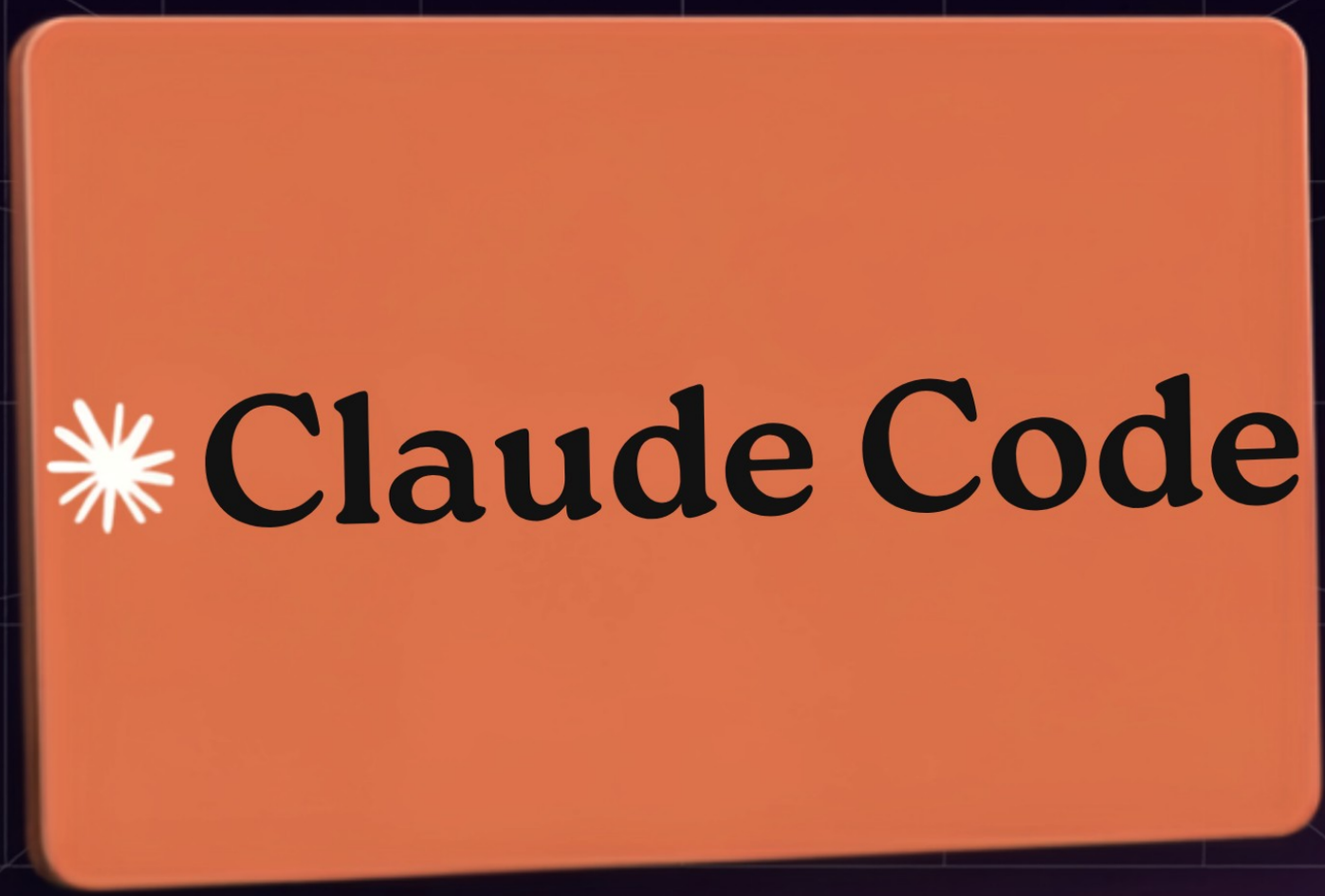
Manual de referencia

# Códigos de Claude

## Paso a paso




Instalar, usar y dominar Claude Code
con los comandos que usa la comunidad

por **Paco • Diario Vida IA**

LÉEME PRIMERO

# Una caja de herramientas, no un examen

Esto no es un listado para empollar. Cada comando de aquí existe por una razón muy concreta: **ahorrarte tiempo, tokens y disgustos** cuando trabajas con Claude Code. Si te llevas tres, ya has ganado el rato de leerlo.

## ¿Es para ti?

Si usas Claude Code —o piensas usarlo— para escribir, automatizar, investigar o programar, sí. **No hace falta ser programador.** La terminal asusta de lejos y se hace pequeña en cuanto le pierdes el respeto: es solo una ventana donde escribes órdenes en vez de hacer clic.

## La promesa

- **Menos horas** peleando con la herramienta.
- **Menos tokens** tirados a la basura — y menos factura a final de mes.
- **Menos sustos** — nada de borrar lo que no se debe tocar.

## Cómo usarlo

No lo leas de corrido. Mira el índice, ve a lo que necesites **hoy** y vuelve mañana a por lo siguiente. Es de consulta, no de lectura.

