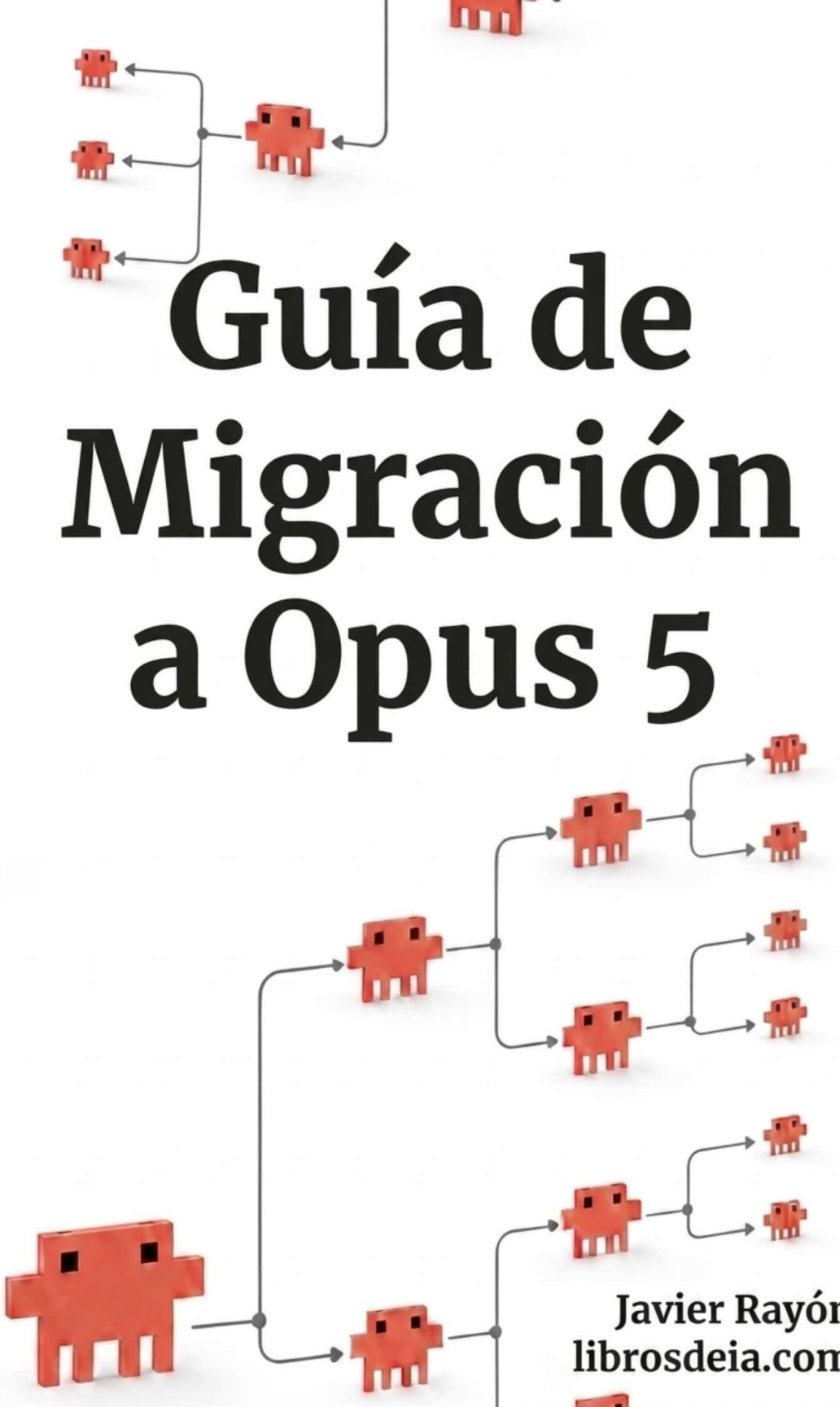


Guía de Migración a Opus 5

Javier Rayón
librosdeia.com

Contenido

1	Los cuatro modelos de la familia Claude 5	2
2	Qué modelo elegir para cada tarea	4
3	Seis patrones de prompt que ya no funcionan	6
4	Esfuerzo y longitud son ajustes distintos	10
5	El razonamiento está activo por defecto	12
6	Comandos nuevos y subagentes en segundo plano	14
7	Migración en 20 minutos	17
8	Qué cubre cada libro de la serie	20

Migrar a Opus 5 Qué cambia, qué borrar de tus prompts y cómo actualizarte en 20 minutos

© 2026 Javier Rayón. Todos los derechos reservados.

