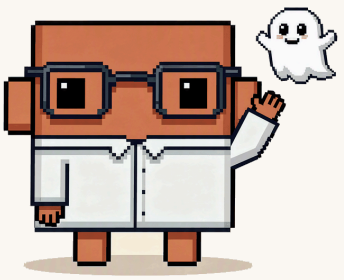
Y no se puede convertir en un chat normal a mitad. Si lo empiezas en incógnito, se queda ahí.

## La letra pequeña

«No se guarda» no significa que desaparezca al instante. Se retienen **30 días por defecto**, o más si tu organización tiene una política de retención propia. En Team y Enterprise entran además en las exportaciones de datos y en la API de cumplimiento.

---

Sirve para lo que parece: probar algo raro, tratar un dato puntual que no quieres fichado, o no ensuciar la memoria con una consulta que no te representa.



19 · EL AJUSTE DE PRIVACIDAD

El interruptor que decide si tus conversaciones se usan para mejorar los modelos vive fuera de los menús que se miran a diario, y por eso casi nadie lo ha abierto nunca. Aquí está la ruta exacta.

## Dónde está

**Ajustes, Privacidad**, y dentro el interruptor **Ayuda a mejorar nuestros modelos de IA**. La ruta es la misma en navegador, escritorio y móvil.

## A qué afecta

A los productos de consumo: **Claude gratuito, Pro, Max y las sesiones de Claude Code**. Si lo tienes encendido, tus chats y tus sesiones de programación se usan para entrenar modelos futuros.

## Qué pasa al apagarlo

No se usarán **nuevos** chats ni sesiones para entrenar. Lo que ya hubiera entrado en un entrenamiento empezado, o en un modelo ya entrenado, sigue dentro: eso no se puede deshacer y está dicho así en la fuente.

## La excepción

Si los clasificadores de seguridad marcan una conversación tuya, **puede seguir usándose** para mejorar los modelos internos de confianza y seguridad, tengas el interruptor como lo tengas.

---

No hay respuesta correcta universal. Lo que no vale es no saber cómo lo tienes.

Los plazos están publicados y son concretos. Merece la pena tenerlos a mano, porque son la respuesta a la pregunta que hace todo el mundo antes de meter algo de un cliente.

## 30 días Un chat que borras

Desaparece **de tu historial al instante**, y de los sistemas de almacenamiento del servidor **en un plazo de 30 días**. No es inmediato, pero tiene fecha.

## 5 años Si dejas activada la mejora del modelo

Tus datos pueden retenerse **en formato despersonalizado hasta cinco años** dentro de las canalizaciones de entrenamiento. Este es el número que sorprende a casi todo el mundo.

## 2 años Si algo se marca como incumplimiento

Cuando los sistemas detectan una violación de la política de uso, se retienen **entradas y salidas hasta dos años**.

## 7 años Las puntuaciones de esa marca

Las puntuaciones de clasificación de confianza y seguridad se guardan **hasta siete años**. Son metadatos, no la conversación, pero son siete años.

21 · LA REGLA DE ORO

Anthropic publica una guía de prompts y tiene una regla que resume el resto. No es un truco: es la vara de medir con la que se juzga cualquier petición antes de mandarla.

## La regla

**Enséñale tu prompt a un compañero que no tenga contexto de la tarea y pídele que lo siga. Si él se lía, Claude también.** Eso es todo. Si necesitas explicárselo de viva voz para que se entienda, lo que falta va dentro del prompt.

## La analogía que usan

Piensa en Claude como en **un empleado brillante recién llegado**: sabe muchísimo, pero no conoce vuestras normas ni vuestra forma de trabajar. Cuanto más preciso seas con lo que quieres, mejor sale.

## Lo de «por encima de lo esperado»

Aquí hay un detalle contraintuitivo. Si quieres que se estire, **hay que pedirlo explícitamente**; no lo deduce de una petición vaga. Su ejemplo es literal: «Crea un panel de analítica» sale mediocre. Esto sale bien:

```
Crea un panel de analítica. Incluye tantas funciones e
interacciones relevantes como puedas. Ve más allá de lo
básico y haz una implementación completa.
```

## Y cuando el orden importa

Si los pasos tienen que ir en un orden concreto, o tienen que estar todos, **dáselos como lista numerada**. En prosa se pierde alguno; en lista, no.

---

Casi todas las respuestas decepcionantes vienen de peticiones que valdrían para cualquiera. Media de todos los casos posibles, que es lo que has pedido.

22 · EXPLICA EL PORQUÉ

Este es el consejo con mejor relación entre esfuerzo y resultado de toda la guía oficial, y el que menos gente aplica. Dar el motivo de una instrucción funciona mejor que dar la instrucción.

## El ejemplo de la documentación

Menos eficaz:

```
NUNCA uses puntos suspensivos.
```

Más eficaz:

```
Tu respuesta la va a leer en voz alta un motor de
síntesis de voz, así que nunca uses puntos suspensivos:
el motor no sabría cómo pronunciarlos.
```

## Por qué funciona

Porque con el motivo delante **generaliza**. La primera versión prohíbe una cosa. La segunda le explica el mundo en el que estás, así que también evita las abreviaturas raras, los símbolos que no se leen y los emojis, sin que hayas tenido que enumerarlos.

Está dicho tal cual en la fuente: Claude es lo bastante listo como para generalizar a partir de la explicación.

## Cómo se aplica a lo tuyo

Cada vez que escribas una prohibición o una exigencia, añádele la coletilla **«porque…»**. «Máximo 5 puntos» pasa a ser «máximo 5 puntos porque va en una diapositiva y no cabe más». La diferencia se nota en la primera respuesta.

---

Es la traducción exacta de lo que haces con una persona: no le das órdenes sueltas, le explicas para qué es.

Los ejemplos son, según la documentación, **una de las formas más fiables** de dirigir el formato, el tono y la estructura de la salida. Y hay número recomendado, que casi nadie conoce.

## El número

**Entre tres y cinco ejemplos** para el mejor resultado. Uno solo se queda corto y el modelo copia sus casualidades; más de cinco empieza a no aportar.

## Las tres condiciones

**Relevantes**: que se parezcan de verdad a tu caso, no a una versión limpia de tu caso. **Diversos**: que cubran los casos raros y varíen lo bastante como para que no aprenda un patrón que no querías. **Estructurados**: envueltos en etiquetas para que no los confunda con las instrucciones.

## Cómo se envuelven

```
<examples>
  <example>
    Entrada: ...
    Salida esperada: ...
  </example>
  <example>
    Entrada: ...
    Salida esperada: ...
  </example>
</examples>
```

## El atajo que ahorra la mitad del trabajo

Puedes **pedirle a él que evalúe tus ejemplos** —si son relevantes y variados— o que genere más a partir de los que ya tienes. Está recomendado en la propia guía y es la forma más rápida de pasar de dos ejemplos flojos a cinco buenos.

---

Si el resultado sale con un vicio raro que no habías pedido, mira tus ejemplos: normalmente lo tienen los tres.

Cuando un prompt mezcla instrucciones, contexto, ejemplos y datos variables, empieza a confundirlos. La solución oficial es de una simplicidad que da rabia: etiquetas tipo XML.

## Qué se gana

Envolver cada tipo de contenido en su propia etiqueta **reduce la mala interpretación**. Deja de tener que adivinar dónde acaba tu instrucción y dónde empieza el material.

```
<instrucciones>
Resume el documento en 5 puntos para el comité.
</instrucciones>

<contexto>
La reunión es mañana y hay que decidir sobre el proveedor.
</contexto>

<entrada>
... aquí el documento ...
</entrada>
```

## Las dos reglas

**Nombres consistentes y descriptivos** entre todos tus prompts: si hoy usas «entrada» y mañana «input», pierdes la ventaja. Y **anida cuando haya jerarquía natural**: varios documentos dentro de una etiqueta de documentos, cada uno con su índice.

## No hace falta que sea XML de verdad

No se valida contra nada ni tiene que estar bien formado. Son marcas visuales para separar bloques. Cualquier nombre de etiqueta que se entienda vale.

---

Es la técnica que más rinde en prompts largos y la que menos se usa fuera de la gente que programa contra la API.

Cuando trabajas con entradas grandes —a partir de unos 20.000 tokens—, **el orden de las piezas cambia el resultado**. Y no es un matiz.

> Este apartado vale para cualquiera que pegue un documento en el chat, no solo para quien programa. Es el dato más accionable de toda la documentación de prompts.

## El dato

Pon los documentos largos **arriba del todo**, por encima de tu pregunta, de las instrucciones y de los ejemplos. Según las pruebas de Anthropic, dejar la pregunta al final **puede mejorar la calidad de la respuesta hasta un 30%**, sobre todo con entradas complejas de varios documentos.

Traducido a lo que hace todo el mundo: pegar el documento y **debajo** escribir qué quieres. Casi todos hacemos lo contrario, escribimos la petición y luego pegamos. Eso cuesta calidad.

## Con varios documentos

Envuelve cada uno con su etiqueta y su metadato de origen, para que pueda citar de cuál viene cada cosa:

```
<documentos>
  <documento indice="1">
    <origen>informe-anual-2025.pdf</origen>
    <contenido>
      ...
    </contenido>
  </documento>
</documentos>

Ahora, con eso delante: ¿qué tres riesgos repiten
los dos informes?
```

---

Regla de bolsillo: material arriba, pregunta abajo. Cuesta cero y es lo más rentable que vas a leer aquí.

La documentación oficial incluye tres bloques de texto para pegar cuando el modelo se pasa de listo. No son opiniones de nadie: son los remedios que publica Anthropic para tres vicios concretos.

## Cuando sobreingeniería

Síntoma: le pides un arreglo pequeño y te devuelve tres archivos nuevos, una capa de abstracción y opciones de configuración que nadie pidió. El freno:

```
Evita la sobreingeniería. Haz solo los cambios pedidos o
claramente necesarios. Un arreglo de un fallo no necesita
limpiar el código de alrededor. No crees ayudantes ni
abstracciones para operaciones de una sola vez. No diseñes
para requisitos hipotéticos futuros.
```

## Cuando trampea para que pasen los tests

Síntoma: los tests pasan y el código no sirve, porque ha metido valores fijos que solo funcionan con esos casos. El freno:

```
Implementa una solución general que funcione con todas las
entradas válidas, no solo con los casos de prueba. No metas
valores fijos. Los tests están para verificar, no para
definir la solución. Si la tarea es inviable o algún test
está mal, dímelo en vez de rodearlo.
```

## Cuando habla de código que no ha abierto

Es el vicio más caro, porque suena convincente. El freno, tal cual lo publican:

```
Nunca especules sobre código que no hayas abierto. Si te
nombro un archivo, DEBES leerlo antes de responder.
preguntas sobre el proyecto.
```

---

Los tres se pegan al final de tus instrucciones y se quedan. En Claude Code, su sitio es el CLAUDE.md.