## Empieza por aquí — los tres imprescindibles

```
/clear                    # tarea nueva: empiezas con el contexto limpio
/compact                  # cada 20-30 min: libera memoria sin perder el hilo
Shift+Tab  ->  Auto Mode  # deja de pedirte permiso a cada paso
```

Con esos tres ya notas la diferencia el primer día. ¿Aún no lo tienes instalado? Empieza por el capítulo siguiente y en cinco minutos estás dentro.

# En este manual

> **RUTA RÁPIDA · 15 MINUTOS**
> ¿Con prisa? Lee solo esto: Instalación → Parte 2 (que deje de pedir permiso) → Parte 3 (tu día a día) → Parte 8 (ahorra tokens). Con eso ya trabajas mejor hoy mismo.

Cada parte lleva su nivel: **básico** (para todos), **intermedio** o **dev**. Toca el título para saltar a la parte.

**10** **Tu stack mínimo + veredicto** TODOS

El resumen de todo

## Diccionario rápido

*Por si alguna palabra te suena a chino. Una línea cada una, sin tecnicismos.*

| Palabra | En cristiano |
|---|---|
| Terminal | La ventana de texto donde escribes órdenes. Sin botones, solo letras. |
| Comando | Una orden que escribes para que pase algo. |
| Flag | Un ajuste que le añades a un comando; empieza por `--`. Ejemplo: `--model`. |
| Prompt | Lo que le pides a Claude en lenguaje normal. |
| Token | El “trozo” en que se mide el texto. Más tokens = más coste y más lento. |
| Contexto | Lo que Claude tiene en memoria ahora mismo, en esta sesión. |
| Sesión | Una conversación de trabajo: de cuando arrancas a cuando sales. |
| MCP | La forma de conectar Claude con otras apps (Drive, n8n…). |
| Hook | Un script tuyo que se dispara solo en cierto momento. |
| Skill | Una “habilidad” que le enseñas una vez y usa sola. |

EMPIEZA AQUÍ

# Instala Claude Code

ELIGE TU CAMINO · 5 MINUTOS

*Dos formas de tenerlo funcionando. La fácil es para empezar hoy sin complicarte. La avanzada es para quien ya vive en la terminal y quiere control. Las dos acaban en el mismo sitio: escribir «claude» y ponerte a trabajar.*

> ANTES DE NADA
>
> Claude Code necesita una **cuenta de pago** (Pro, Max, Team o Enterprise) o una clave de API de la consola. La cuenta **gratuita** de la web NO da acceso. Aparte de eso, solo te hace falta internet: no necesitas tarjeta gráfica.

## Camino fácil — para todos

BÁSICO

Un solo comando. No necesitas Node, ni configurar nada, y se actualiza solo. Es el método que recomienda Anthropic.

**1. Abre la terminal.** En Mac se llama Terminal; en Windows, PowerShell; en Linux, la tuya de siempre.

**2. Pega el instalador de una línea:**

```
# macOS, Linux o WSL
curl -fsSL https://claude.ai/install.sh | bash

# Windows  (en PowerShell, no en CMD)
irm https://claude.ai/install.ps1 | iex
```

**3. Escribe `claude`** y pulsa Enter. La primera vez se abre el navegador para iniciar sesión con tu cuenta de Claude.

**4. Comprueba que todo va:**

```
claude --version     # debe salir un número de versión
claude doctor        # chequeo de instalación y configuración
```

> **TRUCO**
> ¿La terminal te impone? Tienes el mismo Claude Code dentro de VS Code (extensión) y en la app de escritorio, con botones. Pero el instalador de una línea es más rápido de lo que parece: en dos minutos estás dentro.

### Camino avanzado — power users

DEV

Para quien ya tiene entorno de desarrollo y quiere fijar versiones, integrarlo en scripts o meterlo en el editor.

**Vía npm** (si ya usas Node 18+). Instala el mismo binario, pero lo actualizas tú a mano. **Nunca con `sudo`.**

```
npm install -g @anthropic-ai/claude-code
npm install -g @anthropic-ai/claude-code@latest   # para actualizar
```

**Fijar versión o canal** (instalador nativo): `claude install stable` (estable, ~1 semana de retraso), `latest`, o una versión concreta como `2.1.118`.

**Windows en serio:** para algo más que lo básico, usa **WSL2** — te da un entorno Unix donde Git, Docker y Node van finos. Git for Windows es opcional (habilita la herramienta Bash).

**Servidores, CI o Docker:** autentícate con una clave de API en vez del navegador, y usa el modo headless `-p` (Parte 1).

```
export ANTHROPIC_API_KEY=sk-...          # auth sin navegador
claude -p "revisa este diff" --output-format json
```

**En tu editor:** extensión de **VS Code** o **JetBrains** para tenerlo dentro del IDE; o `claude.ai/code` en el navegador para lanzar sesiones desde cualquier sitio, incluso el móvil.