Publicado por librosdeia.com

Guía de distribución gratuita. Se permite compartir el fichero completo y sin modificar. Ninguna parte de esta publicación puede reproducirse de forma parcial, alterada o dentro de otra obra sin la autorización previa por escrito del autor.

Los nombres de productos y servicios mencionados son marcas registradas de sus respectivos propietarios. Esta guía no está afiliada a Anthropic ni patrocinada por Anthropic.

Los datos de modelos, precios y comportamientos recogidos aquí corresponden a la documentación de la plataforma consultada el 1 de agosto de 2026. Los precios se expresan en dólares por millón de tokens de entrada y de salida. La información se ofrece sin garantía de ningún tipo y el autor no será responsable de los daños derivados de su uso.

Primera edición: septiembre de 2026.

SECCIÓN 1

Los cuatro modelos de la familia Claude 5

Claude 5 no llegó en un único lanzamiento. Anthropic presentó Fable 5 el 9 de junio de 2026, Sonnet 5 el 30 de junio y Opus 5 el 24 de julio. Por eso encontrarás tres anuncios distintos, publicados con varias semanas de diferencia.

Opus 5 fue el último en llegar y es el modelo que cambia las decisiones de migración. Ofrece inteligencia de frontera por $5 / $25, la mitad que Fable 5 y lo mismo que Opus 4.8. También ha sustituido a Opus 4.8 como modelo Opus por defecto en Claude Code. Es posible que ya lo estés usando sin haber cambiado la configuración.

1.1 La familia, de un vistazo

Modelo	Disponible desde	Precio $/Mtok	Contexto	Uso recomendado
Fable 5	9 jun 2026	$10 / $50	1M	Problemas abiertos que requieren explorar antes de definir la tarea
Opus 5	**24 jul 2026**	**$5 / $25**	**1M**	Arquitectura, refactors amplios y depuración difícil
Sonnet 5	30 jun 2026	$3 / $15	1M	Desarrollo diario y tareas bien acotadas

Modelo	Disponible desde	Precio $/Mtok	Contexto	Uso recomendado
Haiku 4.5	oct 2025	$1 / $5	200K	Búsquedas, resúmenes y tareas mecánicas de volumen

Fable 5, Opus 5 y Sonnet 5 admiten un millón de tokens de contexto y hasta 128.000 tokens de salida. Haiku 4.5 admite 200.000 tokens de contexto y 64.000 de salida, suficiente para las tareas mecánicas a las que está orientado.

Opus 4.8 continúa disponible al mismo precio que Opus 5. La diferencia está en su comportamiento: Opus 5 razona por defecto, se revisa por su cuenta y delega con más facilidad. Las secciones siguientes explican cómo adaptar los prompts a esos cambios.

1.2 Dos precios que se confunden con facilidad

Algunas fuentes sitúan el precio de Opus 5 en $10 / $50. Esa es la tarifa del **modo rápido**, que usa el mismo modelo con una latencia menor. La tarifa normal de Opus 5 es $5 / $25.

Sonnet 5 también puede provocar errores de cálculo. Su precio promocional de lanzamiento fue $2 / $10, pero esa oferta termina el 31 de agosto de 2026. Para presupuestar los meses siguientes, usa el precio estándar: $3 / $15.

1.3 La caché admite prompts más pequeños

Opus 5 reduce el mínimo de caché de prompt a 512 tokens. Opus 4.8 y Sonnet 5 exigen 1.024 tokens.

Esto afecta sobre todo a los agentes con un prompt de sistema corto. Si uno de esos prompts no llegaba antes al mínimo, se pagaba completo en cada llamada. Con Opus 5 puede entrar en caché sin cambiar el código. Revisa los agentes que se quedaban fuera por unos pocos cientos de tokens.

SECCIÓN 2

Qué modelo elegir para cada tarea

Elige el modelo según el tipo de trabajo. Después ajusta el esfuerzo. Son dos decisiones distintas.

