# Claude Code en modo headless: desarrollá tu agente conversacional sin cargar crédito al API

Gonzalo Gomez · ggomez.dev · @ggomezdev

Cada prueba que hacés contra el API de Anthropic mientras desarrollás un agente conversacional te cuesta plata. Ajustás el system prompt, mandás 40 mensajes de prueba, algo no te cierra, volvés a ajustar, otros 40 mensajes. Todo eso es facturación por tokens antes de tener un solo usuario real.

Si tenés suscripción de Claude (Pro o Max), hay una alternativa que casi nadie usa para esto: el modo headless de Claude Code. Le pasás el prompt y el mensaje del usuario, te devuelve la respuesta en texto plano, y el consumo sale de tu suscripción, no del API. Este documento es la implementación de principio a fin, con los casos de uso donde tiene sentido y el límite donde deja de tenerlo.

## La idea en una línea

Claude Code no es solo la terminal interactiva. Con el flag `-p` (o `--print`) corre sin interfaz: procesa un prompt, imprime el resultado y termina. Eso lo convierte en un binario que tu backend puede invocar como cualquier otro comando.

```
claude -p "Respondé este mensaje de un cliente: ¿tienen envíos a Córdoba?"
```

Eso es todo el concepto. Lo que sigue es hacerlo usable dentro de un agente real.

## El comando base para un agente

Para integrarlo a un sistema necesitás tres cosas: salida estructurada, tu system prompt, y control de herramientas.

```
claude -p "¿Tienen envíos a Córdoba?" \
  --append-system-prompt "$(cat system_prompt.txt)" \
  --output-format json \
  --allowedTools ""
```

- `--output-format json` te devuelve un objeto con `result` (el texto de la respuesta), `session_id` y metadata. Sin esto estás parseando texto plano y en algún momento se te rompe.
- `--append-system-prompt` suma tu prompt de agente al system prompt base de Claude Code.
- `--allowedTools ""` le saca las herramientas de código. Para un agente de FAQ no querés que se ponga a leer archivos del servidor. Si tu agente sí necesita herramientas, las habilitás explícitamente, y también podés conectarle servidores MCP con `--mcp-config`.

## Implementación de principio a fin

El patrón es un wrapper: tu canal de comunicación (WhatsApp, web chat, lo que sea) recibe el mensaje, tu backend invoca `claude -p`, y devolvés el texto por el mismo canal. Con Python:

```
import subprocess
import json

SYSTEM_PROMPT = open("system_prompt.txt").read()
```

```
def responder(mensaje: str, session_id: str | None = None) -> dict:
    cmd = [
        "claude", "-p", mensaje,
        "--append-system-prompt", SYSTEM_PROMPT,
        "--output-format", "json",
        "--allowedTools", "",
    ]
    if session_id:
        cmd += ["--resume", session_id]

    proc = subprocess.run(cmd, capture_output=True, text=True, timeout=120)
    data = json.loads(proc.stdout)
    return {
        "texto": data["result"],
        "session_id": data["session_id"],
    }
```

Y el endpoint que recibe el webhook del canal:

```
from fastapi import FastAPI, Request

app = FastAPI()
sesiones = {}  # en serio usá SQLite, esto es para el ejemplo

@app.post("/webhook")
async def webhook(req: Request):
    body = await req.json()
    usuario = body["from"]
    mensaje = body["text"]

    respuesta = responder(mensaje, sesiones.get(usuario))
    sesiones[usuario] = respuesta["session_id"]

    return {"reply": respuesta["texto"]}
```

Ese mapeo usuario a `session_id` es la parte que hace que esto funcione como agente conversacional y no como respuestas sueltas. Cada corrida de `-p` es stateless por defecto, pero `--resume` con el `session_id` de la corrida anterior te mantiene el hilo completo de la conversación sin que tengas que armar el historial a mano. Es el equivalente a mandar el array de `messages` completo en el API, con la diferencia de que Claude Code te lo guarda él.

## Streaming

Para no esperar la respuesta completa antes de empezar a mostrarla, existe `stream-json`:

```
claude -p "..." \
  --output-format stream-json \
  --verbose \
  --include-partial-messages
```

Te emite eventos JSON delimitados por línea, a medida que pasan. El `--verbose` no es opcional, `stream-json` lo requiere, y `--include-partial-messages` es lo que te da los deltas de texto a nivel token. En tu wrapper cambiás `subprocess.run` por `subprocess.Popen` y vas leyendo stdout línea por línea. Para un canal como WhatsApp el streaming no te sirve de mucho porque el mensaje se manda entero igual, pero para un chat web o una consola de pruebas cambia la experiencia por completo.

## Casos de uso donde esto tiene sentido

- **Iterar el system prompt de tu agente.** El caso principal. Corrés cientos de mensajes de prueba contra el prompt sin que el medidor del API se mueva. Es lo que uso para probar los prompts del boilerplate de agente de WhatsApp que tengo publicado (github.com/GonzaGomezDev/claude-whatsapp-chatbot-skills) antes de que nada toque el API.
- **Suite de regresión de prompts.** Un script que corre tus 30 mensajes de prueba después de cada cambio al prompt y compara contra las respuestas esperadas. Con `--output-format json` el parseo es trivial.
- **Demos a clientes.** Mostrás el agente funcionando end to end en una llamada sin haber configurado billing de nada.
- **Prototipos internos y scripts personales.** Un cron que resume tus logs, un clasificador de mails para vos. Uso personal real, que es exactamente para lo que está la suscripción.

## El límite, y es importante

Esto es para desarrollo y pruebas locales, no para producción. Por dos motivos.

El primero es de términos de uso: la suscripción de Claude es para tu uso individual. Servir respuestas a usuarios finales de tu producto por esta vía es revender la suscripción, y eso está fuera de lo permitido. El segundo es técnico y te lo dice la propia arquitectura: cada request levanta un proceso, compartís los rate limits de tu suscripción con tu propio uso de Claude Code, y no tenés los controles de un backend de producción. Cuando el agente pasa a atender clientes reales, migra al API con el Anthropic SDK, que además es un cambio chico si armaste el wrapper como función separada, como en el ejemplo de arriba.

Yo lo veo así: el modo headless te saca el costo de la etapa donde más requests quemás, que es cuando el agente todavía no funciona. La etapa donde el agente ya funciona es justamente la que tiene que pagar su propio API.

Los flags y formatos de salida están documentados en https://code.claude.com/docs/en/headless por si algo cambió desde que escribí esto.

**-Gonza**