### ¿Cuál elijo?

| Si eres… | Ve por… |
|---|---|
| Nuevo, o no quieres pelearte con nada | **Camino fácil** — instalador de una línea |
| Dev con Node y varios proyectos | **Avanzado** por npm, fijando versión |
| Windows y vas a usarlo en serio | **Avanzado** con WSL2 |
| Lo quieres en scripts o en un servidor | **Avanzado** con API key y modo `-p` |

0

PARTE 0 BÁSICO

# Cómo leer esto

PARA SITUARTE · 2 MINUTOS

*Tres cosas distintas que todo el mundo mete en el mismo saco y llama “comandos”. No son lo mismo, y distinguirlas es la mitad del manual.*

| Tipo | Qué es | Dónde vive |
|---|---|---|
| Comando de terminal | Lo escribes en la shell antes o alrededor de encender Claude | Tu terminal (bash / zsh) |
| Comando de sesión / | Built-in dentro de Claude Code | Dentro de la sesión |
| Patrón de prompt | Una instrucción en lenguaje normal disfrazada de comando | Cualquier chat — la barra es decorativa |

> **REGLA DE ORO**
> Si escribes / dentro de Claude Code y la palabra no sale en el desplegable, no es un comando real. Es texto. Punto.

## Dos niveles: oficial y comunidad

En este manual verás dos tipos de cosas. No las confundas:

- **Oficial** — está en la documentación de Claude Code. Existe y funciona (aunque los nombres cambien con las versiones).
- **Comunidad** — prácticas y prompts que mucha gente usa y que funcionan bien, pero no son comandos oficiales. Útiles, no referencia.

Las partes 1 a 4 y la 8 son sobre todo **oficial**. Las partes 5, 7 y 9 mezclan oficial con **comunidad**, y te lo aviso en cada una.

1

PARTE 1 BÁSICO

# Antes de encender Claude

ARRANQUE · GANA TIEMPO DESDE EL MINUTO UNO

*Esto se escribe en la shell, no dentro de Claude.*

## Instalar y mantener

```
# Instalación nativa (la recomendada ahora)
curl -fsSL https://claude.ai/install.sh | bash

claude update            # actualiza a la última versión
claude --version         # qué versión tienes
claude auth login        # iniciar sesión / cambiar de cuenta
claude doctor            # diagnóstico de configuración
```

Claude Code saca versión **casi cada semana**: los nombres cambian. Cuando dudes, `claude --help` o escribe `/` dentro de la sesión.

## Arrancar y retomar — los del día a día

| Comando | Para qué | Tu caso |
|---|---|---|
| `claude` | Abre sesión interactiva | El pan de cada día |
| `claude "explica esto"` | Sesión con un prompt inicial ya cargado | Arrancar un sub-bloque directo |
| `claude -c` | Retoma la última conversación de esta carpeta | “Seguimos donde lo dejé” |
| `claude -r "fase-4A"` | Retoma una sesión por nombre o ID | Volver a la FASE 4A de ayer |
| `claude -n "fase-4A"` | Le pones nombre a la sesión | Una sesión por sub-bloque |
| `claude -p "..."` | Modo headless: responde y sale | Scripts y automatización |

## Flags que de verdad te sirven

Van detrás del comando. Estos son los que encajan con tu flujo:

| Flag | Para qué | Tu caso |
|---|---|---|
| `--model opus` | Elige modelo al arrancar (`opus`/`sonnet`/`haiku`) | Opus para arquitectura, Sonnet para lo mecánico |
| `--add-dir ../otra` | Acceso a carpetas extra | Ver `~/DiarioVida/` y `~/Claude/hub/` a la vez |
| `--append-system-prompt` | Añade reglas tuyas sin romper lo de fábrica | Inyectar tu voz (Isra Bravo + Pratchett) |
| `-w nombre` | Arranca en un git worktree aislado | Probar un cambio gordo sin tocar tu rama buena |
| `--max-budget-usd 5` | Tope de gasto (modo `-p`) | Que una tarea autónoma no se dispare |
| `--safe-mode` | Arranca con todo lo tuyo desactivado | Cuando algo se rompe y no sabes qué config falla |

2

PARTE 2 BÁSICO

# El infierno de las confirmaciones

AHORRA HORAS · EVITA SUSTOS

*Por defecto Claude pide permiso antes de cada comando, edición o llamada de red. Es la red de seguridad. Pero en tareas largas, agota. Hay una escalera — de lo más sensato a lo más bestia. Para tu máquina real, no subas hasta arriba.*

## Escalón 1 · Shift + Tab (en caliente)

Dentro de la sesión, pulsa **Shift + Tab** para rotar entre modos de permiso: `default → acceptEdits → plan` (con `auto` y `bypassPermissions` añadidos al ciclo si los has habilitado). Sin reiniciar, reversible. **Empieza siempre por aquí.**

## Escalón 2 · Auto Mode (el recomendado)

Es el punto medio oficial y la **alternativa recomendada** al modo bestia. Auto-aprueba leer y editar ficheros, pero un clasificador de seguridad sigue revisando lo peligroso y bloquea por defecto `curl | bash`, force-push a `main`, despliegues a producción y borrados masivos.

```
claude --permission-mode auto      # arrancar directamente en Auto Mode
# …o entra con Shift+Tab hasta llegar a 'auto'
```

RECOMENDACIÓN

Este es tu modo por defecto. Te quita el 90 % de las interrupciones sin dejar que nada se cargue tu `.env`.