Trabajo	Modelo	Esfuerzo
Refactor amplio, diseño de arquitectura o depuración difícil	Opus 5	`high` (valor por defecto)
Funcionalidad acotada, revisión de código, tests o migración repetitiva	Sonnet 5	`medium`
Búsqueda en el repositorio, resumen o renombrado masivo	Haiku 4.5	— (no admite esfuerzo)
Problema abierto que primero exige averiguar qué ocurre	Fable 5	razona siempre
Sesión interactiva en la que la latencia es prioritaria	Opus 5, modo rápido	`high`

Fable 5 es la opción más cara y también la más fácil de usar donde no hace falta. Está pensado para problemas abiertos, cuando todavía hay que investigar antes de convertir el encargo en una tarea concreta. Un refactor grande puede ser difícil, pero ya tiene un objetivo definido. En ese caso, Opus 5 suele ser la elección adecuada y cuesta la mitad.

El modo rápido de Opus 5 mantiene la misma calidad de razonamiento y reduce la latencia. Se factura a $10 / $50. Puede compensar en una sesión interactiva larga, donde cada espera interrumpe el trabajo. Para procesos en segundo plano, la tarifa normal suele tener más sentido.

2.1 Ajustar el esfuerzo

Opus 5 admite niveles de esfuerzo desde `low` hasta `max`, con `high` como valor por defecto. Muchas tareas bien definidas funcionan con `low` o `medium`, así que este ajuste es el primer control de coste que tienes a mano.

Baja el esfuerzo en estos casos:

- La tarea está definida por completo desde el primer turno.
- El mismo patrón ya se ha resuelto varias veces en el repositorio.
- Los tests comprueban el resultado y devuelven una señal rápida.

En esas situaciones, el razonamiento adicional suele limitarse a confirmar decisiones que ya estaban tomadas.

Sube el esfuerzo cuando el problema es nuevo para el proyecto o cuando varias restricciones compiten entre sí. Por ejemplo, un cambio que debe mejorar el rendimiento sin romper la compatibilidad ni rebajar la seguridad merece más tiempo de razonamiento. Súbelo también cuando un error puede salir caro.

Cambiar a un modelo más barato sirve para otra clase de trabajo. Haiku 4.5 cuesta una quinta parte que Opus 5 y resuelve con holgura tareas como localizar una función, extraer las firmas de un módulo o resumir un log. Los subagentes pueden usar un modelo distinto al de la sesión principal. Así, Opus 5 toma las decisiones y Haiku se ocupa de las búsquedas.

2.2 Una referencia para vigilar el coste

La documentación de Claude Code publica estas cifras orientativas por desarrollador: unos 13 dólares por día activo, entre 150 y 250 dólares al mes, y menos de 30 dólares diarios para el 90 % de los usuarios. Tómalas como referencia, no como presupuesto obligatorio.

Si el gasto está muy por encima, revisa primero la delegación sin límite y el contexto que se carga sin aportar nada. Las secciones 3 y 6 explican cómo corregir ambos problemas.

SECCIÓN 3

Seis patrones de prompt que ya no funcionan

Opus 5 revisa su trabajo y corrige errores antes de responder. Si el prompt también le ordena verificar el resultado, el modelo repite el proceso. Gasta más tokens, pero la respuesta no mejora.

La documentación oficial de prompting para Opus 5 es directa: estas instrucciones deben eliminarse. La tabla resume los seis patrones que hay que buscar.

Lo que tenías escrito	Qué ocurre con Opus 5	Qué hacer
«Incluye un paso final de verificación»	Repite la revisión que el modelo ya hace	Borrarlo
«Usa un subagente para verificar el trabajo»	Abre otra sesión para repetir la revisión	Borrarlo
«Revísalo dos veces antes de responder»	Duplica la verificación nativa	Borrarlo
«Resume el progreso cada tres llamadas»	Añade más narración a un modelo que ya informa de su progreso	Borrarlo
«Usa subagentes siempre que puedas»	Delega tareas que cabrían en el hilo principal	Sustituirlo por un límite explícito
«No pienses, responde directamente»	Puede hacer visibles etiquetas internas del razonamiento	Borrarlo

3.1 Quitar la verificación duplicada

Las tres primeras instrucciones producen el mismo problema. Pueden aparecer como un paso del plan, un criterio de aceptación o una frase al final del prompt. En los tres casos, se eliminan.

```
# Antes
Implementa el endpoint según la especificación. Cuando termines,
incluye un paso final de verificación en el que revises tu propio
trabajo dos veces y compruebes que cumple todos los requisitos.

# Ahora
Implementa el endpoint según la especificación.
```

El ahorro principal está en la salida. Cada orden de revisión puede abrir otra ronda de razonamiento y reescritura sobre un trabajo que el modelo ya había comprobado. Reducir unas palabras del prompt ahorra poco; evitar una revisión completa ahorra bastante más.

