Aprende Claude Code

Guía completa en español · 2026

claude-rho-snowy.vercel.app

Contents

1 Introducción: ¿Qué es Claude Code? 9

1.1 Contenido del tutorial . . . 9
1.1.1 Instalación . . . 9
1.1.2 Primeros pasos . . . 9
1.1.3 CLI, app y móvil . . . 9
1.1.4 Recetas prácticas . . . 9
1.1.5 Proyectos guiados . . . 9
1.1.6 Escribir buenos prompts . . . 9
1.1.7 Glosario . . . 9
1.1.8 Skills . . . 10
1.1.9 Subagentes . . . 10
1.1.10 Plugins . . . 10
1.1.11 Flujos de trabajo pro . . . 10
1.1.12 Comandos . . . 10
1.1.13 Configuración . . . 10
1.1.14 Servidores MCP . . . 10
1.1.15 Hooks . . . 10
1.1.16 Permisos . . . 10
1.1.17 Uso avanzado . . . 10
1.1.18 Preguntas frecuentes . . . 10
1.1.19 Solución de problemas . . . 10
1.1.20 Recursos . . . 10
1.1.21 Comparativa . . . 10
1.1.22 Pymes y oficina . . . 11
1.1.23 Perfiles técnicos . . . 11
1.2 ¿Qué es Claude Code? . . . 11

2 Instalación 12

2.1 Requisitos previos . . . 12
2.2 Paso 1: Instalar Claude Code . . . 12
2.2.1 Alternativa: con npm . . . 12
2.3 Paso 2: Obtener tu API key . . . 13
2.4 Paso 3: Configurar la API key . . . 13
2.4.1 Opción A: Variable de entorno (recomendado) . . . 13
2.4.2 Opción B: Al iniciar Claude Code . . . 13
2.5 Paso 4: Primera ejecución . . . 13
2.6 Instalación en Windows . . . 14
2.7 Actualizar Claude Code . . . 14
2.8 Desinstalar . . . 14

3 Primeros pasos **15**
3.1 Iniciar una sesión . . . 15
3.2 Tu primera tarea . . . 15
3.3 Entender el flujo de trabajo . . . 15
3.4 Modos de operación . . . 16
3.4.1 Modo normal (por defecto) . . . 16
3.4.2 Modo auto . . . 16
3.4.3 Modo plan . . . 16
3.5 Ejemplos de primeras tareas . . . 16
3.5.1 Entender un proyecto existente . . . 16
3.5.2 Crear un archivo nuevo . . . 16
3.5.3 Arreglar un error . . . 17
3.5.4 Ejecutar tests . . . 17
3.6 Salir de Claude Code . . . 17
3.7 Iniciar con un prompt directo . . . 17
3.8 Modo ”print” (salida directa) . . . 17

4 CLI, app de escritorio, web y móvil **18**
4.1 Las superficies de Claude Code . . . 18
4.2 Terminal (CLI) . . . 18
4.3 App de escritorio (Mac y Windows) . . . 19
4.4 Web (claude.ai/code) . . . 19
4.5 Extensiones de IDE (VS Code, JetBrains) . . . 19
4.6 Controlar Claude Code desde el móvil . . . 19
4.7 ¿Cuál elijo? . . . 20

5 Recetas prácticas **21**
5.1 Aprender a programar . . . 21
5.1.1 Pedir una explicación a tu nivel . . . 21
5.1.2 Aprender haciendo un mini-proyecto guiado . . . 21
5.1.3 Entender un concepto que se te resiste . . . 21
5.1.4 Practicar con ejercicios . . . 22
5.2 Crear tu primer proyecto . . . 22
5.2.1 Una página web personal . . . 22
5.2.2 Una pequeña app de tareas (to-do list) . . . 22
5.2.3 Un script útil para tu día a día . . . 22
5.3 Entender código que no escribiste . . . 22
5.3.1 Entender un proyecto entero . . . 23
5.3.2 Explicar un archivo o función concreta . . . 23
5.3.3 Traducir jerga técnica . . . 23
5.4 Resolver errores . . . 23
5.4.1 Entender y arreglar un error . . . 23
5.4.2 Cuando algo ”no funciona” pero no hay error . . . 23
5.4.3 Arreglar los tests que fallan . . . 24
5.5 Mejorar tu código . . . 24
5.5.1 Pedir una revisión como si fuera un mentor . . . 24
5.5.2 Hacer el código más legible . . . 24
5.5.3 Añadir comprobaciones de seguridad . . . 24
5.6 Git y control de versiones . . . 24
5.6.1 Empezar a usar git sin saber git . . . 24
5.6.2 Guardar tu trabajo con un buen mensaje . . . 25
5.6.3 ”He liado algo, quiero volver atrás” . . . 25

5.6.4 Subir el proyecto a GitHub . . . 25
5.7 Trabajar con datos y archivos . . . 25
5.7.1 Analizar un archivo Excel o CSV . . . 25
5.7.2 Convertir entre formatos . . . 25
5.7.3 Limpiar datos desordenados . . . 25
5.7.4 Generar un gráfico . . . 26
5.8 Automatizar tareas repetitivas . . . 26
5.8.1 Organizar la carpeta de descargas . . . 26
5.8.2 Renombrar muchos archivos a la vez . . . 26
5.8.3 Buscar texto en muchos archivos . . . 26
5.9 Comandos de terminal sin miedo . . . 26
5.9.1 "¿Cómo se hace esto en la terminal?" . . . 26
5.9.2 Instalar herramientas . . . 27
5.9.3 Antes de ejecutar un comando que viste en internet . . . 27
5.10 Documentar y explicar . . . 27
5.10.1 Crear un README para tu proyecto . . . 27
5.10.2 Comentar tu propio código . . . 27
5.10.3 Preparar una explicación para presentar tu trabajo . . . 27

6 Proyectos guiados **28**
6.1 Proyecto 1: Tu web personal . . . 28
6.2 Proyecto 2: App de lista de tareas . . . 29
6.3 Proyecto 3: Un script útil en Python . . . 30

7 Cómo escribir buenos prompts **32**
7.1 1. Sé específico, no genérico . . . 32
7.2 2. Di tu nivel y pide explicaciones . . . 32
7.3 3. Pide ver antes de actuar . . . 32
7.4 4. Da contexto: pega errores, datos y ejemplos . . . 32
7.5 5. Divide las tareas grandes . . . 33
7.6 6. Describe el resultado que quieres, no la solución técnica . . . 33
7.7 7. Pide que te corrija y te enseñe . . . 33
7.8 Plantillas listas para usar . . . 33
7.8.1 Para crear algo nuevo . . . 33
7.8.2 Para arreglar algo . . . 33
7.8.3 Para entender algo . . . 34
7.8.4 Para mejorar lo que ya tienes . . . 34
7.9 Errores comunes al empezar . . . 34

8 Glosario **35**
8.0.1 Terminal (o consola) . . . 35
8.0.2 CLI . . . 35
8.0.3 Comando . . . 35
8.0.4 Directorio . . . 35
8.0.5 Repositorio (repo) . . . 35
8.0.6 git . . . 35
8.0.7 Commit . . . 36
8.0.8 GitHub . . . 36
8.0.9 Branch (rama) . . . 36
8.0.10 Pull Request (PR) . . . 36
8.0.11 API . . . 36
8.0.12 API key . . . 36