Claude Code es Claude trabajando **dentro de tu ordenador**: lee tus archivos, los edita, ejecuta comandos y ve el resultado. La diferencia con el chat no es de potencia, es de acceso.

## Lo que cambia de verdad

En el chat le pegas un archivo y te devuelve texto que copias. Aquí abre el archivo, lo cambia, ejecuta los tests, **ve que fallan y lo vuelve a intentar**. El bucle se cierra sin ti en medio.

## Dónde vive

Es una herramienta de terminal, pero no solo: hay extensión de **VS Code** y de **JetBrains**, aplicación de escritorio, sesiones en la web y control remoto desde el móvil. La terminal es el sitio donde todo funciona seguro.

## Qué plan hace falta

Está incluido **a partir de Pro**. También entra en Max, Team y Enterprise. Con la cuenta gratuita no. Y se puede usar con una cuenta de la Consola pagando por consumo, que es otra cosa.

## Para quién no es

Si no tocas archivos ni ejecutas nada, no lo necesitas: el chat te da lo mismo con menos ceremonia. Claude Code se paga solo cuando hay un proyecto con muchos archivos, o una tarea que se repite y se puede automatizar.

---

El resto de esta parte es la parte del manual que casi nadie lee y donde está casi todo lo que la hace útil.

La instalación es de un comando y la autenticación no necesita clave de API si ya pagas una suscripción. Estos son los pasos y los comandos exactos.

## Instalar

Una vez instalado, el binario nativo se puede reinstalar o fijar a una versión concreta:

```
claude install
claude install 2.1.240
```

## Entrar

La sesión se abre escribiendo **claude** en la carpeta del proyecto. La primera vez pedirá identificarse. Desde la propia terminal:

```
claude auth login
claude auth status
claude auth logout
```

## Cuando algo falla

Antes de buscar en foros, ejecuta el diagnóstico. Es de solo lectura y no cambia nada:

```
claude doctor
```

## Para automatizar

Si vas a llamarlo desde un script o desde integración continua, hace falta un token de larga duración en vez de una sesión interactiva:

```
claude setup-token
```

---

Instalar Claude Code **no crea ningún fichero de ajustes**. Todo lo que veas en tu máquina lo has creado tú o lo ha creado él al guardarte una decisión.

El modo de arranque se fija con la clave **defaultMode**, y se cambia en caliente durante la sesión.

> El modo decide qué hace Claude **sin preguntarte**. Los nombres son los exactos porque también se escriben en los ajustes.

## 1 default

Pregunta la primera vez que usa cada herramienta. En la interfaz aparece como **Manual**.

## 2 acceptEdits

Acepta solo ediciones de archivos y comandos corrientes —**mkdir, touch, mv, cp**— dentro del directorio de trabajo.

## 3 plan

Lee y ejecuta comandos de solo lectura para explorar, **pero no toca tu código fuente**.

## 4 auto

Aprueba solo, con comprobaciones en segundo plano que verifican que lo que hace encaja con lo que pediste.

## 5 dontAsk

Al revés que auto: **deniega todo** salvo lo preaprobado con reglas.

## 6 bypassPermissions

Se salta las preguntas, incluidas las de rutas protegidas como **.git** y **.claude**. Solo en contenedor o máquina virtual.

Los modos son la brocha gorda. Las **reglas** son el pincel fino: dicen qué se permite, qué se pregunta y qué se prohíbe, herramienta por herramienta. Se editan con el comando de permisos o a mano en los ajustes.

## La sintaxis

Solo hay dos formas: **Herramienta** o **Herramienta(especificador)**. Sin paréntesis, la regla cubre todos los usos de esa herramienta.

```
Bash              todos los comandos de shell
Read              todas las lecturas de archivo
WebFetch          todas las descargas de web
```

Con especificador se acota:

```
Bash(npm run build)             ese comando exacto
Read(./.env)                    leer ese archivo
WebFetch(domain:example.com)    solo ese dominio
```

## Los comodines

Un asterisco dentro de una regla de Bash **casa con cualquier texto, espacios incluidos**, así que una sola regla cubre una familia entera. Una regla sin asterisco casa con un comando exacto y solo con ese.

```
Bash(git*)      cualquier comando de git
```

## El sitio donde se guardan

Cuando respondes «sí, y no vuelvas a preguntar» a una petición de permiso, eso se guarda como regla de permitir en **.claude/settings.local.json**. No es magia: es un fichero que puedes abrir y limpiar.

---

Las listas de reglas de varios ficheros **se suman**, no se pisan. Tu fichero personal puede añadir sin quitarle nada al del equipo.

Hay una segunda forma de regla que casi nadie usa y que resuelve casos que la primera no puede: cazar por el valor de un parámetro de entrada.

## Cómo es

Las reglas de **denegar** y de **preguntar** pueden casar con un parámetro de primer nivel de cualquier herramienta, con la forma **Herramienta(parametro:valor)**.

```
Agent(model:opus)              subagentes que pidan Opus
Agent(isolation:worktree)      los que pidan un worktree
Bash(run_in_background:true)   comandos en segundo plano
```

## Las reglas del juego

Cada regla nombra **un** parámetro: para vigilar dos, se escriben dos reglas. El asterisco vale como comodín en el valor, y sin él la coincidencia es exacta. Un parámetro que el modelo no ponga **nunca casa**, así que una regla con comodín no atrapa las llamadas que lo omiten.

## Lo que no se puede hacer, y por qué

No se puede cazar el campo principal de contenido de una herramienta: **command** en Bash, **file_path** en Read, Edit y Write, **path** en Grep y Glob, **url** en WebFetch.

El motivo es bueno: una regla como **Bash(command:rm *)** se esquivaría con un comando compuesto, así que Claude Code **la ignora y avisa al arrancar**. La forma correcta es Bash(rm *).

---

Las reglas de permitir siguen usando el especificador propio de cada herramienta: permitir un parámetro no demuestra que la llamada entera sea segura.

Hay un paso de seguridad que descoloca la primera vez: has puesto reglas en el proyecto y no se aplican. No están rotas. Falta confiar en la carpeta.

## El problema que resuelve

Si te descargas un repositorio de un desconocido, ese repositorio puede traer un **.claude/settings.json** con reglas de permitir y con hooks. Si se aplicaran solos al abrir la carpeta, clonar un repositorio sería ejecutar código ajeno.

Por eso las reglas de permitir del fichero compartido y los hooks del proyecto **esperan a que confíes en la carpeta**.

## Tu fichero local es distinto

Las reglas de **.claude/settings.local.json** se aplican sin ese paso **mientras el fichero no esté seguido por git**, porque es tuyo y no del repositorio. Si alguien lo mete en el control de versiones, pasa a necesitar confianza como el compartido.

## El fichero que Claude escribe para sí

Las decisiones de confianza se guardan, junto con tu sesión y la configuración de MCP, en **~/.claude.json**. Es un quinto fichero que mantiene él y que tú no tienes que editar.

---

Si un ajuste del equipo no te hace efecto, esta es la primera sospecha, antes que un error de sintaxis.

Hay más de setenta comandos con barra. Estos son los que se usan de verdad todos los días, agrupados por para qué sirven.

## Contexto /context · /compact · /clear

**/context** dibuja lo que ocupa la ventana ahora mismo, en cuadrícula de colores. **/compact** resume lo viejo para hacer sitio. **/clear** empieza de cero. El primero se mira antes de usar los otros dos.

## Trabajo /plan · /diff · /code-review

**/plan** entra en modo plan para cambios grandes. **/diff** abre el visor de cambios sin confirmar. **/code-review**, con alias **/review**, revisa el diff o un PR buscando fallos.

## Ajustes /model · /effort · /permissions

**/model** cambia de modelo y lo guarda por defecto. **/effort** sube o baja el esfuerzo. **/permissions** abre las reglas de permitir, preguntar y denegar.

## Memoria /memory · /init · /doctor

**/memory** abre los CLAUDE.md y la memoria automática. **/init** genera un CLAUDE.md leyendo tu proyecto. **/doctor**, con alias **/checkup**, diagnostica y arregla la instalación.

## Extras /agents · /hooks · /mcp · /plugin

Los cuatro paneles de configuración: subagentes, hooks, servidores MCP y plugins. Es más rápido mirarlos aquí que buscar el fichero.

Los que no se usan a diario pero resuelven un problema concreto cuando aparece. Merece la pena saber que existen.

## 1 /btw

Pregunta algo al margen **sin que entre en el historial de la conversación**. Es el que te ahorra abrir otra sesión para una duda tonta que ensuciaría el contexto.

## 2 /branch y /fork

**/branch** saca una rama de la conversación actual para probar otro camino. **/fork** copia la conversación entera a una sesión nueva en segundo plano.

## 3 /rewind

Vuelve atrás en la conversación. Cuando se ha ido por un camino equivocado, deshacer sale más barato que explicar.

## 4 /usage

Con alias **/cost**. Lo que llevas gastado. Es la respuesta honesta a «¿por qué me he quedado sin límite?».

## 5 /export y /copy

**/export** saca la conversación como texto plano; **/copy** copia la última respuesta al portapapeles.

## 6 /fewer-permission-prompts

Repasa tus transcripciones, ve qué comandos apruebas siempre y te propone una lista de permitidos. Quita la mitad de las preguntas de un tirón.

Un **CLAUDE.md** es un fichero de texto con instrucciones que Claude lee al empezar cada sesión. Es la diferencia entre explicarle tu proyecto todos los días y explicárselo una vez.

## Dónde puede ir, de más general a más concreto

```
~/.claude/CLAUDE.md      tú, en todos tus proyectos
./CLAUDE.md              el proyecto, para todo el equipo
./.claude/CLAUDE.md      lo mismo, otro sitio válido
./CLAUDE.local.md        tú, solo en este proyecto
```

El de política gestionada por la organización va aparte: en macOS, **/Library/Application Support/ClaudeCode/CLAUDE.md**; en Linux y WSL, **/etc/claude-code/CLAUDE.md**.

## Cómo se cargan

Se lee el de tu carpeta y el de **todas las carpetas por encima**. No se pisan: **se concatenan**, de la raíz hacia abajo, así que lo más cercano a donde arrancaste se lee el último. Dentro de cada carpeta, el CLAUDE.local.md va después del CLAUDE.md.

Los de las subcarpetas no se cargan al arrancar: entran **cuando Claude lee archivos de esa subcarpeta**.

## Para empezar el tuyo

El comando **/init** analiza el proyecto y escribe un primer CLAUDE.md con los comandos de compilar, de test y las convenciones que encuentre. Si ya existe uno, propone mejoras en vez de machacarlo.

---

Para comprobar cuáles se han cargado de verdad en esta sesión: **/context**, y mira la lista de ficheros de memoria.