## Escalón 3 · Allowlist quirúrgica (la más segura)

En vez de abrir la mano entera, pre-apruebas solo comandos concretos y seguros. Lo más fino:

```
claude --allowedTools "Read" "Edit" "Bash(git status:*)" "Bash(git diff:*)"

# o permanente en ~/.claude/settings.json  ->  permissions.allow
# o en sesión con el comando  /permissions
```

### Escalón 4 · --dangerously-skip-permissions (YOLO)

Salta **absolutamente todas** las confirmaciones. Equivale a `--permission-mode bypassPermissions`.

```
claude --dangerously-skip-permissions
```

> **RIESGO**
> Se niega a arrancar como root. Hay incidentes documentados de pérdida total de datos (un `rm -rf` que se expandió al `~/` del usuario). Solo es razonable dentro de un contenedor desechable o CI. En tu ThinkCentre, con el `.env` y `marius.db`: ni se te ocurra. Para eso está el Auto Mode.

Si algún día quieres tenerlo disponible pero sin arrancar en él, usa `--allow-dangerously-skip-permissions`: lo añade al ciclo de Shift + Tab y entras solo cuando tú quieras.

> **CLAVE**
> Shift + Tab para el día a día → Auto Mode como modo por defecto → allowlist para lo que repitas mucho. El escalón 4, fuera de tu máquina.

3

PARTE 3 BÁSICO

# Dentro de la sesión

TU DÍA A DÍA

*Comandos que sí salen al escribir "/". Hay 60-90+; aquí los que de verdad tocas. Como cambian cada semana, la fuente de verdad es el desplegable de "/" o /help.*

## Sesión y contexto

| Comando | Para qué |
|---|---|
| `/clear` | Borra el contexto, empiezas limpio (la conversación sigue en disco) |
| `/compact` | Comprime el historial para liberar contexto — cada 20-30 min de trabajo largo |
| `/rewind` | Deshace y restaura contexto anterior (incluso revierte un `/clear`) |
| `/resume` | Retoma una sesión anterior |
| `/recap` | Resumen de lo que ha pasado en la sesión |

## Proyecto y memoria

| Comando | Para qué |
|---|---|
| `/init` | Crea el `CLAUDE.md` del proyecto (tu memoria persistente salió de aquí) |
| `/memory` | Edita lo que Claude recuerda del proyecto |
| `/config` | Ajustes de Claude Code |
| `#` al inicio | Añade una nota a la memoria sin abrir el archivo |

## Modelo y razonamiento

| Comando | Para qué |
|---|---|
| `/model` | Cambia modelo (Opus / Sonnet / Haiku) a media sesión |
| `/effort` | Sube o baja el esfuerzo de razonamiento |
| `think → ultrathink` | Escala de razonamiento como texto. Aquí es donde el "/ultrathink" viral sí es real |

## Calidad — tu obsesión: cero regresiones

| Comando | Para qué |
|---|---|
| `/review` · `/code-review` | Revisión del diff actual. Más fiable que pedir “revísame esto” |
| `/security-review` | Caza vulnerabilidades — antes de exponer cualquier endpoint del Hub al túnel |

## Permisos, MCP y autónomo

| Comando | Para qué |
|---|---|
| `/permissions` | Gestiona qué puede tocar Claude (la allowlist del Escalón 3) |
| `/mcp` | Servidores MCP conectados (n8n, Drive…) |
| `/agents` | Define y lanza subagentes |
| `/plan` | Enseña qué va a tocar y por qué antes de tocar nada — es tu método de sub-bloques |
| `/branch` (antes `/fork`) | Bifurca la conversación en una sesión nueva |
| `/btw` | Pregunta lateral sin ensuciar el contexto principal |

## Atajos de input que valen oro

```
@archivo.md   @carpeta/      # mete fichero o carpeta en contexto sin copiar-pegar
@https://url                 # mete el contenido de una URL (solo lectura)
!comando                     # ejecuta shell y mete la salida en el contexto
```