8.0.13 Frontend . . . 36
8.0.14 Backend . . . 36
8.0.15 Base de datos . . . 36
8.0.16 Framework / Librería . . . 37
8.0.17 Dependencia . . . 37
8.0.18 npm . . . 37
8.0.19 Paquete . . . 37
8.0.20 Función . . . 37
8.0.21 Variable . . . 37
8.0.22 Bug . . . 37
8.0.23 Debug (depurar) . . . 37
8.0.24 Stack trace / Traceback . . . 37
8.0.25 Deploy (desplegar) . . . 37
8.0.26 Servidor . . . 37
8.0.27 localhost . . . 38
8.0.28 JSON . . . 38
8.0.29 Entorno (environment) . . . 38
8.0.30 Variable de entorno . . . 38
8.0.31 Hook . . . 38
8.0.32 MCP . . . 38

9 Claude Code para pymes y oficina 39
9.1 Hojas de cálculo y datos . . . 39
9.1.1 Resumen de ventas para una reunión . . . 39
9.1.2 Limpiar una lista de contactos . . . 39
9.1.3 Cruzar clientes y pedidos . . . 40
9.2 Documentos y plantillas . . . 40
9.2.1 Facturas desde una plantilla . . . 40
9.2.2 Resumir PDFs de una carpeta . . . 40
9.2.3 Crear una plantilla de presupuesto . . . 40
9.3 Automatizar tareas repetitivas . . . 40
9.3.1 Organizar una carpeta caótica . . . 40
9.3.2 Renombrar archivos por lote . . . 41
9.3.3 Programa pequeño para pedidos diarios . . . 41
9.4 Informes y análisis . . . 41
9.4.1 Gráfico de gastos por categoría . . . 41
9.4.2 Informe mensual de ventas . . . 41
9.5 Comunicación y clientes . . . 41
9.5.1 Detectar problemas repetidos en reseñas . . . 41
9.5.2 Plantillas de email . . . 42
9.6 Tu presencia online sin saber programar . . . 42
9.6.1 Web de una sola página . . . 42

10 Para perfiles técnicos y equipos 43
10.1 Bases de código grandes . . . 43
10.1.1 Generar contexto inicial del repositorio . . . 43
10.2 Revisión de código automatizada . . . 43
10.2.1 Revisar el diff local . . . 43
10.2.2 Revisión desde GitHub CLI . . . 43
10.3 Refactors y migraciones a gran escala . . . 44
10.3.1 Migración con plan previo . . . 44
10.4 Testing y TDD . . . 44

10.4.1 Tests unitarios para un módulo . . . 44
10.4.2 Trabajar en TDD . . . 44
10.5 CI/CD y modo headless . . . 44
10.5.1 Ejemplo en GitHub Actions . . . 44
10.6 Estandarizar el equipo . . . 45
10.6.1 Skill interna para antes del PR . . . 45
10.7 Integraciones . . . 45
10.7.1 Consultar datos mediante MCP . . . 45

11 Skills **46**
11.1 ¿Qué es una Skill? . . . 46
11.2 Estructura de una Skill . . . 46
11.2.1 Ejemplo de SKILL.md . . . 46
11.2.2 Campos del frontmatter . . . 47
11.3 Dónde se guardan . . . 47
11.4 Cómo invocar una Skill . . . 47
11.4.1 Manualmente . . . 47
11.4.2 Automáticamente . . . 48
11.5 Crear tu primera Skill (sin saber) . . . 48
11.6 Skills oficiales incluidas . . . 48
11.7 Edición en vivo . . . 48

12 Subagentes **49**
12.1 ¿Para qué sirven? . . . 49
12.2 Crear un subagente . . . 49
12.2.1 Ejemplo: un subagente revisor de código . . . 49
12.2.2 Campos del frontmatter . . . 50
12.3 Dónde se guardan . . . 50
12.4 Cómo invocarlos . . . 50
12.4.1 Delegación natural . . . 50
12.4.2 Mención directa . . . 51
12.4.3 Asistente interactivo . . . 51
12.4.4 Desde la terminal . . . 51
12.5 Crear uno sin saber YAML . . . 51
12.6 Subagentes en paralelo (ejemplo real) . . . 51

13 Plugins **52**
13.1 ¿Qué es un plugin? . . . 52
13.2 Añadir un marketplace . . . 52
13.3 Instalar un plugin . . . 52
13.4 Gestionar plugins . . . 53
13.5 Plugins útiles que la gente instala . . . 53
13.6 Empezar rápido con plugins . . . 53
13.7 Crear tu propio plugin . . . 53

14 Flujos de trabajo pro **54**
14.1 Plan Mode (modo planificación) . . . 54
14.1.1 Cómo activarlo . . . 54
14.2 Checkpoints y Rewind (deshacer) . . . 54
14.2.1 Cómo deshacer . . . 54
14.3 Tareas en background . . . 55
14.3.1 Cómo usarlo . . . 55

14.4 Output styles (formato de salida) . . . 55
14.5 Los flujos que más se recomiendan . . . 55
14.6 Comprueba tu versión . . . 56

15 Comandos **57**
15.1 Slash commands . . . 57
15.2 Flags de la CLI . . . 58
15.3 Modelos disponibles . . . 58
15.4 Atajos de teclado en sesión interactiva . . . 59
15.5 Comandos de ejemplo . . . 59
15.5.1 Sesión headless en CI/CD . . . 59
15.5.2 Cambiar modelo para una sesión . . . 59
15.5.3 Ver diagnóstico de instalación . . . 59

16 Configuración **60**
16.1 Archivo de configuración principal . . . 60
16.1.1 Ejemplo de settings.json . . . 60
16.2 CLAUDE.md — Memoria del proyecto . . . 61
16.2.1 Crear CLAUDE.md automáticamente . . . 61
16.2.2 Estructura recomendada de CLAUDE.md . . . 61
16.3 Variables de entorno . . . 61
16.4 Modelo por defecto . . . 62
16.5 Integración con VS Code . . . 62
16.6 Integración con JetBrains . . . 62
16.7 Configuración de proxy . . . 62
16.8 Múltiples perfiles / proyectos . . . 63

17 Servidores MCP **64**
17.1 ¿Qué es MCP? . . . 64
17.2 Cómo funciona . . . 64
17.3 Añadir un servidor MCP . . . 64
17.4 Servidores MCP populares . . . 65
17.5 Gestionar servidores MCP . . . 66
17.6 Ámbito de los servidores MCP . . . 66
17.7 Crear tu propio servidor MCP . . . 66
17.7.1 Ejemplo mínimo en TypeScript . . . 66
17.8 MCP en la nube vs local . . . 67

18 Hooks **68**
18.1 ¿Qué son los hooks? . . . 68
18.2 Tipos de hooks . . . 68
18.3 Configurar hooks . . . 68
18.4 Variables de entorno disponibles en hooks . . . 69
18.5 El matcher . . . 69
18.6 Hook de tipo command . . . 70
18.6.1 Ejemplo: hook que lee datos del evento por stdin . . . 70
18.7 Bloquear operaciones con PreToolCall . . . 70
18.8 Ejemplos prácticos . . . 71
18.8.1 Formatear con Prettier tras edición . . . 71
18.8.2 Ejecutar tests tras cambios . . . 71
18.8.3 Log de todas las operaciones . . . 71
18.9 Gestionar hooks desde la CLI . . . 72