De ahí salen las tres reglas de cómo se escribe, que están publicadas y son concretas.

> El dato que lo explica todo: el CLAUDE.md **no forma parte del prompt de sistema**. Se entrega como un mensaje de usuario justo después. Claude lo lee y lo intenta seguir, pero **no hay garantía de cumplimiento estricto**, y menos con instrucciones vagas.

## Tamaño

**Menos de 200 líneas.** Un fichero largo consume más contexto y **reduce la adherencia**: cuanto más le metes, menos caso hace a cada cosa. Por encima de 4 MiB directamente lo salta.

## Concreción

Escribe instrucciones que se puedan verificar. Los ejemplos son de la propia documentación: «Usa indentación de 2 espacios» en vez de «Formatea bien el código». «Ejecuta npm test antes de hacer commit» en vez de «Prueba tus cambios».

## Coherencia

**Si dos reglas se contradicen, elige una arbitrariamente.** Por eso hay que repasar el fichero de vez en cuando y quitar lo caducado. Es el fallo que más desconcierta: no es que no obedezca, es que le has dado dos órdenes.

## Qué va aquí y qué no

Aquí van los hechos que valen en **todas** las sesiones: comandos de compilación, convenciones, dónde está cada cosa, reglas de siempre. Si una entrada es un procedimiento de varios pasos, o solo importa en una parte del proyecto, su sitio es una skill o una regla por ruta.

---

Un truco poco conocido: los comentarios HTML de bloque se eliminan antes de meterlo en contexto. Sirven para dejar notas a humanos sin gastar tokens.

37 · REGLAS POR RUTA

Cuando el CLAUDE.md empieza a crecer, la solución no es recortarlo: es partirlo en reglas que solo se cargan cuando hacen falta. Vive en **.claude/rules/**.

## Cómo se monta

Ficheros Markdown, uno por tema, con nombre descriptivo. Se descubren **recursivamente**, así que se pueden ordenar en subcarpetas.

## La parte que importa

Una regla **sin** el campo de rutas se carga al arrancar, con la misma prioridad que el CLAUDE.md del proyecto. Una regla **con** rutas solo entra cuando Claude toca archivos que casan con el patrón:

```
---
paths:
  - "src/api/**/*.ts"
---

# Reglas de la API

- Validar siempre la entrada.
- Usar el formato de error estándar.
```

## Las tuyas, en todos los proyectos

En **~/.claude/rules/** van tus manías personales, y se cargan antes que las del proyecto: las del proyecto tienen más prioridad.

---

La diferencia con una skill: una regla se carga sola cuando toca un archivo; una skill se carga cuando la tarea la pide.

Además del CLAUDE.md que escribes tú, hay una memoria que **escribe Claude solo** a partir de tus correcciones. Está encendida de fábrica y mucha gente no sabe que existe.

## Qué se apunta, y qué no

Cuatro tipos, y los marca con una etiqueta en el propio fichero. **user**: tu papel, tu experiencia, cómo trabajas. **feedback**: correcciones que le has dado y enfoques que has confirmado. **project**: trabajo en marcha y decisiones que no se deducen del código. **reference**: dónde está lo de fuera, un panel o un gestor de incidencias.

Y salta a propósito lo que **puede deducir del propio código** —arquitectura, rutas, cómo se arregló un fallo— y lo que ya diga tu CLAUDE.md. No guarda algo cada sesión: decide si serviría en una conversación futura.

## Dónde está

```
~/.claude/projects/<proyecto>/memory/
  MEMORY.md          el índice, se carga siempre
  user_role.md       un recuerdo
  feedback_tests.md  otro
```

Del índice se cargan **las primeras 200 líneas, o los primeros 25 KB**, lo que llegue antes. Lo que pase de ahí no entra. Los ficheros de tema no se cargan al arrancar: los abre cuando los necesita.

## Apagarla o moverla

Se apaga con **autoMemoryEnabled** en los ajustes, o con la variable **CLAUDE_CODE_DISABLE_AUTO_MEMORY** a 1. Se mueve con **autoMemoryDirectory**.

---

Es Markdown normal: se puede abrir con **/memory**, leer, corregir y borrar. Merece la pena mirarlo una vez.

Claude Code lee los ajustes de cuatro sitios, y cada uno tiene un alcance distinto: tú, este proyecto, todo el equipo o toda la organización. Confundirlos es el motivo número uno de «he puesto el ajuste y no hace nada».

## 1 ~/.claude/settings.json · Usuario

Tú, en **todos los proyectos** de esta máquina. Es el sitio de tus preferencias: tema, modelo por defecto, tus propias reglas de permiso.

## 2 .claude/settings.json · Proyecto compartido

Todo el que abra Claude Code en esa carpeta. En un repositorio de git, **se hace commit** para que le llegue al equipo. Aquí van permisos del equipo, hooks, plugins y las variables que el proyecto necesita.

## 3 .claude/settings.local.json · Proyecto local

Tú, **solo en este proyecto**. Es donde Claude guarda tus «no vuelvas a preguntar». Si lo crea él, lo mantiene fuera de git solo; si lo creas tú a mano, el .gitignore es cosa tuya.

## 4 managed-settings.json · Gestionado

Lo que despliega tu organización. **Nada de lo que pongas lo sobrescribe**, salvo un puñado de excepciones de seguridad. Aquí va la política de cumplimiento.

Cuando la misma clave está en varios sitios, se aplica la del nivel más alto que la define. El orden está publicado y merece aprendérselo, porque explica casi todos los ajustes que «no funcionan».

## De mayor a menor

```
1  Ajustes gestionados      (la organización)
2  Argumentos de línea de comandos
3  .claude/settings.local.json
4  .claude/settings.json
5  ~/.claude/settings.json
```

Lo contraintuitivo está en el 4 y el 5: **el fichero del equipo gana al tuyo personal**. Si tu ajuste de siempre deja de aplicarse en un proyecto concreto, mira si el equipo lo fija.

## Las listas no se pisan

Las claves que son listas —**permissions.allow** es el caso típico— **se combinan** en vez de elegir una. Así cada fichero añade sin quitar lo de los demás. Las listas de modelos son la excepción y siguen sus propias reglas.

## Las variables de entorno no son un nivel

Aquí está la trampa fina. No van en la escalera de arriba: **se decide pareja por pareja**. **ANTHROPIC_MODEL** exportada en tu shell manda sobre la clave **model** de cualquier fichero. **ANTHROPIC_DEFAULT_MODEL** solo se aplica **si ningún fichero** define model.

---

Para un solo arranque, sin tocar ningún fichero: **claude --settings** con una ruta o con JSON en línea.

Cuatro sospechosos, en el orden en que hay que mirarlos. Sale más rápido que buscar en internet.

## 1 No has confiado en la carpeta

Las reglas de permitir del fichero compartido y los hooks del proyecto **esperan al paso de confianza**. Es la causa más frecuente y la que menos se sospecha, porque no da error: simplemente no pasa nada.

## 2 Lo pisa un nivel superior

El fichero del equipo gana al tuyo, y lo gestionado gana a todo. Comprueba en qué nivel está puesta la clave y si hay otro por encima que la fije.

## 3 Hay una variable de entorno delante

Si tienes **ANTHROPIC_MODEL** exportada, tu clave model no manda, y no lo verás en ningún fichero. Mira el shell antes que los ajustes.

## 4 El JSON está roto

Una coma de más y el fichero entero deja de leerse. **/doctor** lo detecta. También conviene comprobar el nombre exacto de la clave: no hay corrector.

Una **skill** es una carpeta con un fichero **SKILL.md** dentro. Escribes tu método una vez y Claude lo abre él solo cuando la conversación lo pide, o tú lo llamas escribiendo la barra y su nombre.

## Cuándo hacerse una

El criterio oficial es útil: cuando llevas pegando las mismas instrucciones, la misma lista de comprobación o el mismo procedimiento de varios pasos en el chat. O cuando **un apartado de tu CLAUDE.md ha dejado de ser un hecho y se ha convertido en un procedimiento**.

## La diferencia con el CLAUDE.md

El CLAUDE.md se carga entero **en todas** las sesiones y gasta contexto siempre. El cuerpo de una skill **solo se carga cuando se usa**. Por eso un material de referencia largo no cuesta casi nada hasta que hace falta.

## Los comandos son skills

Esto confunde a mucha gente: **los comandos personalizados se han fusionado con las skills**. Un fichero en **.claude/commands/deploy.md** y una skill en **.claude/skills/deploy/SKILL.md** crean los dos el comando /deploy y funcionan igual.

Los ficheros de commands que ya tengas siguen funcionando. Las skills añaden lo que aquellos no tienen: una carpeta para ficheros de apoyo, control de quién las invoca, y que Claude pueda cargarlas solo.

---

Si hay un fichero de comando y una skill con el mismo nombre, **gana la skill**.

El sitio donde la guardas decide quién puede usarla. Son cuatro, y el orden de prioridad tiene una sorpresa.

## Las cuatro ubicaciones

```
~/.claude/skills/<nombre>/SKILL.md    todos tus proyectos
.claude/skills/<nombre>/SKILL.md      solo este proyecto
<plugin>/skills/<nombre>/SKILL.md     donde esté el plugin
Empresa: por ajustes gestionados      toda la organización
```

## Quién gana

Aquí está lo contraintuitivo, y va al revés que en los ajustes: **empresa gana a personal, y personal gana a proyecto**. Si tienes una skill llamada deploy en tu carpeta personal y otra en el proyecto, **/deploy ejecuta la tuya personal**.

Una skill de cualquiera de esos niveles también sustituye a una de las que vienen de fábrica con el mismo nombre. Pero **no sustituye a sus alias**: si pones una skill llamada code-review, /code-review ejecuta la tuya y /review sigue ejecutando la de fábrica.

## Los plugins no chocan

Las skills de plugin usan el espacio de nombres **plugin:skill**, así que conviven con las tuyas sin pisarse.

## Quitar una sin borrarla

Para conservar una skill pero que Claude deje de invocarla por su cuenta, se le pone **disable-model-invocation: true** en el frontmatter, o el valor «user-invocable-only» en la clave **skillOverrides** de los ajustes si no quieres tocar el fichero.

---

El nombre de la carpeta es el comando que escribes. Elígelo corto.

El SKILL.md tiene dos partes: el frontmatter en YAML entre guiones, que dice **cuándo** usarla, y el Markdown de debajo, que dice **qué hacer**. Todos los campos son opcionales; solo el de descripción está recomendado.

---

## description — Lo único que de verdad importa

Qué hace y cuándo usarla: es lo que Claude lee para decidir si la carga. **Pon el caso de uso principal el primero**, porque el texto se corta a los 1.536 caracteres en el listado. Si lo omites, usa el primer párrafo del cuerpo.