La variante con subagente es la más cara. La nueva sesión hereda contexto, vuelve a leer los ficheros y redacta su propio informe. Además, el criterio de revisión queda repartido entre dos hilos que no comparten todo el estado.

3.2 Reducir la narración del progreso

Opus 5 explica con más frecuencia qué fichero abre, qué encuentra y qué hará después. Las órdenes que exigían un resumen cada cierto número de llamadas estaban pensadas para modelos que informaban poco. Ahora duplican la narración.

```
# Antes
Cada tres llamadas a herramienta, resume qué has hecho y qué queda.

# Ahora
(nada)

# Si la sesión sigue dando demasiados detalles
Informa solo al terminar un paso del plan o al quedarte bloqueado.
```

Primero borra la instrucción antigua. Añade la última frase solo si la sesión sigue interrumpiendo el trabajo con demasiadas actualizaciones.

3.3 Poner límites a los subagentes

Opus 4.8 necesitaba un empujón para delegar. De ahí salió el consejo de incluir «usa subagentes siempre que puedas» en tareas grandes. Ese consejo aparece en libros anteriores de esta misma serie, escritos para el modelo de entonces. Opus 5 delega con más facilidad, por lo que esa misma frase puede abrir sesiones innecesarias.

Cada subagente carga su propio contexto, lee ficheros y consume tokens. Sustituye la invitación abierta por una regla que indique cuántos puede lanzar y para qué.

```
# Antes
Usa subagentes siempre que puedas para paralelizar el trabajo.

# Ahora
Lanza como mucho dos subagentes, y solo para búsquedas en paralelo
sobre partes distintas del repositorio. El resto del trabajo va en
el hilo principal.
```

Dos subagentes son un punto de partida razonable para desarrollo habitual. Una auditoría grande puede necesitar más. Ajusta la cifra a tu trabajo; lo que cuenta es que exista un límite y un criterio de uso.

3.4 Pedir brevedad sin apagar el razonamiento

La orden «no pienses, responde directamente» se usaba para ahorrar tiempo en respuestas sencillas. En Opus 5, el razonamiento está activo por defecto. Forzar el comportamiento contrario puede hacer que aparezcan etiquetas internas en la respuesta visible.

Si solo necesitas una respuesta corta, pide una longitud concreta. Por ejemplo: «Responde en cinco frases como máximo». La sección 4 explica cómo separar la longitud del nivel de esfuerzo.

3.5 Conservar las comprobaciones externas

Los tests, los linters, la integración continua y la revisión humana siguen siendo necesarios. Producen señales externas al modelo y pueden detectar errores que su revisión interna no encuentra.

También se conserva el plan de verificación. Pedir qué tests hay que escribir, qué casos límite cubrir y qué revisar antes del despliegue crea un entregable que otra persona puede evaluar.

Usa este criterio: borra las instrucciones que piden al modelo repetir su propia revisión. Conserva las que generan una comprobación externa o un plan que utilizará otra persona.

3.6 Revisar todos los lugares donde guardas prompts

Estas instrucciones suelen estar repartidas. Búscalas en:

- El `CLAUDE.md` del proyecto y el global del usuario.
- Las skills propias, sobre todo en los criterios de aceptación.
- Los comandos personalizados de la carpeta de configuración.

- Las plantillas de prompt guardadas fuera del repositorio.
- Los system prompts de los agentes que llaman a la API.

Las plantillas externas y los agentes en producción se olvidan con facilidad porque el repositorio no permite encontrarlos con una única búsqueda.

3.7 Un system prompt más corto

Anthropic redujo el system prompt de Claude Code de unos 800 tokens a 164, un recorte del 80 %, sin perder rendimiento.

Muchas instrucciones antiguas compensaban limitaciones de modelos anteriores. Cuando el modelo deja de tener esa limitación, la instrucción permanece y sigue consumiendo tokens. Revisar y borrar esas capas reduce la entrada, evita rondas de razonamiento repetidas y deja menos órdenes compitiendo entre sí.