19 Permisos **73**
19.1 Filosofía de permisos . . . 73
19.2 Herramientas y sus permisos . . . 73
19.3 Permitir y denegar operaciones . . . 73
19.3.1 Sintaxis de permisos . . . 74
19.4 Gestión interactiva de permisos . . . 74
19.5 Permisos durante una sesión . . . 74
19.6 Modo sin permisos (peligroso) . . . 75
19.7 Buenas prácticas de seguridad . . . 75
19.8 Configuración de permisos para equipo . . . 75

20 Uso avanzado **76**
20.1 Subagentes . . . 76
20.2 Git worktrees . . . 76
20.3 Modo headless / CI-CD . . . 77
20.4 Modo /fast (Opus acelerado) . . . 77
20.5 Sesiones largas y compactación . . . 77
20.5.1 Compactar el contexto . . . 77
20.5.2 Reanudar una sesión . . . 78
20.6 CLAUDE.md en subdirectorios . . . 78
20.7 Flujos de trabajo con git . . . 78
20.7.1 Crear feature branches automáticamente . . . 78
20.7.2 Code review automatizado . . . 78
20.8 Integración con tmux / pantallas divididas . . . 78
20.9 Tips de productividad . . . 79
20.10Exportar la sesión . . . 79

21 Preguntas frecuentes **80**
21.1 Coste y cuenta . . . 80
21.2 Seguridad y privacidad . . . 80
21.3 Uso y aprendizaje . . . 80
21.4 Comparativas . . . 80

22 Solución de problemas **81**
22.1 Al instalar . . . 81
22.1.1 ”command not found: claude” . . . 81
22.1.2 ”EACCES: permission denied” al instalar . . . 81
22.1.3 ”Node.js version too old” / versión incompatible . . . 81
22.2 Al iniciar sesión / autenticación . . . 82
22.2.1 ”Invalid API key” o errores de autenticación . . . 82
22.2.2 ”Rate limit exceeded” / ”Too many requests” . . . 82
22.2.3 ”Insufficient credit” / saldo agotado . . . 82
22.3 Durante el uso . . . 82
22.3.1 Claude Code va lento o se ”atasca” . . . 82
22.3.2 Hace cambios que yo no quería . . . 82
22.3.3 No me pide permiso / me pide demasiado permiso . . . 82
22.3.4 Una función de la documentación no me aparece . . . 82
22.3.5 Un servidor MCP no conecta . . . 83
22.4 El comodín que siempre funciona . . . 83

23 Recursos **84**
23.1 Documentación oficial . . . 84
23.2 Skills . . . 84
23.3 Herramientas . . . 85
23.4 MCP (Model Context Protocol) . . . 85
23.5 Cursos y tutoriales en español (YouTube) . . . 85
23.6 Tutoriales en inglés . . . 85
23.7 Cuentas de X para estar al día . . . 85

24 Comparativa **86**
24.1 ¿Cuándo elegir Claude Code? . . . 87
24.2 Claude Code + tu editor (no es o lo uno o lo otro) . . . 87
24.3 Resumen . . . 87

25 Sugerencias y contacto **88**

Chapter 1

Introducción: ¿Qué es Claude Code?

Actualizado junio 2026 · Claude Code 2.xLa guía más completa en español para dominar Claude Code, la CLI de Anthropic que convierte tu terminal en un asistente de IA experto. Sin conocimientos previos necesarios.

Empezar ahora Ver recetas prácticasInstalación rápida

```
npm install -g @anthropic-ai/claude-code
```

Requiere Node.js 20+ · Funciona en macOS, Linux y WSL

1.1 Contenido del tutorial

1.1.1 Instalación

Instala Claude Code en macOS, Linux o Windows en menos de 2 minutos.

1.1.2 Primeros pasos

Tu primera sesión: cómo hablar con Claude Code y entender la interfaz.

1.1.3 CLI, app y móvil

Terminal, app de escritorio, web, IDE y cómo controlarlo desde el móvil.

1.1.4 Recetas prácticas

Más de 40 ejemplos reales del día a día, con prompts listos para copiar.

1.1.5 Proyectos guiados

Construye una web, una app y un script paso a paso, con todos los prompts.

1.1.6 Escribir buenos prompts

Cómo pedir las cosas para obtener mejores resultados. Antes y después.

1.1.7 Glosario

Términos técnicos explicados con palabras normales y analogías.

1.1.8 Skills

Enseña a Claude tareas a tu manera con archivos SKILL.md reutilizables.

1.1.9 Subagentes

Ayudantes especializados que trabajan en paralelo mientras tú revisas.

1.1.10 Plugins

Instala bundles de skills, agentes y MCP desde el marketplace.

1.1.11 Flujos de trabajo pro

Plan mode, rewind, tareas en background y los trucos más recomendados.

1.1.12 Comandos

Todos los slash commands y flags disponibles con ejemplos reales.

1.1.13 Configuración

Personaliza el comportamiento, modelos y memoria con settings.json.

1.1.14 Servidores MCP

Conecta herramientas externas: bases de datos, APIs, navegador y más.

1.1.15 Hooks

Automatiza acciones antes y después de cada herramienta o respuesta.

1.1.16 Permisos

Controla qué puede hacer Claude Code en tu máquina con seguridad.

1.1.17 Uso avanzado

Subagentes, worktrees, modo headless y flujos de trabajo pro.

1.1.18 Preguntas frecuentes

Precio, seguridad, privacidad y las dudas más comunes al empezar.

1.1.19 Solución de problemas

Errores comunes de instalación y uso, y cómo resolverlos rápido.

1.1.20 Recursos

Enlaces oficiales, repos de skills, MCP y cursos actualizados.

1.1.21 Comparativa

Claude Code frente a Cursor, Windsurf, Copilot y ChatGPT.

1.1.22 Pymes y oficina

Automatiza Excel, facturas, informes y tareas de oficina sin programar.

1.1.23 Perfiles técnicos

Code review, refactors, testing, CI/CD y estandarización de equipo.

1.2 ¿Qué es Claude Code?

Claude Code es una CLI (interfaz de línea de comandos) desarrollada por Anthropic que te permite interactuar con el modelo Claude directamente desde tu terminal o desde tu editor de código favorito (VS Code, JetBrains, etc.).

A diferencia de los chatbots web, Claude Code tiene acceso **directo a tus archivos**, puede ejecutar comandos de terminal, leer y escribir código, hacer git commits, buscar en tu base de código y mucho más — todo con tu supervisión.

En resumen, Claude Code es:

- **Agente de código:** lee, edita y crea archivos directamente en tu proyecto.
- **Extensible:** conecta herramientas externas vía Model Context Protocol (MCP).
- **Seguro:** tú controlas cada permiso. Nada ocurre sin tu aprobación.

Chapter 2

Instalación

Instala Claude Code en tu sistema en un par de minutos. Solo necesitas una cuenta de Anthropic (suscripción de Claude o cuenta de la consola).

2.1 Requisitos previos

- **Sistema operativo:** macOS, Linux, o Windows (con WSL o nativo).
- **Cuenta de Anthropic** — una suscripción de Claude o una cuenta de la consola (`console.anthropic.com`).
- **Node.js 20+** — *solo* si instalas por npm. Con el instalador nativo (recomendado) no hace falta.