---

## when_to_use — Contexto extra para el disparo

Frases que deberían activarla, ejemplos de peticiones. Se añade a la descripción y **cuenta para el mismo tope** de 1.536 caracteres.

---

## allowed-tools — Permisos por adelantado

Herramientas que puede usar sin preguntar durante el turno en que se invoca la skill. **El permiso se borra con tu siguiente mensaje**: no se queda abierto.

---

## context: fork — Que corra en un subagente

Ejecuta la skill en un contexto aparte, de forma que su trabajo no ensucie la conversación principal. Con **agent** eliges qué tipo de subagente.

---

## model y effort — Cambiar de motor solo para esto

Modelo y nivel de esfuerzo mientras la skill está activa. **No se guarda en los ajustes**: al mensaje siguiente vuelve el de la sesión.

---

## paths — Que solo aparezca donde toca

Patrones de rutas. Con esto puesto, Claude solo la carga sola cuando trabaja con archivos que casan.

El ejemplo de la documentación oficial, que además enseña el truco más útil del formato: meter la salida de un comando **antes** de que Claude lea nada.

## Crear la carpeta

```
mkdir -p ~/.claude/skills/resumir-cambios
```

## Escribir el SKILL.md

En **~/.claude/skills/resumir-cambios/SKILL.md**:

```
---
description: Resume los cambios sin commitear y señala lo
  arriesgado. Úsala cuando pregunten qué ha cambiado o
  pidan revisar el diff.
---

## Cambios actuales

!`git diff HEAD`

## Tu tarea

Resume lo de arriba en tres líneas y señala lo que pueda
romper algo en producción.
```

## El detalle que hay que ver

La línea con la exclamación y el comando entre comillas invertidas **ejecuta el comando y mete su salida en el prompt** antes de que Claude lo lea. Por eso la respuesta se basa en tu diff real y no en lo que él pueda suponer de los archivos que tenga abiertos.

---

Ese es todo el mecanismo. Una skill útil son quince líneas; las de doscientas suelen ser un CLAUDE.md disfrazado.

Un **hook** es un comando tuyo que se ejecuta en un momento fijo del ciclo de vida de la sesión. La diferencia con una instrucción es total: una instrucción es contexto, un hook **se ejecuta pase lo que pase**.

## Para qué son de verdad

La documentación lo dice sin rodeos: si una instrucción tiene que aplicarse en un punto concreto —antes de cada commit, después de cada edición—, **escríbela como hook, no como CLAUDE.md**. El CLAUDE.md se puede desobedecer; un hook no.

Casos típicos: pasar el formateador después de cada edición, bloquear comandos peligrosos antes de que corran, avisarte por notificación cuando termine, registrar en un fichero qué ha tocado.

## De qué tipo pueden ser

Cinco. **command** ejecuta un comando de shell, que es el 90% de los casos. **http** llama a una URL. **mcp_tool** invoca una herramienta de un servidor MCP. **prompt** le pregunta a un modelo. **agent** lanza un subagente para que decida.

## Dónde se configuran

En los mismos ficheros de ajustes: los tuyos en **~/.claude/settings.json**, los del proyecto en **.claude/settings.json**, los personales de un proyecto en el local. También pueden venir dentro de un plugin, o en el frontmatter de una skill o de un subagente.

---

Para ver los que tienes puestos y comprobar que se cargan: el comando **/hooks**.

Hay unos treinta eventos documentados. Estos son los que se usan en el 95% de los casos reales; el resto están en el apéndice.

## 1 PreToolUse

Antes de que una herramienta se ejecute. **Puede bloquearla.** Es el evento del que cuelga cualquier control de seguridad: revisar el comando antes de que corra.

## 2 PostToolUse

Después de que una herramienta termine bien. El sitio del formateador, del linter y de los tests automáticos tras cada edición.

## 3 PostToolUseFailure

Después de que una herramienta **falle**. Sirve para meterle contexto útil sobre por qué ha fallado en vez de dejarle adivinar.

## 4 UserPromptSubmit

Antes de que procese lo que acabas de escribir. **Puede bloquear.** Con él se inyecta contexto fresco en cada mensaje, o se para algo que no debería mandarse.

## 5 SessionStart y SessionEnd

Al empezar y al terminar. El de inicio distingue si vienes de arrancar, de reanudar, de limpiar o de compactar, y el de fin, por qué se acabó.

## 6 Stop

Cuando Claude termina de responder. **Puede impedir que pare**, que es como se monta un bucle que no se rinde hasta que algo esté verde.

La estructura es siempre la misma: el evento, un **matcher** que dice a qué se aplica, y la lista de cosas que hacer. Este ejemplo pasa el formateador después de cada edición de archivo.

```
{"hooks": {
  "PostToolUse": [{
    "matcher": "Edit|Write",
    "hooks": [{
      "type": "command",
      "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/formatear.sh",
      "timeout": 60
    }]
  }]
}}
```

## Cómo se lee el matcher

Si son solo letras, dígitos, guiones, espacios, comas o barras verticales, se trata como **texto exacto o lista**: «Edit|Write» casa con esas dos. Si lleva cualquier otro carácter, pasa a tratarse como **expresión regular sin anclar**, y así una sola regla cubre todas las herramientas de un servidor MCP:

```
Edit|Write          esas dos, exactas
^Notebook           regex: las que empiecen así
mcp__memory__.*     todas las de ese servidor
*  o  vacío  o  sin matcher      todas
```

## La ruta del script

Usa siempre **${CLAUDE_PROJECT_DIR}**, que apunta a la raíz del proyecto, o rutas absolutas. Es el fallo más repetido: una ruta relativa que funciona en tu carpeta y en la de al lado no.

---

Los tiempos máximos por defecto: 600 segundos para command, http y mcp_tool; 30 para prompt; 60 para agent.

Lo que devuelve tu script decide qué pasa después, y las reglas son tres.

> Aquí está el error que comete todo el mundo la primera vez: **el código de salida 1 NO bloquea nada**. Para bloquear hay que salir con **2**. Está documentado y aun así es la pregunta más repetida.

## Salida 0 · todo bien

Completado con normalidad. Si lo que imprime empieza por llave, se interpreta como JSON; si no, como texto. En **UserPromptSubmit**, **UserPromptExpansion** y **SessionStart**, esa salida **se le enseña a Claude**, que es la vía para inyectarle contexto.

## Salida 2 · bloqueo

Impide la acción, y **lo que escribas en el error estándar es el motivo**. En PreToolUse bloquea la llamada; en UserPromptSubmit bloquea el mensaje; en Stop y SubagentStop impide que pare.

Con PostToolUse la herramienta ya se ejecutó, así que el 2 no la deshace: lo que hace es **enseñarle el error a Claude** para que reaccione.

## Cualquier otro código

Error que **no bloquea**. La acción sigue adelante. Aquí es donde cae el 1, y por eso los hooks de medio internet no hacen lo que creen que hacen.

## Lo que no se puede deshacer

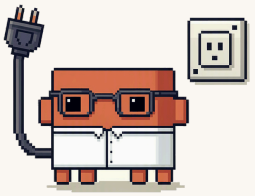
Una salida 2 **no se puede sobrescribir**: aunque el JSON diga que se permita, el bloqueo manda.

---

Y si un hook agota su tiempo, se cancela sin decidir nada: en PreToolUse eso significa que **no bloquea** y la llamada sigue el flujo normal de permisos.

El **Protocolo de Contexto de Modelo** es un estándar abierto para conectar Claude con herramientas de fuera: tu gestor de incidencias, tu base de datos, tu Figma. Conectas un servidor y Claude puede leer y actuar en ese sistema.

## Cuándo tiene sentido



El criterio oficial es bueno: **cuando te descubres copiando datos de otra herramienta al chat**. Si lo estás pegando a mano, hay un servidor que se lo daría solo.

## Los cuatro transportes

**HTTP** es el recomendado para servidores remotos y el más soportado. **SSE** está **obsoleto**: úsalo solo si un servicio no ofrece otra cosa. **stdio** ejecuta el servidor como proceso local en tu máquina, para cosas que necesitan acceso al sistema. **WebSocket** mantiene conexión abierta en las dos direcciones, para servidores que empujan eventos sin que se los pidan.

## El error de configuración más común

Una entrada de JSON que tiene **url** pero no tiene **type** es un error, porque sin type se interpreta como servidor stdio. El mensaje lo dice claro: añade type con http, sse o ws. En el JSON, **streamable-http** vale como alias de http, para que se puedan copiar configuraciones tal cual de la documentación de cada servidor.

---

Nombres reservados que no puedes usar: **workspace**, **claude-in-chrome**, **computer-use**, **Claude Preview** y **Claude Browser**.

Todo pasa por el comando **claude mcp**. Estas son las formas que vas a usar, con la sintaxis exacta.

## Remoto por HTTP

```
claude mcp add --transport http notion https://mcp.notion.com/mcp

claude mcp add --transport http api https://api.ejemplo.com/mcp \
  --header "Authorization: Bearer tu-token"
```

## Local por stdio

```
claude mcp add --env AIRTABLE_API_KEY=TU_CLAVE \
  --transport stdio airtable -- npx -y airtable-mcp-server
```

## Los dos guiones no son decorativos

El **--** separa las opciones de Claude de **el comando del servidor y sus argumentos**. Todo lo que va detrás se pasa tal cual. Sin él, Claude Code intentaría interpretar los flags del servidor como suyos y fallaría.

Segundo detalle de la misma familia: **--env** acepta varios pares, así que si el nombre del servidor va justo detrás, lo lee como otro par y lo rechaza. Pon otra opción en medio.

## Autenticación y mantenimiento

```
claude mcp login <nombre>     lanzar el OAuth
claude mcp logout <nombre>    borrar credenciales
claude mcp list               ver estado de todos
```

---

Si un valor de configuración lleva un espacio o un salto de línea pegado —de copiar un token—, avisa nombrando el campo pero **no lo recorta**: lo usa tal cual. Hay que arreglarlo a mano.

Al añadir un servidor eliges alcance, y eso decide en qué proyectos se carga y si tus compañeros lo ven. Es la decisión que más se equivoca.

---

## local El de por defecto

Solo en el proyecto donde lo añadiste y solo para ti. Se guarda en **~/.claude.json** bajo la ruta de ese proyecto. Para servidores de desarrollo, pruebas y **cualquier cosa con credenciales que no quieras en el control de versiones**.

---

## project Para el equipo

Se guarda en un **.mcp.json** en la raíz del proyecto, con formato estándar. Se hace commit y así todo el mundo tiene las mismas herramientas. Claude **pide aprobación** antes de usar servidores que vengan de ahí.

---

## user En todos tus proyectos

También en **~/.claude.json**, pero disponible en todo lo que abras en esta máquina, y privado. Para utilidades tuyas que usas en todas partes.