SECCIÓN 4

Esfuerzo y longitud son ajustes distintos

El esfuerzo controla cuánto razona el modelo antes de responder. La longitud se controla con una instrucción en el prompt. Bajar el esfuerzo para obtener una respuesta más corta suele producir el mismo volumen de texto, pero con menos razonamiento detrás.

	Esfuerzo	Longitud
Qué controla	Cuánto razona antes de responder	Cuánto texto entrega
Dónde se ajusta	En la configuración del modelo, de `low` a `max`	En el prompt, con una instrucción explícita
Qué consume	Tokens de razonamiento	Tokens de salida visible
Cómo reducirlo	Bajar el nivel de esfuerzo	Fijar frases, palabras, apartados o líneas

Opus 5 no tiene un parámetro de verbosidad. Para limitar la respuesta, hay que describir la longitud deseada igual que cualquier otro requisito.

4.1 Cuatro comportamientos que producen respuestas largas

Opus 5 tiende a extender cuatro partes del trabajo:

- Las respuestas que muestra al usuario.
- Los documentos que escribe en el repositorio.
- Las explicaciones sobre su propio progreso y sus correcciones.
- El alcance de la tarea, cuando detecta trabajo relacionado.

Cada comportamiento necesita una instrucción distinta. Por ejemplo, limitar la respuesta a cinco frases no evita que el modelo modifique un fichero adicional. Para eso hay que acotar también el alcance.

4.2 Cuatro instrucciones listas para copiar

Guárdalas en el `CLAUDE.md` si quieres aplicarlas siempre. Inclúyelas en el prompt cuando solo afecten a una tarea.

Para acortar la respuesta:

```
Responde en cinco frases como máximo. Sin preámbulo, sin recapitulación
final y sin repetir lo que ya está en el mensaje anterior.
```

Para limitar un documento:

```
El documento no debe pasar de 400 palabras. Un apartado por punto del
índice, sin introducción ni conclusión.
```

Para mantener el alcance:

```
Limítate a lo pedido. Si detectas trabajo adyacente pendiente,
enuméralo al final en tres líneas como mucho, sin ejecutarlo.
```

Para reducir la narración de correcciones:

```
Narra solo las correcciones que cambian una decisión anterior. Las
correcciones de detalle aplícalas sin comentarlas.
```

«Sé conciso» deja la longitud a criterio del modelo. Un límite medible funciona mejor: cinco frases, 400 palabras o tres líneas. Elige la unidad que encaje con el resultado que esperas.

4.3 Combinar esfuerzo y longitud

Para una tarea diaria bien especificada, prueba esfuerzo `medium` y fija una respuesta breve. Así reduces el razonamiento sin renunciar a controlar el texto visible.

Un problema difícil que debe quedar documentado puede necesitar esfuerzo `high` y una extensión mayor. Indica el índice y el límite de palabras desde el principio.

El caso que más contenido innecesario genera es usar esfuerzo alto sin indicar la longitud. El modelo dedica más razonamiento a la tarea y después redacta todo lo que considera útil, aunque nadie haya pedido un informe largo.

SECCIÓN 5

El razonamiento está activo por defecto

Opus 4.8 no razonaba si la petición omitía el bloque `thinking`. Opus 5 sí lo hace. Una petición idéntica puede consumir ahora parte de `max_tokens` en razonamiento antes de generar la respuesta.

Este cambio afecta sobre todo a las integraciones que llaman a la API. Claude Code ya gestiona la configuración, así que quienes solo usan esa herramienta pueden leer esta sección como referencia.

5.1 Por qué algunas respuestas quedan cortadas

`max_tokens` limita la suma del razonamiento y el texto visible. Ambos comparten el mismo presupuesto.

```
{
  "model": "claude-opus-5",
  "max_tokens": 4096,
  "messages": [ ... ]
}
```

Con Opus 4.8, una petición como esta reservaba 4.096 tokens para la respuesta porque no había razonamiento si se omitía `thinking`. Con Opus 5, esos 4.096 tokens deben cubrir primero el razonamiento y después la respuesta. Una carga que antes funcionaba cerca del límite puede empezar a devolver texto incompleto sin que el código haya cambiado.