4

PARTE 4 BÁSICO

# Atajos de teclado

VELOCIDAD · ADIÓS AL RATÓN

*Varían un poco según terminal. En macOS, los Alt/Option necesitan activar “Option as Meta”. Si dudas: /terminal-setup.*

| Atajo | Qué hace | Cuándo |
|---|---|---|
| Shift + Tab | Rota el modo de permisos | Tu botón anti-confirmaciones |
| Esc | Interrumpe a Claude a media acción | Va por mal camino, lo paras |
| Esc Esc | Rebobina código y conversación a un punto anterior | Deshacer de verdad (mejor que Ctrl+C) |
| Alt + T · Meta + T | Activa o desactiva el razonamiento extendido | Subir el cerebro para algo difícil |
| Ctrl + B | Manda la tarea a segundo plano (tmux: dos veces) | Lanzar algo largo y seguir |
| Ctrl + X Ctrl + K | Mata todos los agentes en segundo plano | Un subagente se desboca |
| Ctrl + T | Muestra u oculta la lista de tareas | Ver en qué anda |
| Ctrl + L | Limpia la pantalla (conserva el historial) | Despejar ruido visual |
| Ctrl + R | Busca en el historial de prompts | Reutilizar un prompt anterior |
| Ctrl + G | Abre tu $EDITOR para un prompt largo | Prompts de varios párrafos |
| Ctrl + C / Ctrl + D | Cancelar / salir (no reasignables) | — |

# 5

PARTE 5 BÁSICO

## Prompts que usa la comunidad

COMUNIDAD · NO SON COMANDOS, PERO AYUDAN

*Estos NO son comandos oficiales: son patrones de prompt que mucha gente usa a diario. Los escribes como texto normal (la barra "/" es opcional). No hacen magia, pero encauzan bien la respuesta. Aquí van los que más se repiten.*

**Nivel: comunidad.** Ninguno está en la documentación oficial, salvo `ultrathink`, que además es una palabra clave real de razonamiento (Parte 3). El resto son prompts.

| # | Prompt | Qué hace |
|---|---|---|
| 1 | `ultrathink` | Máximo razonamiento (además, palabra clave real — Parte 3) |
| 2 | `godmode` | Sé exhaustivo, cubre todos los ángulos |
| 3 | `L99` | Nivel experto máximo, sin simplificar |
| 4 | `UDA` | Causa raíz "militar" (se solapa con inversion / premortem) |
| 5 | `OODA` | Bucle Observar-Orientar-Decidir-Actuar |
| 6 | `persona` | Fija un rol experto (equivalente real en la Parte 6) |
| 7 | `eli5` | Explica como a alguien sin contexto previo |
| 8 | `skeptic` | Cuestiona tu pregunta antes de responder |
| 9 | `noyap` | Corta preámbulo y relleno |
| 10 | `premortem` | Asume que el plan ya fracasó y busca el porqué |
| 11 | `tree` | Varias rutas antes de elegir la mejor |
| 12 | `falsify` | Intenta refutar la respuesta (se solapa con redteam) |
| 13 | `steelman` | Reconstruye la versión más fuerte de un argumento |
| 14 | `redteam` | Ataca la idea desde todos los ángulos |
| 15 | `inversion` | Qué garantizaría el fracaso, y lo evita (≈ premortem) |
| 16 | `socratic` | Te enseña a base de preguntas |
| 17 | `firstprinciples` | Descompone en verdades fundamentales |
| 18 | `devilsadvocate` | Defiende la postura contraria |
| 19 | `punch` | Recorta ~40 % y aprieta (oro para guiones) |
| 20 | `pitch` | Pitch de 30 s para alguien ocupado |

**HALLAZGO**

No necesitas los 20: varios se solapan (premortem ≈ inversion ≈ UDA; redteam ≈ devilsadvocate ≈ falsify). Con seis vas sobrado: noyap, punch, eli5, premortem, steelman, pitch. En la Parte 6 los conviertes en comandos reales tuyos.

6

PARTE 6 INTERMEDIO

# Convierte un prompt en comando real

HAZLO TUYO · REUTILIZA PARA SIEMPRE

*El siguiente paso natural: los prompts que más repitas se convierten en comandos reales tuyos guardándolos como un .md en ~/.claude/commands/. Luego salen al escribir "/" y se reutilizan en todos tus proyectos.*

## Ejemplo 1 · /noyap real — corta el relleno

Archivo: `~/.claude/commands/noyap.md`

```
---
description: Respuesta directa, sin preámbulo ni relleno
---
Responde sin introducción, sin resumen final y sin disculpas.
La respuesta primero. Si hay riesgo, una línea al final con un aviso.
```

## Ejemplo 2 · /punch real — aprieta guiones (tu caso estrella)

Archivo: `~/.claude/commands/punch.md`

```
---
argument-hint: [pega el guion]
description: Recorta el guion ~40% y aprieta cada linea
---
Recorta este texto ~40% sin perder el mensaje. Cada frase debe pegar
más fuerte. Mantén el cierre "Ahí lo dejo". Voz: directa estilo Isra
Bravo, ironía seca tipo Pratchett.

Texto: $ARGUMENTS
```

## Cómo se crea

```
mkdir -p ~/.claude/commands
# y dentro pones tus .md como los de arriba
```

— **De proyecto:** `.claude/commands/` dentro del repo → se comparte por git.

— **Personal:** `~/.claude/commands/` → disponible en todos tus proyectos.