Un servidor MCP no es una extensión de navegador: es una tubería por la que entra texto que Claude va a leer como si se lo hubieras escrito tú.

> El aviso oficial es de una línea y conviene leerlo entero: **verifica que confías en cada servidor antes de conectarlo. Los servidores que traen contenido externo pueden exponerte a inyección de prompt.**

## El riesgo, concreto

Si un servidor te trae el contenido de una incidencia, de un correo o de una página, dentro de ese contenido puede haber instrucciones escritas por otra persona. Claude las lee. La regla de defensa es la misma que en el apartado 17: **lo que llega por una herramienta es dato, nunca orden**.

## La aprobación del proyecto

Por eso los servidores de **.mcp.json** piden aprobación en sesiones interactivas. Y desde la versión 2.1.196, las aprobaciones se leen **solo de ficheros que no estén en el repositorio** hasta que confíes en la carpeta: un repositorio clonado **no puede aprobarse a sí mismo** sus propios servidores.

Donde no se puede preguntar —ejecuciones con -p, el SDK, sesiones en la nube— **se cargan sin preguntar**. Para dejar uno fuera pase lo que pase, está la clave **disabledMcpjsonServers**.

## El límite de salida

Avisa cuando la salida de una herramienta MCP pasa de **10.000 tokens** y la corta por defecto en **25.000**. Se sube con la variable **MAX_MCP_OUTPUT_TOKENS**; el umbral del aviso es fijo.

---

Para reiniciar las aprobaciones de un proyecto: **claude mcp reset-project-choices**.

Un **subagente** es un Claude aparte, con su propia ventana de contexto, sus propias herramientas y su propio prompt. Trabaja aislado y devuelve solo un resumen. Es la herramienta para que lo verboso no te ensucie la conversación.

## Para qué sirven

Cuatro usos, y el primero es el que justifica todo lo demás. **Preservar el contexto principal** metiendo en otro sitio las operaciones que devuelven mucho ruido. **Imponer restricciones** de herramientas y permisos a una tarea concreta. **Reutilizar** configuraciones entre proyectos. Y **enrutar por modelo**: mandar lo barato a Haiku.

## Dónde viven

```
.claude/agents/        este proyecto
~/.claude/agents/      todos tus proyectos
<plugin>/agents/       donde esté el plugin
```

Por encima de los tres están los ajustes gestionados de la organización y el flag **--agents**, que los define al vuelo con JSON. Cuando dos sitios definen el mismo nombre, **gana el de más prioridad**.

## Cómo se invocan

Nombrándolo en tu petición, y Claude decide si delega. O **garantizándolo** con una arroba y su nombre. O poniendo la sesión entera a correr como ese subagente:

```
claude --agent revisor
```

---

Cuando uno termina, Claude puede **reanudarlo por su nombre** con todo su contexto, sin empezar de cero.

Es un Markdown con frontmatter, igual que una skill. Este es el ejemplo de la documentación, y debajo lo que se puede poner.

```
---
name: revisor
description: Revisa código buscando calidad y buenas
  prácticas. Úsalo después de escribir código.
tools: Read, Grep, Glob, Bash
model: sonnet
permissionMode: default
---

Eres un revisor de código veterano. Analiza en busca de:
- Legibilidad y mantenimiento
- Problemas de seguridad

Da feedback concreto y accionable, con ejemplos.
```

## Los campos que más rinden

**name** y **description** son obligatorios; la descripción es lo que Claude lee para decidir si delega. **tools** es la lista blanca; **disallowedTools** es la lista negra y **se aplica antes**. **model** acepta sonnet, opus, haiku, fable, un identificador completo o **inherit**, que es el valor por defecto.

**permissionMode** fija el modo de permiso solo para él. **maxTurns** le pone tope de vueltas. **memory** le da memoria propia. **isolation: worktree** lo hace trabajar en un worktree de git aislado, que es lo que evita que dos subagentes en paralelo se pisen los ficheros.

## Restringir a quién puede llamar

Un coordinador puede limitarse a lanzar solo ciertos subagentes:

```
tools: Agent(trabajador, investigador), Read, Bash
```

---

La buena práctica que más se salta: **dale solo las herramientas que necesita**. Un investigador con permiso de escritura es un accidente esperando.

Antes de escribir uno propio, conviene saber que hay tres de fábrica y que resuelven la mayoría de los casos.

## 1 Explore

Análisis de código **de solo lectura** y rápido. Corre en Haiku, con tope en Opus. Es el que se usa para «encuéntrame dónde está esto» en un proyecto grande: barre mucho y devuelve poco.

## 2 Plan

El agente de investigación del modo plan. Explora y propone sin tocar nada.

## 3 General-purpose

Tareas complejas de varios pasos que no encajan en los otros dos. Es el cajón de sastre.

Un **plugin** empaqueta de golpe varias cosas: skills, subagentes, hooks y servidores MCP. Es la forma de repartir una configuración entera en vez de ir pieza a pieza.

## Qué trae dentro

Las skills de un plugin viven en su carpeta **skills/**, los subagentes en **agents/** y los hooks en **hooks/hooks.json**. También puede traer configuración de servidores MCP. Todo se activa al habilitar el plugin y se apaga al deshabilitarlo.

## Los tres alcances al instalar

**Para ti**, disponible en todos tus proyectos. **Para este proyecto**, compartido con quien colabore. **Solo local**, tú y solo en este repositorio. Es la misma lógica de alcances que en el resto de la herramienta.

## Desde la terminal

```
claude plugin
claude --plugin-dir <carpeta o .zip>
claude --plugin-url <url de un .zip>
```

## Los espacios de nombres

Las skills de plugin se llaman **plugin:skill**, así que nunca chocan con las tuyas. Un plugin llamado mi-plugin con una skill deploy da **/mi-plugin:deploy**, y convive con tu propia /deploy.

## Las rutas dentro de un plugin

**${CLAUDE_PLUGIN_ROOT}** apunta a donde está instalado, y cambia al actualizarse. **${CLAUDE_PLUGIN_DATA}** es su carpeta de datos persistentes.

---

En VS Code hay interfaz gráfica: se escribe **/plugins** en la caja del prompt.

Claude Code no vive solo en la terminal. La extensión de VS Code es la forma recomendada de usarlo en ese editor, y cambia bastante la experiencia.

## Lo que hace falta

**VS Code 1.94.0 o superior** y una cuenta con suscripción de pago —Pro, Max, Team o Enterprise— o una cuenta de la Consola. **No hace falta clave de API**: se identifica al abrir la extensión por primera vez.

## Qué añade sobre la terminal

Revisar y **editar los planes antes de aceptarlos**, aceptar ediciones automáticamente según se hacen, mencionar archivos con arroba **indicando el rango de líneas de tu selección**, historial de conversaciones, y varias conversaciones abiertas en pestañas o ventanas distintas.

## Instalar

En VS Code, **Cmd+Shift+X** en Mac o **Ctrl+Shift+X** en Windows y Linux, buscar «Claude Code» e instalar. Funciona igual en Cursor y en el resto de derivados de VS Code, y está en el registro Open VSX. Si no aparece tras instalarla, reinicia el editor.

## El detalle que despista

La extensión **trae su propia copia del CLI** para el panel de chat. Para poder escribir **claude** en la terminal integrada del editor hace falta además la instalación independiente.

---

El atajo que se usa a diario: **Cmd+Esc** o **Ctrl+Esc** alterna el foco entre el editor y la caja del prompt. Hay extensión equivalente para JetBrains.

Una sesión no tiene por qué ocupar tu terminal mientras dura. Se puede mandar al fondo, dejarla corriendo y volver luego. Es la parte menos conocida de la herramienta.

## Al fondo

Con el flag **--bg** arranca en segundo plano y te devuelve el control al momento. Durante la sesión, el comando **/background**, con alias **/bg**, la manda al fondo a mitad.

## Volver a ellas

```
claude agents          panel de sesiones en paralelo
claude attach <id>     engancharla a esta terminal
claude logs <id>       ver su salida reciente
claude stop <id>       pararla
claude respawn <id>    reiniciarla con su conversación
```

## Reanudar y bifurcar

**claude -c** continúa la conversación más reciente de esta carpeta. **claude -r** reanuda una concreta por identificador o por nombre. Y **--fork-session** crea un identificador nuevo al reanudar, en vez de reutilizar el original: así pruebas otro camino sin perder el primero.

## Ponerles nombre

Con **--name** o **-n** se les pone nombre para encontrarlas después. Con veinte sesiones abiertas, los identificadores no valen para nada.

---

También hay sesiones en la nube con **--cloud**, y **--teleport** trae una sesión web de vuelta a tu terminal.

La API es Claude sin interfaz: tu programa manda texto y recibe texto. Se paga por token, no por cuota, y eso cambia por completo cuándo tiene sentido.

## Todo pasa por un sitio

Esto simplifica más de lo que parece: **todo va por un único punto de entrada**, el de mensajes. Las herramientas, los formatos de salida forzados, la caché y el razonamiento **no son APIs distintas**: son parámetros de esa misma llamada.

## Cuándo compensa frente a la suscripción

Compensa cuando el que va a hablar con Claude **no eres tú**: un formulario que clasifica lo que llega, un proceso nocturno, un producto con usuarios. Si eres tú escribiendo, la suscripción sale mucho más barata y te da además Claude Code.

## Los cuatro escalones

**Una llamada suelta** para clasificar, resumir, extraer o responder. **Un flujo de trabajo** cuando encadenas pasos y el orden lo decide tu código. **Un agente** cuando la tarea es abierta y el modelo tiene que decidir qué hacer. Y **agentes gestionados** cuando quieres que Anthropic te lleve el bucle y el entorno donde corren las herramientas.

El consejo oficial es empezar por el escalón más simple que resuelva el caso. Casi todo se resuelve en los dos primeros.

---

Antes de montar un agente, cuatro preguntas: ¿es complejo de verdad?, ¿lo vale por coste y latencia?, ¿Claude sabe hacerlo?, ¿y qué pasa si se equivoca? Si alguna falla, baja un escalón.

61 · LA PRIMERA LLAMADA

En Python son cuatro líneas. Lo importante no es el código, son las tres decisiones que lleva dentro.

```
import anthropic

client = anthropic.Anthropic()

response = client.messages.create(
    model="claude-opus-5",
    max_tokens=16000,
    system="Eres un revisor de textos en español.",
    messages=[{"role": "user", "content": "Revisa: ..."}],
)

for block in response.content:
    if block.type == "text":
        print(block.text)
```

## Uno · la clave no se escribe

El constructor sin argumentos **resuelve las credenciales solo**: mira la variable de entorno, y si no, el perfil que dejó una sesión de identificación. Poner la clave en el código es el error de principiante que acaba en un repositorio público.