Revisa en los logs el motivo de parada de las respuestas. Si empiezan a terminar por límite de tokens tras cambiar el modelo, aumenta `max_tokens`. Opus 5 admite hasta 128.000 tokens de salida.

5.2 Condiciones para desactivar el razonamiento

La API permite desactivarlo, pero no acepta esa opción con esfuerzo `xhigh` o `max`. Esa combinación devuelve un error 400.

```
{
  "model": "claude-opus-5",
  "thinking": { "type": "disabled" },
  "max_tokens": 4096,
  "messages": [ ... ]
}
```

La API comprueba la combinación en cada petición. Si un servicio asigna el esfuerzo según el tipo de tarea, puede funcionar con normalidad hasta que reciba una petición con un nivel incompatible.

5.3 Dos fallos difíciles de detectar

Desactivar el razonamiento puede provocar dos comportamientos sin lanzar un error.

El primero ocurre cuando el modelo escribe una llamada a herramienta como texto normal en vez de emitir un bloque estructurado `tool_use`. El turno termina correctamente, pero el cliente no ejecuta la herramienta. El resultado visible puede parecer válido aunque el fichero no se haya escrito, la consulta no se haya lanzado o el correo no se haya enviado.

El segundo comportamiento hace visibles etiquetas internas del razonamiento. Este fallo es más evidente porque aparece directamente en la respuesta que recibe el usuario.

5.4 Mantener el razonamiento y bajar el esfuerzo

La opción más segura es dejar el razonamiento activo y controlar el coste con `effort`. Opus 5 mantiene un buen rendimiento en tareas mecánicas con esfuerzo `low`, sin los dos problemas asociados al razonamiento desactivado.

Fable 5 requiere razonamiento en todas las peticiones. Si una integración comparte configuración entre modelos, necesita una rama que evite intentar desactivarlo cuando use Fable 5.

SECCIÓN 6

Comandos nuevos y subagentes en segundo plano

Las novedades de Claude Code cambian la forma de gestionar sesiones largas, subagentes y contexto. Estas son las que más afectan al trabajo diario con Opus 5.

6.1 Subagentes en segundo plano

Los subagentes se ejecutan **en segundo plano por defecto**. Puedes encargar una búsqueda larga y continuar trabajando en la sesión principal mientras llega el resultado.

También pueden anidarse hasta una profundidad de tres niveles. Un subagente puede abrir otro, y este puede abrir un tercero. Como Opus 5 delega con facilidad, una instrucción demasiado abierta puede crear varias sesiones y aumentar el coste con rapidez.

Claude Code aplica topes por sesión y por número de subagentes simultáneos. En el trabajo habitual es raro alcanzarlos. Si ocurre, suele indicar que la tarea se ha dividido más de lo necesario. El límite explícito de la sección 3 ayuda a evitarlo.

6.2 Gestionar varias sesiones

Tres comandos facilitan el trabajo con sesiones largas:

- `/fork` copia la conversación actual a una sesión nueva en segundo plano. Úsalo para probar otro diseño o una librería distinta sin llenar el contexto principal.
- `/subtask` lanza un subagente desde la sesión actual. Permite delegar una tarea concreta cuando tú lo decides.
- `/resume` muestra sesiones anteriores y permite reanudarlas, también en segundo plano.

Otros dos comandos completan el flujo. `/rewind` retrocede sobre el trabajo realizado y `/rename` asigna un nombre reconocible a la sesión.

Un flujo sencillo consiste en describir la tarea completa al inicio y lanzar con `/sub-task` una búsqueda que pueda ejecutarse de forma independiente. Mientras tanto, el trabajo continúa en el hilo principal. Si aparece una alternativa que merece una investigación propia, `/fork` la separa del contexto principal. Antes de terminar, `/rename` deja las sesiones preparadas para encontrarlas después con `/resume`.

6.3 Encontrar peso muerto en el contexto

`/doctor`, también disponible como `/checkup`, revisa la instalación y detecta contexto cargado que no está aportando nada.

Combínalo con estos datos y comandos:

- `/context` muestra qué ocupa la ventana de contexto. Las definiciones de servidores MCP se cargan cuando hacen falta, por lo que conectar un servidor ya no consume contexto desde el primer momento. Si dejaste de conectar servidores para no llenar la ventana, puedes revisar esa decisión.
- `/usage` atribuye el consumo a skills, subagentes, plugins y servidores MCP. También señala cualquier elemento que supere el 10 % del total.
- La caché de prompt dura una hora con suscripción. Con créditos de uso o con la API, dura cinco minutos.

