Cómo ahorrar tokens con un agente de IA: la traza real y 5 reglas

> Un agente de IA es el mismo modelo con herramientas y un bucle, y **en cada vuelta se le reenvía la conversación entera**: por eso casi todo lo que gasta es releer. En la traza real de abajo, el 99 % de los tokens fue de entrada. Para que gaste menos: **conecta solo las herramientas que vas a usar, dale un comando para comprobar su trabajo, permisos justos, tareas cortas en sesión nueva y revisa el diff, no el resumen.**

La traza completa, vuelta a vuelta

El encargo a Claude Code (`claude -p`), en un mini juego de rol en Python donde la tienda cobraba el oro dos veces: *«Encuentra el fallo y arréglalo sin tocar los tests. Para ejecutar los tests usa exactamente: uv run --with pytest pytest -q»*. Con 5 herramientas: Read, Edit, Bash, Glob y Grep.

Vuelta · tokens de entrada	Qué pidió el agente	Qué pasó
1 · 7.835	**Bash**: listar los archivos (*ls -R*) y pasar los tests.	2 tests fallan, 1 pasa. Con eso ya sabe dónde mirar: la tienda.
2 · 8.561	**Bash**: ver el código de los cuatro archivos con *cat -n*.	**Denegado.** Ese comando no estaba en su lista de permitidos y el modo *dontAsk* lo rechaza sin preguntar.
3 · 8.977	**Read ×4** a la vez: *tienda.py*, *test_tienda.py*, *personaje.py* e *inventario.py*.	Nadie le dice qué hacer: lee el rechazo y cambia a la herramienta de lectura, que sí tenía permitida.
4 · 10.617	**Edit** en *juego/tienda.py* y **Bash** con el comando exacto de los tests.	Ve el doble cobro (*pagar(coste)* y debajo *oro -= coste*), borra la segunda línea y pasan los 3 tests.
5 · 11.337	Ninguna herramienta.	Escribe «arreglado, pasan los 3 tests». Cuando el modelo contesta sin pedir nada, el bucle se para.

Dónde se van los tokens: la misma tarea con 5 y con 191 herramientas

Medida	Con 5 herramientas	Con las 191 que tenía conectadas
Tokens de entrada en la 1.ª llamada, antes de leer el código	7.835	**39.675** (5 veces más)
Tokens de entrada sumando todas las vueltas	47.327	**165.952**
Tokens de salida (lo que escribió)	1.748	1.343
De cada 100 tokens, cuántos fueron releer	96	**99**
Coste a precio de API (con caché de prompts)	0,095 $	0,19 $
Resultado	El mismo: una línea borrada. El código que leyó, 4 archivos con 2.044 caracteres.	

Las 5 reglas, con lo que tienes que hacer

Qué haces	Qué consigues	Cuándo usarlo
1. Conecta solo lo que vas a usar. En la sesión, desactiva servidores con `/mcp`. Con `claude -p`: `--tools` y `--strict-mcp-config`.	Cada herramienta conectada viaja descrita en cada vuelta. Menos herramientas, menos contexto antes de empezar.	Antes de una tarea larga o de un agente que vas a lanzar muchas veces, como un script o una tarea programada.
2. Dale una forma de comprobar su trabajo. Pon en el encargo el comando exacto: tests, linter o build.	Comprueba él mismo el resultado en vez de decirte «arreglado» sin saberlo.	Siempre que el proyecto tenga tests o algo que se pueda ejecutar para saber si funciona.
3. Dale permisos justos. `--allowedTools` con lo que necesita y `--permission-mode dontAsk`; en la sesión, `/permissions`.	Lo que no está permitido se rechaza sin preguntar, y el agente busca otro camino sin romper nada.	Cuando lo dejas trabajando solo y no vas a estar para aprobar cada comando.
4. Tareas cortas, sesión nueva. `/clear` para empezar limpio, `/compact` para resumir y `/context` para ver qué llena la ventana.	El contexto solo crece: de 7.835 a 11.337 tokens en cinco vueltas. Y lo que queda en medio se aprovecha peor.	Al cambiar de tarea, o cuando empieza a ignorar instrucciones que le diste al principio.