## Dos · el contenido es una lista

**response.content no es texto**: es una lista de bloques, y cada uno tiene su tipo. Puede haber bloques de razonamiento, de uso de herramienta y de texto. Hay que comprobar el tipo antes de leer, como arriba.

## Tres · no escatimes en max_tokens

Si se agota, la respuesta se corta a media frase y hay que repetirla. La recomendación es **unos 16.000 sin streaming** y **unos 64.000 con streaming**. Solo se baja con motivo: clasificar algo puede ir con 256.

---

Para respuestas largas de verdad —hasta 128.000 tokens— **el streaming es obligatorio**, o la petición se cae por tiempo de espera.

Lo que había antes era decirle cuántos tokens podía gastar pensando. Lo que hay ahora son dos parámetros distintos.

> Si aprendiste esto hace un tiempo, tu memoria está caducada: **budget_tokens está eliminado** en los modelos actuales. No es que se ignore, es que **devuelve error 400**.

```
response = client.messages.create(
    model="claude-opus-5",
    max_tokens=16000,
    thinking={"type": "adaptive", "display": "summarized"},
    output_config={"effort": "high"},
    messages=[...],
)
```

## thinking

## display

Por defecto es **omitted**: los bloques de razonamiento llegan con el texto vacío. Si vas a enseñarle el razonamiento a un usuario, hay que pedir **summarized** a propósito, o parecerá que la respuesta tarda mucho sin motivo.

## effort

Va **dentro de output_config**, no arriba del todo: es el fallo de sintaxis más repetido. Cinco valores, de low a max, con **high** por defecto. Para subagentes y tareas mecánicas, low; cuando la corrección importa más que el coste, max.

## Y otra cosa que ya no va

**Las respuestas prerellenadas están eliminadas.** Poner un turno de asistente al final para forzar el formato devuelve 400. Para eso están ahora las salidas estructuradas.

---

El razonamiento en crudo no se devuelve nunca, en ningún modelo. Solo el resumen, y hay que pedirlo.

Es el ahorro más grande que hay y el que peor se usa, porque tiene una regla que lo rompe todo si no la sabes.

## La regla

**La caché casa por prefijo.** Cualquier byte que cambie en el prefijo **invalida todo lo que va detrás**. El orden de composición es: herramientas, sistema, mensajes. De ahí la única estrategia posible: **lo estable delante, lo que varía detrás**.

## Cómo se activa

```
response = client.messages.create(
    model="claude-opus-5",
    max_tokens=16000,
    cache_control={"type": "ephemeral"},
    system=texto_largo_y_estable,
    messages=[{"role": "user", "content": pregunta}],
)
```

Eso es la caché automática, que guarda el último bloque cacheable y es lo que quieres en casi todos los casos. Para control fino se pone **cache_control** en bloques concretos, con un máximo de **cuatro puntos de corte** por petición.

## El mínimo que nadie menciona

El prefijo mínimo cacheable son **unos 1.024 tokens**. Por debajo de eso **no cachea y no avisa**: simplemente no pasa nada.

## Comprobar que funciona

Mira **usage.cache_read_input_tokens**. Si sale cero una y otra vez con el mismo prefijo, hay un invalidador escondido: una fecha con la hora actual en el sistema, un identificador aleatorio, un JSON serializado sin ordenar las claves.

---

Los números: escribir cuesta 1,25 veces con caché de 5 minutos y 2 veces con la de 1 hora. Leer cuesta 0,1 veces.

Si el trabajo no tiene prisa, mandarlo por lotes cuesta **la mitad**, en entrada y en salida. Es el descuento más fácil de conseguir y el que menos gente aplica.

## El flujo

```
lote = client.messages.batches.create(requests=[
    {"custom_id": "ticket-1", "params": {...}},
    {"custom_id": "ticket-2", "params": {...}},
])

# se consulta hasta que processing_status == "ended"
estado = client.messages.batches.retrieve(lote.id)

for r in client.messages.batches.results(lote.id):
    if r.result.type == "succeeded":
        print(r.custom_id, r.result.message.content)
```

## La trampa

**Los resultados llegan en cualquier orden.** Hay que emparejarlos por **custom_id**, nunca por posición. Es el fallo que se cuela en producción y no aparece en las pruebas con tres elementos.

## Los cuatro finales

Cada resultado trae un tipo: **succeeded**, **errored**, **canceled** o **expired**. Solo del primero se puede leer el mensaje. Los otros tres hay que tratarlos, y expired existe: un lote puede caducar.

## Los descuentos se suman

Lote y caché se acumulan. Opus 5 pasa de 5 y 25 dólares por millón a **2,50 y 12,50** solo por el lote, y la parte cacheada baja además a una décima parte.

---

Lo que no entra: el modo rápido no funciona con lotes, y los agentes gestionados tampoco, porque sus sesiones son interactivas.

Una **herramienta** es una función tuya que Claude puede pedir que se ejecute. Tú la describes, él decide cuándo la necesita, tú la ejecutas y le devuelves el resultado. Ese bucle es la base de todo lo que se llama agente.

## El bucle, en una frase

Mandas la petición con la lista de herramientas. Si la respuesta viene con **stop_reason igual a tool_use**, ejecutas lo que pide, le devuelves el resultado y vuelves a llamar. Se repite hasta que termina normal.

## Las dos reglas de las llamadas en paralelo

Un mismo mensaje puede pedir **varias herramientas a la vez**, y viene activado por defecto. Ejecútalas en paralelo y devuelve **todos los resultados en un único mensaje de usuario**.

Si los repartes en varios mensajes, **le estás enseñando a dejar de pedir cosas en paralelo**. Y si una falla, devuelve su resultado con la marca de error, no la quites: un resultado que falta desconcierta más que uno que falló.

## El modo estricto

Con **strict: true** en la definición de la herramienta —arriba, junto al nombre, **no dentro de tool_choice**— se garantiza que los argumentos validan exactamente contra tu esquema. Requiere que el esquema tenga los campos obligatorios y prohíba propiedades extra.

## Y una advertencia de formato

Los modelos actuales pueden escapar el JSON de los argumentos de forma distinta. **Léelo siempre con un parseador de JSON**, nunca comparando cadenas.

---

Si no quieres escribir el bucle, los SDK traen un ejecutor que lo lleva por ti y te deja los ganchos por turno para aprobar, registrar o reintentar.

Hay herramientas que **no ejecutas tú**: corren en la infraestructura de Anthropic y el resultado llega en la misma respuesta. Se declaran y ya está.

## 1 Búsqueda web

Tipo **web_search_20260209**. Acepta límite de usos y listas de dominios permitidos o bloqueados. Cuesta **10 $ por cada 1.000 búsquedas**, más los tokens de lo que traiga. Cada búsqueda cuenta como un uso, den los resultados que den.

## 2 Descarga de páginas

Tipo **web_fetch_20260209**. **Sin coste adicional**: solo pagas los tokens del contenido. Solo descarga URL que ya estén en la conversación. Usa **max_content_tokens** o un PDF grande te come el presupuesto.

## 3 Ejecución de código

Tipo **code_execution_20260521**. **1.550 horas gratis al mes** por organización y luego 0,05 $ por hora y contenedor. Es **gratis del todo** si va acompañada de búsqueda o descarga web.

## 4 Búsqueda de herramientas

Cuando tienes muchas herramientas, se marcan con **defer_loading** y se cargan bajo demanda. Aviso: **el buscador no puede diferirse y al menos una herramienta tiene que quedar sin diferir**, o devuelve error.

Pedir «devuélvemelo en JSON» y confiar es cómo se rompen las cosas en producción a las tres semanas. Hay dos mecanismos que lo garantizan de verdad.

## Salidas estructuradas

Se restringe el formato de la respuesta con **output_config.format**, pasándole un esquema. El parámetro antiguo **output_format** está obsoleto: no lo uses en código nuevo.

En Python, lo recomendado es **client.messages.parse()**, que además valida la respuesta contra tu esquema y te devuelve el objeto ya comprobado.

## Los avisos que ahorran una tarde

No se pueden combinar con **citas**: activar las dos cosas devuelve 400. Y la llamada programática a herramientas tampoco es compatible con el modo estricto.

## Contar antes de gastar

Para saber lo que va a costar una petición antes de mandarla, está el contador de tokens. **Nunca uses un contador de otro proveedor**: los troceadores no coinciden y el número no vale.

```
cuenta = client.messages.count_tokens(
    model="claude-opus-5",
    system=sistema,
    messages=mensajes,
)
print(cuenta.input_tokens)
```

## Y mira siempre por qué paró

**stop_reason** puede ser end_turn, max_tokens, stop_sequence, tool_use, pause_turn o **refusal**. El último llega con código 200: si no lo compruebas antes de leer el contenido, te parece que la respuesta vino vacía.

---

Solo con refusal se rellena **stop_details**. En los demás casos es nulo, así que hay que comprobarlo antes de leerlo.

Se confunden constantemente porque suenan igual. Lo que las separa son dos preguntas: **quién pone el bucle** y **quién pone la infraestructura**.

## 1 Bucle a mano

Escribes tú el ciclo de comprobar el motivo de parada, ejecutar la herramienta y volver a llamar. Tú pones el bucle y tú alojas. Para cuando quieres controlar el flujo entero.

## 2 El ejecutor de herramientas del SDK

Escribes solo las funciones; el SDK lleva el bucle. **Tú sigues alojando.** Trae ganchos por turno para aprobar, interceptar errores, reintentar y compactar. Es lo que quieres en la mayoría de los casos.

## 3 Agentes gestionados

Anthropic pone **el bucle y el entorno**: cada sesión levanta un contenedor donde corren shell, ficheros y código. Tú defines el agente una vez y abres sesiones contra él. Para agentes largos, con estado y programados.

## 4 El Agent SDK

**Es otro producto**: es Claude Code empaquetado como biblioteca, con sus herramientas de leer, escribir, editar, shell y búsqueda ya dentro. Pone el bucle completo, pero **alojas tú**.

El último salto no es programar contra la API: es que Claude Code corra solo, sin nadie mirando. Y para eso ya está preparado.

## Modo sin interfaz

El flag **-p** manda una consulta y sale. Es la base de cualquier automatización:

```
claude -p "resume los cambios de hoy" --output-format json

cat informe.md | claude -p "saca las tres decisiones"
```

## Los flags que hacen falta en un script

**--output-format** con json o stream-json para poder procesar la salida. **--json-schema** para obligarla a validar contra un esquema. **--max-turns** para que no se enrede. **--max-budget-usd** para poner un tope de gasto en dólares antes de que pare. **--permission-mode** para fijar qué puede hacer sin preguntar.

## El aviso que cuesta caro aprender

Lo que tarda minutos **va en un paso del proceso, no en manos del agente**. Un agente informa y termina; si le dices que espere a que acabe algo largo, se va antes de tiempo y te quedas el trabajo a medias. Que espere el script, no él.