— `$ARGUMENTS` captura lo que escribas tras el comando.

**RIESGO**

Los comandos consumen tokens (no son gratis). No tienen estado aislado: corren en el contexto actual, no en uno limpio. Y las librerías de terceros son código ajeno en tu máquina — revísalas antes de confiar.

7

PARTE 7 DEV

# Lo que casi nadie usa

NIVEL PRO · EFECTO WOW

*Todo lo que sigue es real y verificado hoy contra los docs oficiales. Es de la categoría "no sabía que se podía": funciones potentes que poca gente aprovecha.*

## 1 · Hooks — Claude ejecuta TUS scripts, siempre

En `settings.json` defines disparadores: que antes de cada edición pase tu linter, que al terminar suene un aviso, que se bloquee tocar `.env`. Esto **no** depende de la buena voluntad de Claude — lo ejecuta el harness, siempre. Es la diferencia entre **pedir** y **garantizar**.

```
// ~/.claude/settings.json
"hooks": {
  "PostToolUse": [{
    "matcher": "Edit|Write",
    "hooks": [{ "type": "command",
      "command": ".claude/hooks/format.sh" }]
  }]
}
```

El script recibe el evento en JSON por **stdin**; la ruta del fichero se lee con `jq -r '.tool_input.file_path'` — ojo: no existe una variable de entorno tipo `$CLAUDE_FILE_PATH`.

Eventos reales: `PreToolUse`, `PostToolUse`, `UserPromptSubmit`, `SessionStart`, `Stop`, `PreCompact`… (30+). Gestiónalos en sesión con `/hooks`.

## 2 · Skills — le enseñas algo una vez y lo usa solo

Carpeta `.claude/skills/<nombre>/SKILL.md` con un frontmatter (`description`) más tus instrucciones. Claude la carga **sola** cuando la tarea encaja. Es el paso natural tras los comandos custom de la Parte 6: un comando lo invocas tú; una skill se activa sola. Se comparte por git con el repo.

```
# .claude/skills/seo-diariovida/SKILL.md
---
name: seo-diariovida
description: Genera el paquete SEO de un video del canal
---
Cuando te pase un guion, devuelve titulos, descripcion y tags...
```

### 3 · Claude headless — dentro de tus scripts de bash

Con `-p` responde y sale; con `--output-format json` la salida es parseable. Aquí Claude deja de ser un chat y se vuelve **una pieza más de tu pipeline**. Esta sola línea impresiona a cualquier dev.

```
cat error.log | claude -p "resume la causa raiz" \
  --output-format json | jq .result
```

### 4 · @import dentro de CLAUDE.md

En tu memoria pones `Lee @docs/estilo.md` y Claude **importa ese fichero de verdad** (hasta 4 niveles de profundidad). Tu `CLAUDE.md` deja de ser un muro de texto y se vuelve un índice modular.

```
# CLAUDE.md
## Estilo
Sigue siempre las reglas de @docs/voz-paco.md
y los formatos de @docs/plantillas.md
```

### Extras que rematan

| Código | Qué hace |
|---|---|
| Ctrl + V  (Alt+V en Windows) | Pega una captura del portapapeles en el prompt → aparece un chip [Image #1]. Poca gente sabe que la terminal traga imágenes. |
| `claude mcp add --transport http <n> <url>` | Conecta un servidor MCP (Drive, n8n…) sin tocar JSON a mano. Lista con `claude mcp list`. |
| `/cost` | Cuánto llevas gastado en la sesión, en tokens y en dinero. |
| `/export` | Vuelca la conversación entera a fichero o portapapeles. |

> **BONUS**
> Menciona `@claude` en un PR o issue de GitHub (tras `/install-github-app`) y trabaja solo; o lanza sesiones desde el navegador en claude.ai/code. Es el “Claude programa mientras duermes”.

**ATENCIÓN — NO CAIGAS EN ESTO**

Dos comandos que NO debes incluir si quieres parecer al día: `/vim` (ahora vive en `/config → Editor mode`) y `/output-style` (ahora en `/config → Output style`). Si los ves en otro "manual de comandos", sabrás que copió de algo viejo.

8

PARTE 8 TODOS

# Ahorra tokens, tiempo y dinero

EL QUE MÁS TE PAGA LA LECTURA

*Cada token cuesta y cada minuto pelea contra tu ventana de trabajo. Estos hábitos te bajan la factura y te suben la velocidad casi sin enterarte. Si solo te llevas una página del manual, que sea esta.*

| Hábito | Qué te ahorra |
|---|---|
| `/compact` cada 20-30 min | Tokens y calidad — el contexto no se infla ni se ensucia |
| `/clear` al cambiar de tarea | Tokens y despistes — empiezas con la mente limpia |
| `@archivo` en vez de pegar | Tokens (mejor tokenización) y trazabilidad en el log |
| `/model haiku` para lo simple | Dinero — el modelo barato sobra para tareas mecánicas |
| `--max-budget-usd 5` | Sustos en la factura — corta solo al llegar al tope |
| `/cost` | Saber cuánto llevas gastado, en vivo |
| `--bare` para scripts | Tiempo de arranque y tokens de carga |

## No uses un Ferrari para ir a por el pan

El truco que más dinero ahorra es elegir bien el modelo. **Opus** para pensar y diseñar; **Sonnet** para ejecutar; **Haiku** para lo simple y rápido. Cambias a media sesión con `/model` o arrancas con `--model haiku`.