La diferencia de duración de la caché corrige una recomendación antigua. La regla de reanudar antes de cinco minutos solo se aplica a créditos de uso y API. Con suscripción, el margen es de una hora.

6.4 Revisiones, equipos y tamaño de workflows

`/code-review ultra` inicia una revisión multiagente en la nube. La ejecución requiere una orden explícita del usuario, de modo que no se activa sola con cada commit. Encaja mejor en cambios grandes que necesitan una revisión adicional antes de pasar a una persona.

Los *agent teams* están desactivados por defecto. Consumen alrededor de siete veces más tokens que una sesión estándar cuando los miembros trabajan en modo plan. Resérvalos para problemas que se beneficien de varias perspectivas simultáneas y cuyo coste esté justificado.

El ajuste `workflowSizeGuideline`, disponible en `/config`, define la longitud de referencia de los workflows dinámicos. Modifícalo si esos workflows salen de forma sistemática más largos o más cortos de lo que necesita el equipo.

6.5 Cambios de herramientas y fallback en la API

La API permite añadir o quitar herramientas entre turnos sin perder la caché de prompt. Esto facilita los agentes que trabajan por fases. Pueden cargar primero herramientas de lectura y añadir después las de escritura sin volver a pagar todo el prompt.

El fallback en servidor permite enrutar un rechazo según su categoría sin mantener una lista de modelos en el código. Por ejemplo, un rechazo de categoría *cyber* puede enviarse a Opus 4.8. El crédito de fallback compensa el coste de caché del nuevo intento.

Un rechazo se recibe como una respuesta HTTP correcta, no como un error. El motivo de parada indica el rechazo e incluye su categoría. Por eso el cliente debe revisar ese campo antes de procesar el contenido de la respuesta.

Una petición rechazada antes de generar texto no se factura. Si el rechazo ocurre a mitad del stream, se factura lo emitido hasta ese punto.

SECCIÓN 7

Migración en 20 minutos

Este proceso tiene seis pasos. Los tiempos están calculados para un proyecto con un `CLAUDE.md`, una configuración global, varias skills y algún comando propio. Si mantienes muchas skills, reserva más tiempo para el segundo paso.

7.1 Confirmar el modelo activo (2 min)

Abre Claude Code en el proyecto que usas a diario y ejecuta `/model`. Si llamas a la API desde una aplicación propia, busca el identificador del modelo en la configuración del servicio.

Terminado cuando: sabes si estás usando `claude-opus-5`, `claude-sonnet-5` o un modelo anterior. Los pasos siguientes dependen de este dato.

7.2 Revisar `CLAUDE.md`, skills y comandos (7 min)

Ejecuta esta búsqueda desde la raíz del proyecto:

```
grep -rniE "verific|dos veces|doble ?check|resume .*progreso|usa subagentes|no pienses" \
  CLAUDE.md ~/.claude/CLAUDE.md .claude/ 2>/dev/null
```

Compara cada coincidencia con la tabla de la sección 3. Borra las instrucciones que piden al modelo revisar de nuevo su propio trabajo. Conserva las que exigen un artefacto para una persona, como un plan de pruebas.

Revisa también las plantillas guardadas fuera del repositorio y los system prompts de los agentes en producción. El comando anterior no puede encontrarlos.

Terminado cuando: al repetir la búsqueda solo aparecen líneas que has decidido conservar.

7.3 Limpiar el contexto (3 min)

Ejecuta `/doctor` y después `/context`. Busca skills que instalaste para una prueba, servidores MCP de proyectos anteriores o ficheros de contexto que ya no se usan.

Terminado cuando: `/usage` no muestra ningún elemento por encima del 10 % cuyo uso no puedas justificar en una frase.

7.4 Limitar la delegación (2 min)

Añade una regla como esta al `CLAUDE.md` del proyecto:

```
Lanza como mucho dos subagentes a la vez, y solo para búsquedas en
paralelo sobre partes distintas del repositorio. El resto del trabajo
va en el hilo principal.
```

Ajusta el número a tu tipo de trabajo. Lo necesario es establecer un límite explícito y describir cuándo se permite delegar.

Terminado cuando: la regla está en el fichero y no queda ninguna instrucción que anime a abrir subagentes sin límite.