## Autenticar sin persona delante

En integración continua no hay nadie para hacer el login interactivo. Se genera un token de larga duración con **claude setup-token** y se guarda como secreto del repositorio.

---

Y para lo periódico, sesiones programadas: que el agente se dispare solo con una cadencia, sin planificador tuyo.

Lo que cambia al pasar de una cuenta individual a una de organización no es el uso: es **quién decide**. Y esa es la parte que hay que entender antes de contratarlo.

## Team

Para equipos **de 2 a 150 personas**. El puesto estándar cuesta **20 $ al mes con pago anual, o 25 $ mensual**; el premium, **100 $ anual o 125 $ mensual**, con **cinco veces más uso**. Trae Claude Code, Cowork, Design, Science, búsqueda de empresa, SSO y facturación centralizada.

## Enterprise

Arranca en **20 $ por puesto** con el consumo aparte, que escala según modelo y tarea. Lo que compras no es uso: son **permisos finos por rol**, aprovisionamiento automático de usuarios, **registros de auditoría**, controles de retención de datos a medida, opción preparada para HIPAA y Claude Security en beta.

## Los tres interruptores que cambian de dueño

Con cuenta de organización, tres cosas dejan de ser tuyas. **La memoria viene apagada** y tiene que encenderla el dueño. **La ejecución de código** viene encendida pero el dueño puede apagarla o limitar la red para todos. Y **la retención de datos** la fija la organización, no tú.

Todo eso vive en **Ajustes de la organización, Capacidades**.

---

Los chats incógnito en Team y Enterprise **entran igualmente** en las exportaciones de datos y en la API de cumplimiento. Incógnito es frente a la memoria, no frente a tu empresa.

71 · AJUSTES GESTIONADOS

Es el nivel que gana a todo lo demás. Si administras Claude Code para un equipo, esto es lo único que se aplica de verdad, porque nadie lo puede sobrescribir.

## Cómo llega a las máquinas

Tres vías: un fichero **managed-settings.json**, una política de gestión de dispositivos, o **ajustes gestionados desde la consola de claude.ai**. Solo los de la consola llegan además a las sesiones en la nube.

## La distinción que hay que tener clara

Los **ajustes** los impone el cliente pase lo que pase. El **CLAUDE.md gestionado** son instrucciones que Claude intenta seguir. No son intercambiables:

```
Bloquear comandos o rutas   ->  permissions.deny
Obligar al aislamiento      ->  sandbox.enabled
Variables de entorno        ->  env
Método de inicio de sesión  ->  forceLoginMethod
Estilo de código            ->  CLAUDE.md gestionado
Recordatorios de datos      ->  CLAUDE.md gestionado
```

## El truco de meter el CLAUDE.md dentro

En vez de desplegar un fichero aparte, se puede poner el contenido directamente con la clave **claudeMd** dentro del propio managed-settings.json. Solo se respeta en ajustes gestionados y de política: ponerla en los tuyos no hace nada.

## Lo que se puede cerrar del todo

**availableModels** limita qué modelos se pueden elegir. **allowManagedHooksOnly** desactiva los hooks de usuario y de proyecto. **disableBypassPermissionsMode** y **disableAutoMode** cierran los dos modos de permiso peligrosos. Y **el CLAUDE.md de política no se puede excluir** con claudeMdExcludes.

---

Para comprobar qué te impone tu organización sin preguntar a nadie: el comando **/config** y el diagnóstico de **/doctor**.

El orden importa. Estos cinco pasos, en este orden, resuelven la inmensa mayoría de los problemas sin buscar en ningún foro.

## 1 claude doctor

Diagnóstico de solo lectura de la instalación y de los ajustes. **No cambia nada.** Detecta el JSON roto, la versión vieja y la configuración incoherente. Es siempre el primer paso.

## 2 /context

Enseña qué hay cargado **de verdad** en esta sesión: qué ficheros de memoria, cuánto ocupa cada cosa. Si tu CLAUDE.md no aparece en la lista, no existe para Claude y todo lo demás que hagas es perder el tiempo.

## 3 ¿Has confiado en la carpeta?

Las reglas de permitir del fichero compartido y los hooks del proyecto **esperan al paso de confianza**. No dan error: simplemente no hacen nada. Es la causa más frecuente de «lo he configurado y no pasa nada».

## 4 ¿Lo pisa otro nivel?

El fichero del equipo gana al tuyo personal, y lo gestionado gana a todo. Y las variables de entorno **no están en esa escalera**: se deciden pareja por pareja, así que una variable exportada en tu shell puede estar mandando sin que aparezca en ningún fichero.

## 5 /doctor y --debug

El chequeo interactivo propone arreglos. Y **claude --debug**, con **--debug-file** para volcarlo a un fichero, enseña qué está pasando por dentro cuando ya no queda otra.

Todo lo que Claude lee a través de una herramienta —una página, un archivo, una incidencia, un correo— es texto que alguien pudo escribir. Y lo lee como si se lo hubieras escrito tú.

> Es el único riesgo de seguridad de esta guía que no se arregla con un ajuste. Se arregla entendiéndolo.

## El ataque, en una frase

Anthropic lo documenta así: es posible que alguien **añada instrucciones de forma poco visible en archivos externos o páginas web** para engañar a Claude y que ejecute código dañino o **saque datos de tu contexto hacia terceros**.

## Dónde aparece en esta guía

En tres sitios distintos, y no es casualidad. En la **ejecución de código** de la app. En los **servidores MCP**, donde el aviso oficial dice que verifiques que confías en cada uno antes de conectarlo. Y en cualquier agente que salga a la red.

## La regla que lo resuelve

## Lo que sí puedes hacer

Empezar **con el acceso a red desactivado** y abrirlo por dominios. Preferir servidores MCP de quien te fías. Poner un hook de **PreToolUse** que revise los comandos antes de que corran. Y no dejar corriendo en modo sin permisos nada que lea material de fuera.

---

El clasificador de inyección y la caja de arena están puestos y ayudan. Pero la última barrera es que tú sepas por dónde entra.

Todos. Yo los he cometido todos. Están puestos en el orden en que suelen aparecer.

## 1 Conversaciones eternas

Seguir un hilo de cuarenta turnos porque «ya tiene el contexto». Cada turno arrastra los anteriores: gastas mucho más y **encuentra peor lo que busca**. Contexto limpio gana a contexto grande. Abre una nueva.

## 2 Un CLAUDE.md de mil líneas

Cuanto más le metes, **menos caso hace a cada cosa**. Por debajo de 200 líneas. Lo que sea un procedimiento va a una skill; lo que solo importe en una carpeta, a una regla por ruta.

## 3 Reglas que se contradicen

Si dos instrucciones chocan, **elige una arbitrariamente**. Parece desobediencia y es un empate. Repasa el fichero cada cierto tiempo.

## 4 Salir con código 1 en un hook

No bloquea nada. **Para bloquear hay que salir con 2.** Es el error más repetido de todos los que hay en este manual.

## 5 Emparejar resultados de lote por posición

Llegan en cualquier orden. Se emparejan por **custom_id**. Con tres elementos de prueba nunca falla; con tres mil, sí.

## 6 Dejarlo en modo sin permisos «para ir rápido»

Ese modo escribe en **.git** y en **.claude** sin preguntar. Su sitio es un contenedor o una máquina virtual, no tu portátil con el trabajo dentro.

Un manual de noventa páginas no se aplica entero. Estos son los cinco movimientos que más cambian, en orden de rentabilidad, y ninguno lleva más de veinte minutos.

## 1 Abre los tres ajustes

**Capacidades**, **Memoria** y **Privacidad**. Enciende la ejecución de código si no la tienes, mira qué ha guardado sobre ti y decide de una vez qué haces con el interruptor de mejora del modelo. Diez minutos, una sola vez.

## 2 Monta un proyecto de verdad

Uno solo, del trabajo que más repites. Con **instrucciones escritas** y tres o cuatro ejemplos de tu propio trabajo bien hecho en la base de conocimiento. Es lo que más se nota de todo el manual.

## 3 Escribe un CLAUDE.md corto

Si usas Claude Code: **/init**, y luego recórtalo a lo que él no puede deducir solo. Menos de 200 líneas. Añádele los tres frenos del apartado 26.

## 4 Convierte tu primera skill

Coge lo que llevas pegando en el chat desde hace semanas y hazlo carpeta. Quince líneas bastan. La segunda ya la escribes en cinco minutos.

## 5 Baja el modelo cuando no haga falta

Si el resultado te vale con Sonnet, cambia. Es más de la mitad del gasto en la API, y en suscripción es la diferencia entre llegar al viernes y no llegar.

A1 · FLAGS DE SESIÓN Y MODELO

Todos los flags de esta referencia se pasan al arrancar: **claude --flag valor**. Están agrupados por para qué sirven, no por orden alfabético, porque así se encuentran.

## Retomar y bifurcar

```
-c, --continue      la conversación más reciente de esta carpeta
-r, --resume        una sesión concreta por id o por nombre
--fork-session      al reanudar, crear id nuevo en vez de reusar
-n, --name          ponerle nombre a la sesión
--session-id        usar un id concreto
```

## Modelo y esfuerzo

```
--model             modelo de esta sesión
--fallback-model    modelo de reserva si el primero falla
--effort            low | medium | high | xhigh | max
--advisor <modelo>  activar la herramienta de asesor
--betas             cabeceras beta para las peticiones
```

## Dónde trabaja

```
--add-dir           añadir directorios de trabajo
-w, --worktree      crear un worktree de git y su sesión
--ref <rama>        basar el worktree en una rama concreta
--tmux              crear una sesión de tmux para el worktree
--ide               conectarse al IDE si hay uno solo válido
```

## Segundo plano y remoto

```
--bg, --background  arrancar al fondo y devolver el control
--cloud             sesión nueva en claude.ai
--teleport          traer una sesión web a tu terminal
--rc, --remote-control   sesión con control remoto
```

A2 · FLAGS DE PERMISOS Y HERRAMIENTAS

El grupo que decide qué puede hacer Claude sin preguntarte. Los tres de arriba son los que se usan a diario; los de abajo, cuando montas algo desatendido.