## Truco pro: que Opus planifique y Sonnet ejecute

DEV

El alias `opusplan` es lo mejor de los dos mundos: usa **Opus** mientras planificas (plan mode) y cambia **solo** a **Sonnet** al ejecutar. Piensas caro, ejecutas barato — sin tener que acordarte de cambiar a media tarea.

```
/model opusplan            # en sesión
claude --model opusplan    # al arrancar
/model opusplan[1m]        # además, contexto de 1M
```

Relacionados: `/advisor` deja a Opus disponible para consultas puntuales durante la ejecución (Anthropic midió −11,9 % de coste y +2,7 % de acierto). `/status` te dice qué modelo llevas ahora mismo.

**TRUCO**

El flujo que usan los equipos: arranca en Sonnet, sube a `/model opus` solo cuando te atascas, y baja a `/model haiku` para lo mecánico. Ahorro típico: 60-80 %.

**TRUCO**

Para tareas largas y repetitivas, deja un techo con `--max-budget-usd` y abre `/cost` de vez en cuando. Es la diferencia entre “me ha costado 2 €” y un susto a fin de mes.

**TRUCO**

¿Llamadas rápidas desde un script? `claude --bare` arranca sin cargar skills, hooks, MCP ni CLAUDE.md: empieza antes y gasta menos. Para el trabajo normal, déjalo como está.

**POR QUÉ IMPORTA**

Una sesión larga sin limpiar no solo gasta más: responde peor, porque Claude arrastra ruido. Limpiar a tiempo es a la vez más barato y más listo. Gana-gana.

9

PARTE 9 INTERMEDIO

# Lo que se está volviendo viral

LO QUE USA LA GENTE AHORA · VERIFICADO

*Esto es lo que de verdad usa la comunidad en 2026: formas de trabajar, no palabras mágicas. Todo verificado; he marcado lo que tiene truco.*

## ★ El CLAUDE.md que se mejora solo

La práctica más viral del año (le llaman **"compounding engineering"**). Cada vez que Claude se equivoca, en lugar de solo corregirle, le dices que lo apunte. Claude escribe sus **propias** reglas, y esa corrección mejora **todas** tus sesiones futuras.

```
# justo después de corregir algo:
Actualiza tu CLAUDE.md para no volver a cometer este error.
```

**Por qué la usa la gente:** el sistema se vuelve más listo solo, sin esfuerzo extra. Es el tip oficial de Anthropic con más recorrido.

## ★ "Entrevístame antes de hacer nada"

En vez de soltar la orden y cruzar los dedos, empiezas pidiéndole que **te pregunte**. Claude saca a la luz lo que no habías pensado: casos límite, decisiones, detalles. Y no es solo para código — sirve para un artículo, una estrategia o un guion.

```
Antes de escribir nada, hazme las preguntas que necesites
para hacerlo bien. Una a una.
```

**Por qué la usa la gente:** entrevistarte primero da mejor resultado que ir directo. Y es de lo más accesible para quien no programa.

## ★ Explorar → Planificar → Programar → Confirmar

Ya casi nadie suelta "hazlo" a pelo. El flujo que recomienda Anthropic: primero **plan mode** (Claude lee y entiende sin tocar nada), luego un plan por escrito, luego ejecutar contra ese plan, y al final el commit. Menos retrabajo, menos sustos. Enlaza directo con tu `/plan` de la Parte 3.

## Lo demás que circula (con su veredicto)

**Leyenda:** ✓ funciona de verdad · ~ funciona pero con truco

| Lo que circula | ¿Va? | Por qué la gente lo usa |
|---|---|---|
| Añadir “use subagents” al final de una orden | ✓ | Lanza más cómputo a un problema difícil sin ensuciar tu sesión |
| `/loop` y `/schedule` (tareas que siguen solas) | ✓ | Automatizan PRs y recados hasta con el portátil cerrado |
| “Enséñame la prueba, no me digas que va” | ✓ | El tip nº 1 de Anthropic: que Claude demuestre con tests o captura |
| El CLAUDE.md de Karpathy (“grill-me”, 140k★) | ✓ | Reglas anti-cagada: no asumir, no sobre-ingeniería, no tocar de más |
| “Context engineering” por encima del prompt | ✓ | En 2026 el contexto pesa más que la frase mágica |
| Skill “últimos 30 días” (escanea Reddit, X, web) | ~ | Útil para temas, pero es skill de terceros: revísala antes |
| Fallback a Gemini CLI para webs bloqueadas | ~ | Truco real pero gris: manda tu consulta a otro sitio |

**EL PATRÓN DE 2026**

El patrón claro de 2026: lo que de verdad ayuda son formas de trabajar — planificar antes, verificar siempre, y dejar que tu CLAUDE.md aprenda de cada error. Eso rinde más que cualquier “comando mágico”.