Qué haces	Qué consigues	Cuándo usarlo
5. Mira el cambio, no el resumen. `git diff` antes de aceptar.	El resumen cuenta lo que dice que hizo; el diff, lo que hizo de verdad. Aquí: una línea borrada.	Antes de hacer commit o de dar la tarea por cerrada.

Flags de Claude Code para controlar un agente

Qué escribes	Qué hace	Cuándo usarlo
`claude -p "encargo"`	Ejecuta el agente sin sesión interactiva y termina al acabar.	En scripts, tareas programadas o para medir una tarea igual varias veces.
`--output-format stream-json --verbose`	Saca cada mensaje como JSON, con la herramienta pedida y los tokens de entrada, de caché y de salida.	Para ver la traza vuelta a vuelta y saber en qué se te van los tokens.
`--tools "Read,Edit,Bash"`	Limita las herramientas propias que tiene disponibles.	Cuando la tarea solo necesita leer, editar y ejecutar: el resto sobra en el contexto.
`--strict-mcp-config`	Solo carga los servidores MCP que le pases con `--mcp-config`; ignora todos los demás.	Cuando tienes muchos conectores instalados y esta tarea no usa ninguno.
`--allowedTools "Bash(uv run --with pytest pytest:*)"`	Lista de lo que puede hacer sin pedir permiso, hasta el comando exacto.	Para dejarle pasar los tests y nada más con Bash.
`--max-budget-usd 1`	Pone un tope de gasto en dólares a la ejecución.	Cuando lo lanzas desatendido y no quieres sorpresas en la factura.

Errores típicos

- **Conectar todos los servidores MCP «por si acaso».** Cada uno mete la descripción de sus herramientas en cada llamada, los uses o no. En la prueba, eso multiplicó por cinco la primera llamada.
- **Creerte el «arreglado».** Sin un comando para comprobarlo, el agente te cuenta lo que cree que ha hecho. Dale los tests y mira que pasan.
- **Darle todos los permisos para que no pregunte.** Es más cómodo, pero lo que no debía hacer lo descubres después. Mejor una lista corta de permitidos y que lo demás se rechace.
- **Encadenar tareas en la misma sesión.** Cada tarea nueva paga el contexto de todas las anteriores en cada vuelta. Cierra con `/clear` y empieza limpio.
- **Mirar solo lo que escribe.** La salida fue menos del 4 % de los tokens. El gasto de verdad está en lo que relee en cada vuelta.

Preguntas frecuentes

¿Qué es un agente de IA?

El mismo modelo de lenguaje de siempre, con herramientas y un bucle. El modelo no ejecuta nada: escribe una petición («quiero usar esta herramienta con estos datos»), un programa alrededor, el arnés, la ejecuta y le pega el resultado en la conversación, y se le vuelve a llamar. El bucle se para cuando el modelo contesta sin pedir ninguna herramienta.

¿Por qué un agente de IA gasta tantos tokens?

Porque en cada vuelta se le reenvía la conversación entera: el encargo, las instrucciones, la descripción de cada herramienta y todos los resultados anteriores. En la traza real de esta chuleta, de cada 100 tokens, 96 (con 5 herramientas) y 99 (con 191) fueron de entrada: releer lo que ya había visto. Lo que escribe es muy poco.

¿Las herramientas conectadas gastan tokens aunque no se usen?

Sí, porque su descripción viaja en el contexto. En la prueba de esta chuleta, con 5 herramientas la primera llamada pesó 7.835 tokens; con las 191 que había conectadas, 39.675, para hacer exactamente el mismo arreglo. Conecta solo lo que la tarea necesita.

¿Por qué el coste no sube tanto como los tokens?

Por la caché de prompts: la parte de la conversación que ya se envió antes se cobra con descuento. En la prueba, 3,5 veces más tokens de entrada salieron solo al doble de precio: 0,19 $ frente a 0,095 $ a precio de API. Ayuda, pero cada vuelta sigue pagando el contexto entero.

¿Cómo veo cuántos tokens gasta mi agente en Claude Code?

En la sesión interactiva, /context enseña qué está llenando la ventana de contexto. En modo no interactivo, claude -p con --output-format stream-json y --verbose devuelve cada mensaje con sus tokens de entrada, de caché y de salida: así se sacó la traza de esta chuleta.