## Permisos

```
--permission-mode   default | plan | acceptEdits | auto
                    | dontAsk | bypassPermissions
--allowedTools      herramientas que no piden permiso
--disallowedTools   reglas de denegación
--dangerously-skip-permissions        saltarse las preguntas
--permission-prompt-tool  herramienta MCP que resuelve los
                    permisos en vez del diálogo
```

## Qué herramientas existen

```
--tools             restringir las herramientas internas
--agent             correr la sesión como un subagente
--agents            definir subagentes al vuelo con JSON
--chrome            integración con Chrome
```

## MCP

```
--mcp-config        cargar servidores desde JSON o fichero
--strict-mcp-config usar solo los de --mcp-config
--channels          servidores cuyos avisos debe escuchar
```

## Contexto y prompt de sistema

```
--autocompact <auto|tokens>  ventana de autocompactado
--append-system-prompt       añadir texto al prompt de sistema
--system-prompt              reemplazarlo entero
--system-prompt-file         cargarlo de un fichero
--append-subagent-system-prompt  añadirlo a los subagentes
```

A3 · FLAGS DE AUTOMATIZACIÓN

Los que hacen falta cuando quien habla con Claude es un script y no una persona. La combinación mínima de un uso serio está en el primer bloque.

## Sin interfaz

```
-p, --print          responder y salir, sin modo interactivo
--output-format      text | json | stream-json
--input-format       text | stream-json
--json-schema        forzar salida válida contra un esquema
--include-partial-messages  eventos parciales del streaming
--replay-user-messages      reemitir los mensajes de entrada
```

## Topes

```
--max-turns          límite de vueltas del agente
--max-budget-usd     tope de gasto en dólares antes de parar
```

## Configuración de arranque

```
--settings           fichero JSON de ajustes, o JSON en línea
--setting-sources    qué fuentes de ajustes cargar
--safe-mode          arrancar sin ninguna personalización
--bare               saltarse hooks, skills y comandos propios
--init               ejecutar los hooks de Setup antes
--init-only          ejecutarlos y salir sin conversación
--maintenance        hooks de Setup con matcher de mantenimiento
```

## Plugins y depuración

```
--plugin-dir         cargar un plugin de una carpeta o .zip
--plugin-url         descargarlo de una URL
--debug              modo depuración, admite filtro por categoría
--debug-file         volcar los registros a un fichero
-v, --verbose        más detalle en la salida
--version            versión y salir
```

A4 · SUBCOMANDOS DE CLAUDE

No son flags: son comandos propios que se escriben detrás de **claude**.
Muchos resuelven en una línea algo que si no se busca por menús.

## Instalación y cuenta

```
claude update            actualizar a la última versión
claude install [ver]     instalar o reinstalar el binario
claude doctor            diagnóstico de solo lectura
claude auth login        identificarse
claude auth status       estado de la sesión, en JSON
claude auth logout       cerrar sesión
claude setup-token       token largo para CI y scripts
```

## Sesiones en paralelo

```
claude agents            panel de sesiones en segundo plano
claude attach <id>       engancharla a esta terminal
claude logs <id>         su salida reciente
claude stop <id>         pararla
claude respawn <id>      reiniciarla con su conversación
```

## Configuración

```
claude mcp               gestionar servidores MCP
claude mcp login <n>     lanzar su OAuth
claude mcp logout <n>    borrar sus credenciales
claude plugin            gestionar plugins
claude import [codex|gemini]   traer config de otro agente
claude project purge     borrar el estado local de un proyecto
```

## Otros

```
claude gateway           servidor de pasarela autoalojado
claude remote-control    servidor de control remoto
claude auto-mode defaults   reglas del clasificador, en JSON
```

Los que se escriben dentro de una sesión. Aquí van los de trabajo diario; en la página siguiente, los de configuración y los raros.

## Contexto

```
/context      dibujar lo que ocupa la ventana ahora mismo
/compact      resumir lo viejo para hacer sitio
/clear        empezar de cero  (alias /reset, /new)
/autocompact  fijar el umbral de autocompactado
/btw          preguntar al margen, sin entrar en el historial
/branch       ramificar la conversación
/fork         copiarla a una sesión nueva en segundo plano
```

## Código

```
/diff          visor de cambios sin confirmar
/code-review   revisar el diff o un PR  (alias /review)
/plan          entrar en modo plan
/goal          fijar una meta y trabajar hasta cumplirla
/batch         cambios a gran escala en paralelo
/autofix-pr    vigilar un PR y arreglar lo que falle
/security-review   revisión de seguridad
```

## Salida

```
/export     exportar la conversación como texto plano
/copy       copiar la última respuesta al portapapeles
/focus      ver solo el último prompt y su respuesta
/insights   informe HTML de tus sesiones recientes
/usage      lo que llevas gastado  (alias /cost)
/rate-limit-options   salidas cuando te quedas sin límite
```

Los de configuración, los de instalación y los que existen aunque casi nadie los use.

## Configuración

```
/config        ajustes  (alias /settings)
/model         cambiar de modelo y guardarlo por defecto
/effort        low | medium | high | xhigh | max | auto
/fast          activar o desactivar el modo rápido
/permissions   reglas de permitir, preguntar y denegar
                (alias /allowed-tools)
/memory        CLAUDE.md y memoria automática
/init          generar un CLAUDE.md leyendo el proyecto
/keybindings   abrir el fichero de atajos de teclado
```

## Extensiones

```
/agents      gestionar subagentes
/hooks       ver los hooks configurados
/mcp         servidores MCP y su autenticación
/plugin      gestionar plugins
/add-dir     añadir un directorio de trabajo
/cd          mover la sesión a otra carpeta
/ide         integraciones con el editor
```

## Mantenimiento y ayuda

```
/doctor      chequeo que diagnostica y arregla (alias /checkup)
/debug       registros de depuración
/help        ayuda y comandos disponibles
/powerup     lecciones interactivas de la herramienta
/bug         reportar un fallo o compartir la conversación
/privacy-settings   ver y cambiar la privacidad
/fewer-permission-prompts   proponer lista de permitidos
```

C · CLAVES DE SETTINGS.JSON

Las que más se usan, agrupadas. Van en cualquiera de los cuatro ficheros de ajustes; el nivel decide quién gana.

## Permisos

```
permissions.allow           reglas que no preguntan
permissions.deny            reglas prohibidas
permissions.disableBypassPermissionsMode
permissions.disableAutoMode
defaultMode                 modo de permiso de arranque
additionalDirectories       directorios de trabajo extra
sandbox.enabled             forzar el aislamiento
```

## Modelo

```
model             modelo por defecto
availableModels   lista blanca de modelos elegibles
agent             correr la sesión como este subagente
```

## Memoria e instrucciones

```
autoMemoryEnabled    encender o apagar la memoria automática
autoMemoryDirectory  moverla de sitio
claudeMd             CLAUDE.md embebido (solo gestionado)
claudeMdExcludes     ficheros CLAUDE.md que hay que saltarse
```

## Hooks y MCP

```
hooks                        los hooks de este nivel
allowManagedHooksOnly        solo los gestionados
allowedHttpHookUrls          URLs permitidas en hooks HTTP
enableAllProjectMcpServers   aprobar todos los de .mcp.json
disabledMcpjsonServers       bloquear estos en todos los modos
env                          variables de entorno del proyecto
```

D · VARIABLES DE ENTORNO

Las variables **no son un nivel** en la escalera de precedencia: cuando una clave tiene su pareja de variable, se decide caso por caso cuál manda. Aquí están las que hay que conocer.

## Credenciales y destino

```
ANTHROPIC_API_KEY      clave de la API
ANTHROPIC_AUTH_TOKEN   token alternativo
ANTHROPIC_BASE_URL     apuntar a otro servidor
```

## Modelo · ojo con la diferencia

```
ANTHROPIC_MODEL           MANDA sobre la clave model de
                          cualquier fichero de ajustes
ANTHROPIC_DEFAULT_MODEL   solo se aplica si NINGÚN fichero
                          define model
```

## Rutas y configuración

```
CLAUDE_CONFIG_DIR        mover ~/.claude a otro sitio
CLAUDE_PROJECT_DIR       raíz del proyecto; disponible en
                         hooks y en servidores MCP stdio
CLAUDE_PLUGIN_ROOT       dónde está instalado un plugin
```

## Comportamiento

```
CLAUDE_CODE_DISABLE_AUTO_MEMORY   a 1, apaga la memoria
                                  automática
CLAUDE_CODE_NEW_INIT              a 1, /init interactivo
                                  por fases
MAX_MCP_OUTPUT_TOKENS             tope de salida de MCP;
                                  por defecto 25.000
```

## Agente

Claude decidiendo solo qué pasos dar para cumplir un encargo abierto, usando herramientas entre medias. Se diferencia de un flujo de trabajo en que ahí **el orden lo decide tu código**.

## Caché de prompts

Guardar un trozo estable de la petición para no volver a procesarlo. Casa **por prefijo**: cualquier cambio invalida todo lo que va detrás.

## CLAUDE.md

Fichero de instrucciones que Claude Code lee al empezar cada sesión. **Se entrega como mensaje de usuario**, no como prompt de sistema, así que no hay cumplimiento garantizado.

## Contexto

Todo lo que Claude tiene delante a la vez. No es memoria: es campo de visión, y se vacía al empezar de nuevo.

## Hook

Comando tuyo que se ejecuta en un momento fijo del ciclo de vida. A diferencia de una instrucción, **se ejecuta pase lo que pase**.

## Inyección de prompt

Meter instrucciones escondidas en material que Claude va a leer, para que las obedezca como si fueran tuyas.

## MCP

Protocolo de Contexto de Modelo. Estándar abierto para conectar Claude con herramientas externas.

### Proyecto

En la app, un espacio con su propio historial, su base de conocimiento, sus instrucciones y **su propia memoria**, separada de la de fuera.

### Razonamiento adaptativo

El modelo decide solo cuánto piensa antes de responder. Sustituye al presupuesto fijo de tokens, que en los modelos actuales **devuelve error**.

### Regla por ruta

Instrucciones en **.claude/rules/** con un campo de rutas, que solo entran en contexto cuando Claude toca archivos que casan con el patrón.

### Skill

Carpeta con un SKILL.md dentro. Su cuerpo **solo se carga cuando se usa**, al revés que el CLAUDE.md.

### Subagente

Un Claude aparte, con su propio contexto y sus propias herramientas, que trabaja aislado y devuelve solo un resumen.

### Token

La unidad en la que se mide todo: precio, contexto y límites. Alrededor de cuatro caracteres. Los modelos desde la generación 4.7 usan un troceador que **produce un 30% más de tokens** para el mismo texto.

### Transporte

Cómo se conecta un servidor MCP: **http** (el recomendado), **sse** (obsoleto), **stdio** (proceso local) o **ws**.

### Ventana de contexto

Cuánto cabe delante a la vez. Un millón de tokens en los modelos grandes, unas 555.000 palabras.

## Esto *no se lee*: se consulta

Noventa páginas no se aplican de una sentada, y no hace falta. Si solo te llevas cinco cosas, que sean las del apartado 75: abrir los tres ajustes, montar un proyecto de verdad, escribir un CLAUDE.md corto, convertir tu primera skill y bajar de modelo cuando no haga falta. El resto está aquí para el día que lo necesites. Guarda el PDF y vuelve por el índice.

**@claudeprofessor**