7.5 Probar un nivel de esfuerzo menor (4 min)

Elige una tarea que repitas cada semana y reduce el esfuerzo un nivel. Si usabas `high`, prueba `medium`.

Compara el resultado con un criterio objetivo, como los tests o el diff esperado. Después revisa el consumo con `/usage`. Mantén el nivel inferior si el resultado conserva la calidad. Si empeora, vuelve al anterior para ese tipo de tarea.

Terminado cuando: has comparado calidad y consumo en una tarea real.

7.6 Revisar `max_tokens` (2 min)

Este paso solo afecta a las aplicaciones que llaman a la API y antes omitían el razonamiento. Busca en los logs recientes respuestas cuyo motivo de parada sea el límite de tokens.

Terminado cuando: no hay respuestas recientes cortadas por agotar el presupuesto. Si las encuentras, aumenta `max_tokens`; Opus 5 permite llegar hasta 128.000.

7.7 Hoja de comprobación

#	Paso	Tiempo	Queda hecho cuando
1	Confirmar el modelo	2 min	Conoces el identificador exacto
2	Revisar prompts y configuración	7 min	La búsqueda solo devuelve líneas conservadas a propósito
3	Limpiar el contexto	3 min	Nada supera el 10 % sin justificación
4	Limitar la delegación	2 min	La regla está en el `CLAUDE.md`
5	Bajar el esfuerzo	4 min	Has comparado resultado y coste
6	Revisar `max_tokens`	2 min	Los logs no muestran respuestas cortadas

Repite el paso 5 con varios tipos de tarea. Una sola prueba sirve para ajustar ese trabajo concreto, pero no para decidir el nivel de esfuerzo de todo un equipo.

SECCIÓN 8

Qué cubre cada libro de la serie

Esta guía se ocupa de una tarea concreta: adaptar la configuración existente a Opus 5. Explica qué ha cambiado, qué instrucciones borrar y cómo completar la migración. Si ese era tu objetivo, ya tienes todo lo necesario.

Los demás temas se desarrollan en tres libros de la misma serie:

Primeros Pasos con Claude Code está pensado para quien acaba de empezar. Explica la primera hora de trabajo, cómo formular una petición, cómo orientarse en un repositorio desconocido y cómo usar tests, `CLAUDE.md` y git. Es el punto de entrada adecuado si algunos términos de esta guía te han resultado nuevos.

Claude Code: Tips Avanzados se dirige a quienes ya usan la herramienta a diario. Trata la paralelización, la orquestación de subagentes, la automatización, el control de coste y la seguridad. También dedica un capítulo a Fable 5 y explica cuándo compensa usarlo. Allí encontrarás ejemplos más amplios sobre los límites de delegación que aquí se resumen en una regla.

Claude Code para CTOs aborda las decisiones de equipo. Cubre la retención de datos, los clasificadores de seguridad, los rechazos, el fallback, el coste total de propiedad y la preparación de un caso de negocio con datos del fabricante. Está dirigido a quien debe aprobar y justificar el presupuesto.

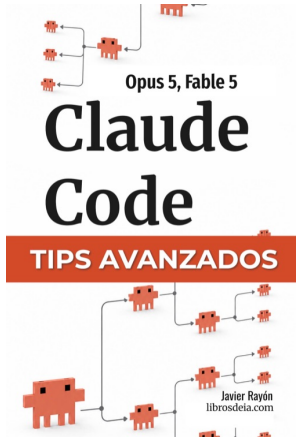
Los tres libros son de pago y están disponibles en librosdeia.com. Ninguno es necesario para aplicar la migración de esta guía.

La colección Claude Code

Los títulos de la colección, en librosdeia.com.



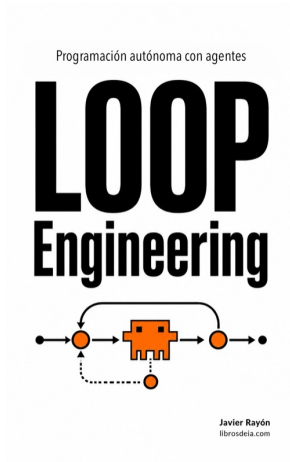
Primeros Pasos con Claude Code



Claude Code. Tips Avanzados para Power Users



Claude Code para CTOs



Loop Engineering