10

PARTE 10 TODOS

# Tu stack mínimo + veredicto

EL RESUMEN DE TODO

## El starter pack que de verdad usas tú

```
claude -n "fase-4A"           # arrancas con nombre de sesión
# Shift+Tab hasta Auto Mode    # callas las confirmaciones con cabeza
/plan                          # le haces enseñar el plan antes de tocar
/compact                       # cada 20-30 min
/code-review                   # tras cada sub-bloque
/clear                         # al empezar el siguiente sub-bloque
claude -c                      # mañana retomas donde lo dejaste
```

## Lo que te llevas

**CLAVE**

Lo tienes todo en un sitio: instalar Claude Code, usarlo el día a día y los comandos y trucos que más usa la comunidad. Del primer comando al nivel pro.

**RECOMENDACIÓN**

Tu problema de las confirmaciones se arregla con Auto Mode (Shift + Tab), no con el YOLO que circula por internet. El YOLO, fuera de tu máquina.

**HALLAZGO**

Jugada de contenido: “De cero a power user con Claude Code: instalación, los comandos que de verdad usa la comunidad y los trucos que te ahorran horas y tokens.” Práctico y verificado. Tu marca.

**PARA TERMINAR**

No te agobies: nadie usa los noventa comandos. Elige tres, conviértelos en costumbre y

> vuelve aquí a por el cuarto cuando te apetezca. Esto es una caja de herramientas — se coge la que hace falta, no todas a la vez.

---

*Verificado contra la documentación oficial de Claude Code (code.claude.com/docs) · Junio 2026. Como Claude Code cambia casi cada semana, valida nombres con "/" en tu sesión o claude --help.*

PARA CONSULTAR

# Índice de comandos

*Busca cualquier comando y salta a su parte. Ordenados alfabéticamente.*

| Comando | Dónde |
|---|---|
| # (añadir a memoria) | Parte 3 |
| `--add-dir` | Parte 1 |
| `/advisor` | Parte 8 |
| `/agents` | Parte 3 |
| `Alt+T` | Parte 4 |
| `--append-system-prompt` | Parte 1 |
| `@archivo` | Parte 3 |
| `--bare` | Parte 8 |
| `/branch` | Parte 3 |
| `/btw` | Parte 3 |
| `-c` / `--continue` | Parte 1 |
| `claude` | Parte 1 |
| `claude --version` | Instalación |
| `claude auth login` | Parte 1 |
| `claude doctor` | Instalación |
| `claude install` | Instalación |
| `claude mcp add` | Parte 7 |
| `claude update` | Instalación |
| `/clear` | Parte 3 |
| `/code-review` | Parte 3 |
| `!comando` | Parte 3 |
| `/compact` | Parte 3 |
| `/config` | Parte 3 |
| `/cost` | Parte 7 |
| `Ctrl+G` | Parte 4 |
| `Ctrl+V` (pegar imagen) | Parte 7 |

| Comando | Dónde |
|---|---|
| `Ctrl+X Ctrl+K` | Parte 4 |
| `--dangerously-skip-permissions` | Parte 2 |
| `/effort` | Parte 3 |
| `Esc` · `Esc Esc` | Parte 4 |
| `/export` | Parte 7 |
| `/hooks` | Parte 7 |
| Hooks (`settings.json`) | Parte 7 |
| `/init` | Parte 3 |
| `--max-budget-usd` | Parte 8 |
| `/mcp` | Parte 3 |
| `/memory` | Parte 3 |
| `--model` | Parte 8 |
| `/model` | Parte 8 |
| `/model opusplan` | Parte 8 |
| `-n` / `--name` | Parte 1 |
| `-p` / `--print` (headless) | Parte 1 |
| `--permission-mode auto` | Parte 2 |
| `/permissions` | Parte 3 |
| `/plan` | Parte 3 |
| `-r` / `--resume` | Parte 1 |
| `/recap` | Parte 3 |
| `/review` | Parte 3 |
| `/rewind` | Parte 3 |
| `--safe-mode` | Parte 1 |
| `/security-review` | Parte 3 |
| `Shift + Tab` | Parte 2 |
| Skills (`SKILL.md`) | Parte 7 |
| `/status` | Parte 8 |
| `ultrathink` | Parte 3 |
| `@url` | Parte 3 |
| `-w` / `--worktree` | Parte 1 |

GRACIAS POR LEER

## Sigamos en contacto

*Si algo de aquí te ha ahorrado tiempo, cuéntamelo. Y si quieres más IA práctica, sin humo y al grano, aquí me tienes:*

**Web y newsletter** diariovida.com

**Escríbeme** pacovida@diariovida.com

**YouTube** @Paco_Vida

*Ahí lo dejo.*