Nota

El método recomendado por Anthropic es el instalador nativo, que no depende de Node.js y se actualiza solo. Si prefieres npm, necesitarás Node.js 20+ (descárgalo desde nodejs.org, versión LTS).

2.2 Paso 1: Instalar Claude Code

Abre tu terminal y usa el **instalador nativo** (recomendado; no necesita Node.js y se actualiza solo):

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

# Windows (PowerShell)
irm https://claude.ai/install.ps1 | iex
```

En Mac también puedes usar Homebrew:

```
brew install --cask claude-code
```

2.2.1 Alternativa: con npm

Si prefieres npm (requiere Node.js 20+):

```
npm install -g @anthropic-ai/claude-code
```

Cualquiera de los métodos instala el comando `claude`. Verifica que se instaló correctamente:

```
claude --version
```

2.3 Paso 2: Obtener tu API key

1. Entra a **console.anthropic.com** y crea una cuenta si no tienes.
2. Ve a **API Keys** en el panel lateral.
3. Haz clic en **Create Key** y copia la clave generada.
4. Guárdala en un lugar seguro — solo se muestra una vez.

Atención

Atención: Las API keys tienen costos por uso. Anthropic ofrece créditos gratuitos al registrarse. Revisa los precios en anthropic.com/pricing.

2.4 Paso 3: Configurar la API key

Tienes dos opciones para configurar tu clave:

2.4.1 Opción A: Variable de entorno (recomendado)

Añade esto a tu `~/.zshrc`, `~/.bashrc` o equivalente:

```
export ANTHROPIC_API_KEY="sk-ant-xxxxxxxxxxxxxxxxxxxxxxx"
```

Luego recarga tu shell:

```
source ~/.zshrc   # o ~/.bashrc
```

2.4.2 Opción B: Al iniciar Claude Code

Si no configuraste la variable, Claude Code te pedirá la clave la primera vez que lo ejecutes y la guardará automáticamente.

2.5 Paso 4: Primera ejecución

Navega a un directorio de trabajo y ejecuta:

```
cd mi-proyecto
claude
```

La primera vez te puede pedir autorizaciones o confirmar la clave. Después verás el prompt interactivo de Claude Code listo para usar.

2.6 Instalación en Windows

En Windows tienes dos caminos:

1. **Nativo (más sencillo):** abre PowerShell y ejecuta `irmhttps://claude.ai/install.ps1|iex`. Se recomienda tener **Git para Windows** instalado para que Claude pueda usar Bash.
2. **Con WSL2:** instala WSL con `wsl--install` y luego instala Claude Code dentro de WSL igual que en Linux.

Nota

Claude Code también tiene extensión para VS Code y JetBrains IDE — las instalas desde el marketplace de tu editor. Ver sección de configuración para más detalles.

2.7 Actualizar Claude Code

Con el **instalador nativo** se actualiza solo en segundo plano: no tienes que hacer nada. Comprueba tu versión con:

```
claude --version
```

Con Homebrew: `brewupgradeclaude-code`. Si instalaste por npm:

```
npm update -g @anthropic-ai/claude-code
```

2.8 Desinstalar

Según cómo lo instalaste:

```
# Homebrew
brew uninstall --cask claude-code

# npm
npm uninstall -g @anthropic-ai/claude-code
```

Chapter 3

Primeros pasos

Aprende a iniciar Claude Code, entender su interfaz y completar tus primeras tareas de programación con IA.

3.1 Iniciar una sesión

Abre tu terminal, navega al directorio de tu proyecto y ejecuta:

```
claude
```

Verás un prompt interactivo como este:

```
+==================================+
|  Claude Code  ·  claude-sonnet  |
+==================================+
> _
```

Claude Code está listo para recibir instrucciones en lenguaje natural. Puedes escribirle como si fuera un compañero de trabajo.

3.2 Tu primera tarea

Prueba algo sencillo para empezar. Escribe:

```
> ¿Qué archivos hay en este directorio?
```

Claude Code ejecutará `ls` o `find` y te mostrará el resultado. Observarás que antes de ejecutar cualquier herramienta te pide confirmación o te muestra lo que va a hacer.

3.3 Entender el flujo de trabajo

Claude Code opera en ciclos de:

1. **Pensar** — analiza tu petición y planifica los pasos.
2. **Actuar** — usa herramientas (leer archivos, ejecutar comandos, editar código).
3. **Mostrar** — te presenta los resultados y cambios realizados.

4. **Esperar** — vuelve a esperar tu siguiente instrucción.

Consejo

Consejo: Sé específico en tus instrucciones. En lugar de ”arregla el código”, di ”el test en auth.test.ts falla con un error 401, revísalo y corrígelo”.

3.4 Modos de operación

3.4.1 Modo normal (por defecto)

Claude Code te pide confirmación antes de editar archivos o ejecutar comandos importantes. Ideal para empezar y mantener control.

3.4.2 Modo auto

Claude completa tareas de forma autónoma sin pedir confirmación en cada paso. Útil cuando confías en la tarea y quieres velocidad:

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

Atención

Usa el modo auto con precaución. Claude puede modificar archivos, instalar paquetes o ejecutar comandos sin pausa. Úsalo en entornos controlados o con proyectos que tengan git para revertir cambios.

3.4.3 Modo plan

Escribe `/plan` antes de tu petición para que Claude te muestre el plan antes de ejecutar nada. Ideal para tareas complejas:

```
> /plan refactoriza el módulo de autenticación para usar JWT
```

3.5 Ejemplos de primeras tareas

3.5.1 Entender un proyecto existente

```
> Explícame la estructura de este proyecto y qué hace cada carpeta
```

```
> ¿Qué hace la función processPayment en src/payments.ts?
```

3.5.2 Crear un archivo nuevo

```
> Crea un componente React llamado Button con variantes primary y secondary en
↪  TypeScript
```

3.5.3 Arreglar un error

```
> Tengo este error en la consola: [pegar el error] - ayúdame a resolverlo
```

3.5.4 Ejecutar tests

```
> Ejecuta los tests y si hay fallos, corrígelos
```

3.6 Salir de Claude Code

Puedes salir con `Ctrl`+`C`, `Ctrl`+`D`, o escribiendo:

```
> /exit
```

3.7 Iniciar con un prompt directo

Si sabes exactamente qué quieres, puedes pasarlo directamente al comando sin entrar al modo interactivo:

```
claude "explica el archivo package.json"
claude "añade tipos TypeScript al archivo utils.js"
```

3.8 Modo ”print” (salida directa)

Para usar Claude Code en scripts o pipelines:

```
claude -p "resume los cambios en este diff" < cambios.diff
```

La bandera `-p` hace que Claude responda una vez y salga, ideal para automatizaciones.

Chapter 4

CLI, app de escritorio, web y móvil

Claude Code no vive solo en la terminal. Puedes usarlo en varias superficies y todas comparten el MISMO motor: tus archivos CLAUDE.md, tus ajustes y tus servidores MCP funcionan igual en todas. Elige la que mejor encaje contigo (o combínalas).

Nota

Dato clave: como todas las superficies usan el mismo motor, puedes empezar una tarea en un sitio y continuarla en otro.

4.1 Las superficies de Claude Code

Superficie	Qué es	Ideal para
Terminal (CLI)	La versión completa, en tu terminal.	Máximo control, scripts, automatización y modo headless.
App de escritorio	App independiente para Mac y Windows, con interfaz visual.	Revisar diffs visualmente, varias sesiones a la vez, tareas programadas y lanzar sesiones en la nube.
Web (claude.ai/code)	En el navegador, sin instalar nada.	Tareas largas en la nube, trabajar en repos que no tienes en local y varias tareas en paralelo.
VS Code / Cursor	Extensión para tu editor.	Diffs en línea, @-menciones, revisión de plan e historial dentro del editor.
JetBrains	Plugin para IntelliJ, PyCharm, WebStorm, etc. (requiere la CLI).	Lo mismo dentro de los IDE de JetBrains.

4.2 Terminal (CLI)

La forma original y más potente. Instalación recomendada (instalador nativo):

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

```
# Windows (PowerShell)
irm https://claude.ai/install.ps1 | iex

# Con Homebrew (Mac)
brew install --cask claude-code
```

También puedes instalarlo con npm (`npminstall-g@anthropic-ai/claude-code`). Tras instalar, ve a tu proyecto y ejecuta `claude`. Más detalle en Instalación (`https://claude-rho-snowy.vercel.app/instalacion`). Es la mejor opción para automatización, scripts y flujos headless (ver Perfiles técnicos (`https://claude-rho-snowy.vercel.app/equipos`)).

4.3 App de escritorio (Mac y Windows)

App independiente con interfaz gráfica. Permite revisar diffs visualmente, ejecutar varias sesiones en paralelo lado a lado, programar tareas recurrentes y lanzar sesiones en la nube. Ideal si prefieres una interfaz visual a la terminal. Requiere una suscripción de pago. Se descarga desde la web oficial; tras instalar, inicia sesión y pulsa la pestaña "Code".

Consejo

Consejo: si la terminal te intimida, la app de escritorio es la forma más cómoda de empezar con la misma potencia.

4.4 Web (claude.ai/code)

Ejecuta Claude Code en el navegador, sin instalar nada en tu ordenador. Perfecta para lanzar tareas largas y volver cuando terminen, trabajar en repositorios que no tienes en local, o correr varias tareas a la vez en la nube. Disponible en navegadores de escritorio y en la app de Claude para iOS. Empieza en claude.ai/code (`https://claude.ai/code`).

4.5 Extensiones de IDE (VS Code, JetBrains)

Si trabajas en VS Code (o Cursor) o en un IDE de JetBrains, hay extensión/plugin oficial que integra Claude Code en el editor con diffs en línea, @-menciones y revisión de plan. La de JetBrains requiere tener la CLI instalada. Ver Configuración (`https://claude-rho-snowy.vercel.app/configuracion`).

4.6 Controlar Claude Code desde el móvil

Sí, puedes usar y supervisar Claude Code desde el teléfono. Estas son las formas:

- **App de Claude para iOS + web:** lanza una tarea larga desde claude.ai/code o la app de iOS y revísala o continúala desde el móvil.
- **Remote Control:** continúa una sesión que tienes en tu ordenador desde el móvil u otro dispositivo.
- **Dispatch:** encarga una tarea desde el móvil; se crea una sesión de escritorio que abres luego en tu ordenador.
- **Teleport:** empieza una tarea en la web o en iOS y tráela a tu terminal con el comando `claude--teleport` (requiere suscripción de claude.ai).

- **Channels:** envía tareas a una sesión desde Telegram, Discord, iMessage o tus propios webhooks.
- **Slack:** menciona a @Claude con un reporte de bug y te devuelve un pull request.

Atención

Importante: el móvil brilla para LANZAR, SUPERVISAR y APROBAR tareas, no para escribir código intensivo. Lo ideal: encargar desde el móvil y revisar a fondo en el ordenador.

4.7 ¿Cuál elijo?

- ¿Te manejas en terminal y quieres todo el poder? Terminal (CLI).
- ¿Prefieres una interfaz visual sin terminal? App de escritorio.
- ¿No quieres instalar nada o trabajar desde varios sitios? Web.
- ¿Ya vives en tu editor? Extensión de VS Code o JetBrains.
- ¿Quieres lanzar o supervisar sobre la marcha? Móvil (iOS/web) + Remote Control.

Nota

No tienes que elegir solo una. Como todas comparten el mismo motor (CLAUDE.md, ajustes, MCP), mucha gente usa la CLI o la app en el ordenador y el móvil para supervisar.

Chapter 5

Recetas prácticas

Casos de uso reales para el día a día. Cada receta incluye el prompt exacto que puedes copiar y pegar en Claude Code. Pensadas para personas que están empezando en programación.

Consejo

Cómo usar esta página: haz clic en Copiar en cualquier prompt, pégalo en tu terminal con Claude Code abierto, y ajusta los detalles entre corchetes [ ] a tu caso.

En esta página

Aprender a programar Crear tu primer proyecto Entender código que no escribiste Resolver errores Mejorar tu código Git y control de versiones Trabajar con datos y archivos Automatizar tareas repetitivas Comandos de terminal sin miedo Documentar y explicar

5.1 Aprender a programar

Claude Code es un profesor particular dentro de tu terminal. No solo escribe código: te explica el porqué de cada cosa a tu nivel.

5.1.1 Pedir una explicación a tu nivel

Prompt:

```
Explícame qué es una función en programación como si nunca hubiera programado.
↪ Usa un ejemplo de la vida real y luego enséñame cómo se escribe en Python.
```

5.1.2 Aprender haciendo un mini-proyecto guiado

Prompt:

```
Quiero aprender Python desde cero. Guíame paso a paso para crear una calculadora
↪ sencilla en la terminal. Explícame cada línea antes de escribirla y espera a
↪ que yo entienda antes de continuar.
```

5.1.3 Entender un concepto que se te resiste

Prompt:

```
No entiendo qué es un "bucle for". Explícamelo con 3 ejemplos sencillos, del más
↪ fácil al más difícil, y dime para qué se usa en la vida real.
```

5.1.4 Practicar con ejercicios

Prompt:

```
Ponme 5 ejercicios de Python para principiantes sobre listas, ordenados de fácil
↪ a difícil. No me des las soluciones todavía: dámelas solo cuando yo te lo
↪ pida después de intentarlo.
```

5.2 Crear tu primer proyecto

Lo mejor de Claude Code es que puede montar un proyecto completo y funcional mientras tú aprendes viéndolo trabajar.

5.2.1 Una página web personal

Prompt:

```
Crea una página web personal sencilla con HTML y CSS en una carpeta nueva
↪ llamada "mi-web". Que tenga: mi nombre, una foto, una breve biografía y mis
↪ enlaces a redes sociales. Que se vea moderna y funcione en el móvil. Cuando
↪ termines, dime cómo abrirla en el navegador.
```

5.2.2 Una pequeña app de tareas (to-do list)

Prompt:

```
Crea una aplicación de lista de tareas que funcione en el navegador, solo con
↪ HTML, CSS y JavaScript (sin librerías). Quiero poder añadir tareas,
↪ marcarlas como completadas y borrarlas. Que se guarden aunque cierre la
↪ página. Explícame cómo probarla.
```

5.2.3 Un script útil para tu día a día

Prompt:

```
Necesito un script en Python que renombre todas las fotos de una carpeta
↪ poniéndoles la fecha en que se hicieron al principio del nombre. Pregúntame
↪ la ruta de la carpeta antes de empezar y avísame antes de cambiar nada.
```

Atención

Antes de tocar tus archivos: cuando un script modifica archivos importantes (fotos, documentos), pídele a Claude que primero haga una prueba sin cambiar nada y te enseñe qué haría. Añade: ”primero muéstrame qué cambiarías sin hacerlo de verdad”.

5.3 Entender código que no escribiste

Cuando heredas un proyecto o sigues un tutorial, Claude Code te traduce el código a lenguaje humano.

5.3.1 Entender un proyecto entero

Prompt:

```
Acabo de descargar este proyecto y no sé por dónde empezar. Explícame: qué hace
↪ la aplicación, qué tecnologías usa, qué hace cada carpeta principal y cuál
↪ es el archivo más importante por el que debería empezar a leer.
```

5.3.2 Explicar un archivo o función concreta

Prompt:

```
Explícame línea por línea, en español sencillo, qué hace el archivo
↪ [src/login.js]. No asumas que sé mucho.
```

5.3.3 Traducir jerga técnica

Prompt:

```
En este proyecto veo palabras como "API", "endpoint" y "middleware". Explícame
↪ cada una con una analogía sencilla y señálame dónde aparecen en el código.
```

5.4 Resolver errores

Los errores son la parte más frustrante al empezar. Claude Code los lee, te explica qué significan y los arregla.

5.4.1 Entender y arreglar un error

Prompt:

```
Me sale este error y no entiendo qué significa:

[pega aquí el error completo]

Explícame en palabras sencillas qué está pasando, por qué ocurre y cómo
↪ arreglarlo.
```

5.4.2 Cuando algo ”no funciona” pero no hay error

Prompt:

```
Mi página debería mostrar una lista de productos pero aparece en blanco. No sale
↪ ningún error. Investiga qué está pasando, dime dónde está el problema y
↪ arréglalo.
```

5.4.3 Arreglar los tests que fallan

Prompt:

```
Ejecuta los tests del proyecto. Si alguno falla, explícame por qué falla y
↪ arréglalo, pero enséñame el cambio antes de aplicarlo.
```

Consejo

Truco: copia todo el mensaje de error, incluida la parte larga que parece ”ruido”. Esa parte (el stack trace) es justo lo que Claude necesita para encontrar el problema rápido.

5.5 Mejorar tu código

5.5.1 Pedir una revisión como si fuera un mentor

Prompt:

```
Revisa el archivo [mi_codigo.py] como si fueras un programador con experiencia
↪ ayudando a un principiante. Dime qué está bien, qué se puede mejorar y por
↪ qué. Sé amable pero honesto.
```

5.5.2 Hacer el código más legible

Prompt:

```
Este código funciona pero es un lío y no lo entiendo cuando vuelvo a él días
↪ después. Reescríbelo para que sea más claro y fácil de leer, con buenos
↪ nombres de variables y comentarios donde haga falta. No cambies lo que hace.
```

5.5.3 Añadir comprobaciones de seguridad

Prompt:

```
En este formulario el usuario puede escribir cualquier cosa. Añade
↪ comprobaciones para que no se rompa si dejan campos vacíos o escriben datos
↪ raros. Explícame qué problemas estabas previniendo.
```

5.6 Git y control de versiones

Git asusta al principio. Deja que Claude Code lo maneje mientras tú aprendes los conceptos.

5.6.1 Empezar a usar git sin saber git

Prompt:

```
Quiero empezar a guardar el historial de cambios de este proyecto con git, pero
↪ nunca lo he usado. Configúralo, haz el primer guardado (commit) y explícame
↪ con palabras sencillas qué acabas de hacer y por qué sirve.
```

5.6.2 Guardar tu trabajo con un buen mensaje

Prompt:

```
Guarda los cambios que he hecho hoy con git. Escribe un mensaje de commit claro
↪ que explique qué cambié. Antes de guardar, enséñame qué archivos van a
↪ entrar.
```

5.6.3 "He liado algo, quiero volver atrás"

Prompt:

```
Creo que rompí algo con mis últimos cambios. Enséñame qué he cambiado desde el
↪ último guardado y ayúdame a volver a como estaba si hace falta. Explícame
↪ las opciones antes de hacer nada.
```

5.6.4 Subir el proyecto a GitHub

Prompt:

```
Quiero subir este proyecto a GitHub por primera vez para tener una copia online.
↪ Guíame paso a paso: qué necesito, qué tengo que hacer en la web de GitHub y
↪ qué comandos ejecutar.
```

5.7 Trabajar con datos y archivos

5.7.1 Analizar un archivo Excel o CSV

Prompt:

```
Tengo un archivo [ventas.csv] con datos de ventas. Dime cuántas filas tiene, qué
↪ columnas hay, y hazme un resumen: total de ventas, mes con más ventas y el
↪ producto más vendido.
```

5.7.2 Convertir entre formatos

Prompt:

```
Tengo un archivo [datos.json] y necesito los mismos datos en formato Excel (CSV)
↪ para abrirlos con el programa de hojas de cálculo. Conviértelo y guárdalo.
```

5.7.3 Limpiar datos desordenados

Prompt:

```
En este archivo [contactos.csv] los números de teléfono están escritos de mil
↪ maneras distintas. Unificalos todos al mismo formato y quita las filas
↪ duplicadas. Enséñame un ejemplo de antes y después.
```

5.7.4 Generar un gráfico

Prompt:

```
Con los datos del archivo [gastos.csv], crea un gráfico de barras de gastos por
↪ categoría y guárdalo como imagen. Usa Python.
```

5.8 Automatizar tareas repetitivas

Si haces algo aburrido y repetitivo en el ordenador, probablemente Claude Code pueda automatizarlo.

5.8.1 Organizar la carpeta de descargas

Prompt:

```
Mi carpeta de Descargas es un caos. Crea un script que ordene los archivos en
↪ subcarpetas según su tipo (imágenes, PDFs, vídeos, instaladores, etc.).
↪ Antes de moverlos, enséñame el plan de qué iría a cada carpeta.
```

5.8.2 Renombrar muchos archivos a la vez

Prompt:

```
Tengo 200 archivos llamados "IMG_0001", "IMG_0002"... Quiero renombrarlos a
↪ "vacaciones-2026-001", "vacaciones-2026-002", etc. Hazlo con un script y
↪ muéstrame primero cómo quedarían los primeros 5 nombres.
```

5.8.3 Buscar texto en muchos archivos

Prompt:

```
Busca en qué archivos de este proyecto aparece la palabra "contraseña" o
↪ "password". Quiero asegurarme de no haber dejado ningún dato sensible
↪ escrito por error.
```

5.9 Comandos de terminal sin miedo

La terminal intimida. Claude Code la usa por ti y te enseña los comandos poco a poco.

5.9.1 "¿Cómo se hace esto en la terminal?"

Prompt:

```
Quiero ver cuánto espacio ocupa cada carpeta dentro de este directorio para
↪ saber qué está llenando mi disco. Hazlo y explícame el comando que usaste
↪ por si quiero repetirlo.
```

5.9.2 Instalar herramientas

Prompt:

```
Necesito instalar [Python / Node.js / git] en mi ordenador (uso macOS). Guíame
↪ paso a paso, comprueba si ya lo tengo instalado primero y dime cómo
↪ verificar que funciona.
```

5.9.3 Antes de ejecutar un comando que viste en internet

Prompt:

```
Encontré este comando en un tutorial y me da miedo ejecutarlo sin saber qué hace:

[pega el comando]

Explícame exactamente qué hace cada parte y dime si es seguro ejecutarlo.
```

Atención

Regla de oro: nunca ejecutes un comando de internet que no entiendas. Pregúntale primero a Claude Code qué hace. Especialmente si llevasudo, rm o curl ... | bash.

5.10 Documentar y explicar

5.10.1 Crear un README para tu proyecto

Prompt:

```
Crea un archivo README.md para este proyecto que explique: qué hace, cómo
↪ instalarlo, cómo usarlo y qué tecnologías usa. Escríbelo para que alguien
↪ que llega nuevo lo entienda.
```

5.10.2 Comentar tu propio código

Prompt:

```
Añade comentarios al archivo [mi_script.py] explicando qué hace cada parte,
↪ pensando en mí dentro de 6 meses cuando ya no me acuerde de nada. No cambies
↪ el código, solo añade explicaciones.
```

5.10.3 Preparar una explicación para presentar tu trabajo

Prompt:

```
Tengo que explicar este proyecto en clase. Hazme un resumen sencillo de qué hace
↪ y cómo funciona por dentro, con un guion de 5 puntos que pueda usar para
↪ presentarlo.
```

Chapter 6

Proyectos guiados

La mejor forma de aprender es construyendo algo real. Aquí tienes tres proyectos completos, de principio a fin, con los prompts exactos en orden. Solo necesitas Claude Code instalado y ganas.

Consejo

Cómo seguir un proyecto: abre una terminal, crea una carpeta para el proyecto, entra en ella y ejecuta claude. Luego ve copiando los prompts en orden. Tómate tu tiempo para leer lo que Claude te explica en cada paso.

Web personalPrincipiante · 30 minApp de tareasPrincipiante + · 45 minScript en Python-Intermedio · 30 min

6.1 Proyecto 1: Tu web personal

Qué construirás: una página web personal con tu nombre, foto, biografía y enlaces, lista para enseñar al mundo.

Qué aprenderás: qué son HTML y CSS, cómo se ve un proyecto web por dentro y cómo publicarlo gratis en internet.

1

Crea la base de la web

Pídele a Claude que monte la estructura inicial:
Prompt:

```
Soy principiante total. Crea una página web personal sencilla con HTML y CSS en
↪ esta carpeta. Que tenga: mi nombre como título grande, un hueco para una
↪ foto, un párrafo de biografía y una fila de enlaces (LinkedIn, GitHub,
↪ email). Que se vea moderna, con colores agradables, y que funcione bien en
↪ el móvil. Explícame qué archivos creas y para qué sirve cada uno.
```

2

Ábrela en el navegador

Mira el resultado por primera vez:
Prompt:

```
Dime exactamente cómo abrir esta web en mi navegador para verla.
```

3

Personalízala a tu gusto

Ahora hazla tuya:
Prompt:

```
Cambia el texto por mi información real: me llamo [tu nombre], soy [a qué te
↪ dedicas], y mi biografía es: [escribe 2 frases]. Pon los enlaces a mis
↪ redes: [pega tus enlaces]. Cambia los colores a tonos [azules / verdes / lo
↪ que quieras].
```

4

Añade un detalle bonito

Aprende pidiendo mejoras visuales:
Prompt:

```
Añade una animación suave para que los elementos aparezcan al cargar la página,
↪ y que los enlaces cambien de color cuando paso el ratón por encima.
↪ Explícame por encima cómo lo has hecho.
```

5

Publícala gratis en internet

Comparte tu web con el mundo:
Prompt:

```
Quiero publicar esta web gratis en internet para tener un enlace que pueda
↪ compartir. Recomiéndame la forma más fácil para un principiante y guíame
↪ paso a paso.
```

6.2 Proyecto 2: App de lista de tareas

Qué construirás: una app donde añadir tareas, marcarlas como hechas y borrarlas, que recuerda tus tareas aunque cierres la página.

Qué aprenderás: qué es JavaScript, cómo una web ”reacciona” a lo que haces y cómo se guardan datos en el navegador.

1

Crea la app básica

Prompt:

```
Soy principiante. Crea una aplicación de lista de tareas que funcione en el
↪ navegador, usando solo HTML, CSS y JavaScript (sin librerías externas). De
↪ momento quiero poder: escribir una tarea en una caja de texto, pulsar un
↪ botón "Añadir" y que aparezca en una lista debajo. Que se vea limpia y
↪ moderna. Explícame para qué sirve cada archivo.
```

2

Marcar tareas como completadas

Prompt:

```
Ahora quiero poder marcar una tarea como completada al hacer clic en ella: que
↪ se ponga tachada y en gris. Y que pueda desmarcarla volviendo a hacer clic.
```

3

Borrar tareas

Prompt:

```
Añade un botón de papelera al lado de cada tarea para poder borrarla. Pídeme
↪ confirmación antes de borrar.
```

4

Que recuerde las tareas

El momento "magia": persistencia de datos.
Prompt:

```
Haz que las tareas se guarden, de forma que si cierro la página y vuelvo a
↪ abrirla, mis tareas sigan ahí. Explícame de forma sencilla cómo lo consigues
↪ (qué es "localStorage").
```

5

Pulir y entender

Prompt:

```
La app ya funciona. Repásala conmigo: explícame en lenguaje sencillo qué hace
↪ cada parte del JavaScript, como si me estuvieras enseñando. Y dime una
↪ mejora que podría intentar yo solo como ejercicio.
```

6.3 Proyecto 3: Un script útil en Python

Qué construirás: un programa que organiza automáticamente una carpeta desordenada (como Descargas) metiendo cada archivo en su sitio.

Qué aprenderás: qué es Python, cómo un programa trabaja con tus archivos y cómo probar algo de forma segura antes de aplicarlo de verdad.

1

Comprueba que tienes Python

Prompt:

```
Comprueba si tengo Python instalado en mi ordenador (uso macOS). Si no lo tengo,
↪ guíame para instalarlo. Cuando esté, dime cómo verificar que funciona.
```

2

Crea el script en modo seguro

Pide que primero solo simule, sin mover nada:
Prompt:

```
Crea un script en Python que organice los archivos de una carpeta en subcarpetas
↪ según su tipo (Imágenes, Documentos, Vídeos, Comprimidos, Otros). MUY
↪ IMPORTANTE: de momento que solo MUESTRE qué movería, sin mover nada de
↪ verdad. Así puedo revisarlo antes. Explícame cómo ejecutarlo.
```

3

Pruébalo con una carpeta de ejemplo

Prompt:

```
Crea una carpeta de prueba con varios archivos falsos de distintos tipos y
↪ ejecuta el script sobre ella para ver qué haría. Enséñame el resultado.
```

4

Actívalo de verdad

Prompt:

```
Perfecto, funciona como esperaba. Ahora añade una opción para que mueva los
↪ archivos de verdad, pero que me pida confirmación antes de empezar y que me
↪ diga cuántos archivos movió al terminar.
```

5

Hazlo reutilizable

Prompt:

```
Haz que el script me pregunte qué carpeta quiero organizar al ejecutarlo, en vez
↪ de tenerlo escrito fijo. Y crea un archivo README.md que explique cómo
↪ usarlo, por si lo retomo dentro de meses.
```

Nota

¿Y ahora qué? Cuando termines estos proyectos, intenta uno tuyo desde cero. Describe a Claude Code qué quieres construir y pídele que te guíe paso a paso, igual que aquí. Echa un vistazo a las recetas prácticas para más ideas.

Chapter 7

Cómo escribir buenos prompts

La calidad de lo que te da Claude Code depende mucho de cómo se lo pidas. No hace falta saber ”hablar técnico”: solo ser claro. Aquí tienes los 7 principios y ejemplos reales antes/después.

7.1 1. Sé específico, no genérico

”Arregla esto” obliga a Claude a adivinar. Cuanto más contexto des, menos adivina y mejor acierta.

Poco claroArregla mi código

Mucho mejorEl botón de ’Enviar’ en login.html no hace nada al pulsarlo. Debería mostrar un mensaje de error si el campo email está vacío. Investiga por qué no funciona y arréglalo.

7.2 2. Di tu nivel y pide explicaciones

Claude se adapta a ti. Si le dices que estás empezando, te explicará las cosas en lugar de soltar código sin más.

Poco claroHaz una API REST con autenticación JWT

Mucho mejorEstoy empezando a programar. Ayúdame a crear una API sencilla con login. Explícame cada paso con palabras normales antes de escribir el código, y dime qué es ’JWT’ cuando lo uses.

7.3 3. Pide ver antes de actuar

Para tareas que cambian archivos o pueden tener consecuencias, pide el plan primero. Así aprendes y evitas sustos.

Prompt: Plantilla segura

```
Antes de hacer ningún cambio, explícame tu plan paso a paso y espera mi
↪  aprobación. Solo entonces empieza.
```

7.4 4. Da contexto: pega errores, datos y ejemplos

No describas el error con tus palabras: **pégalo entero**. No expliques cómo son tus datos: enséñale el archivo.

Poco claroMe da un error al ejecutar el programa

Mucho mejorAl ejecutar ”python app.py” me sale este error: Traceback (most recent call last): File ”app.py”, line 12, in <module> print(precio * cantidad) TypeError: can’t multiply sequence by non-int ¿Qué significa y cómo lo arreglo?

7.5 5. Divide las tareas grandes

En lugar de pedir todo de golpe, ve por partes. Claude trabaja mejor y tú entiendes lo que va pasando.

Poco claroHazme una tienda online completa con carrito, pagos, usuarios y panel de administración

Mucho mejorVamos a hacer una tienda online paso a paso. Empecemos solo por la página que muestra la lista de productos. Cuando funcione, seguimos con el carrito.

7.6 6. Describe el resultado que quieres, no la solución técnica

No tienes que saber *cómo* se hace. Describe *qué* quieres conseguir y deja que Claude proponga el cómo.

Poco claroUsa un useEffect con un debounce y memoización

Mucho mejorQuiero que cuando el usuario escriba en el buscador, los resultados se filtren solos sin tener que pulsar un botón, pero que no vaya lento aunque escriba rápido.

7.7 7. Pide que te corrija y te enseñe

Claude Code no es solo para que haga el trabajo: úsalo para mejorar tú.
Prompt: Modo aprendizaje

```
Cuando termines, dime: ¿qué he hecho yo mal o de forma poco eficiente? ¿Qué
↪ debería aprender para hacer esto mejor la próxima vez por mi cuenta?
```

7.8 Plantillas listas para usar

Copia, rellena los corchetes y pega:

7.8.1 Para crear algo nuevo

Prompt:

```
Quiero crear [qué quieres]. Lo usaré para [para qué sirve]. Soy [tu nivel:
↪ principiante/intermedio]. Usa [tecnología, o "lo que recomiendes"].
↪ Explícame los pasos importantes mientras lo haces.
```

7.8.2 Para arreglar algo

Prompt:

```
Tengo este problema: [qué pasa]. Esperaba que pasara: [qué debería pasar]. Lo
↪ que veo: [qué ves, pega errores]. Investiga la causa, arréglalo y explícame
↪ qué estaba mal.
```

7.8.3 Para entender algo

Prompt:

```
Explícame [el concepto o el archivo] como si tuviera poca experiencia. Usa una
↪ analogía sencilla y un ejemplo pequeño. Luego dime para qué se usa en la
↪ práctica.
```

7.8.4 Para mejorar lo que ya tienes

Prompt:

```
Revisa [archivo o carpeta]. Dime qué se puede mejorar en cuanto a claridad,
↪ errores potenciales y buenas prácticas. Aplica las mejoras importantes y
↪ explícame por qué.
```

Consejo

El mejor consejo de todos: habla con Claude Code como hablarías con un compañero de trabajo paciente. No necesitas comandos mágicos ni vocabulario técnico. La claridad gana siempre.

7.9 Errores comunes al empezar

- **Aceptar cambios sin leerlos.** Claude muestra un diff antes de editar. Léelo: así aprendes y detectas si algo no es lo que querías.
- **No dar contexto del proyecto.** Si tienes un `CLAUDE.md` (ver Configuración (https://claude-rho-snowy.vercel.app/configuracion)), Claude entiende mucho mejor tu proyecto desde el primer mensaje.
- **Rendirse al primer intento.** Si la respuesta no es lo que querías, no empieces de cero: dile *"casi, pero quería que además..."*y refina.
- **Pedir demasiado de una vez.** Tareas enormes en un solo prompt salen peor. Divide y vencerás.

Chapter 8

Glosario

¿Te has perdido con alguna palabra técnica? Aquí tienes los términos que más aparecen al usar Claude Code, explicados con palabras normales y analogías de la vida real.

Consejo

Recuerda: si alguna vez no entiendes una palabra mientras usas Claude Code, ¡pregúntale! Escribe ”explícame qué es [palabra] de forma sencilla” y te lo aclarará al instante.

8.0.1 Terminal (o consola)

Una ventana donde escribes comandos de texto para darle órdenes al ordenador, en lugar de hacer clic con el ratón.

Como enviarle instrucciones escritas a tu ordenador en vez de señalar botones.

8.0.2 CLI

Command Line Interface (interfaz de línea de comandos). Un programa que usas escribiendo en la terminal. Claude Code es una CLI.

8.0.3 Comando

Una orden que escribes en la terminal para que el ordenador haga algo. Por ejemplo, ’ls’ lista los archivos de una carpeta.

8.0.4 Directorio

Es lo mismo que una carpeta. Los programadores suelen decir ’directorio’.

8.0.5 Repositorio (repo)

Una carpeta de proyecto cuyo historial de cambios se guarda con git. Puede estar en tu ordenador y/o en internet (GitHub).

Como una carpeta con ’máquina del tiempo’ incorporada.

8.0.6 git

Una herramienta que guarda el historial de todos los cambios de tu proyecto, para que puedas volver atrás o ver qué cambió y cuándo.

Como el historial de versiones de un documento, pero para código.

8.0.7 Commit

Un 'punto de guardado' en git. Cada commit captura cómo estaba tu proyecto en ese momento, con un mensaje que explica qué cambiaste.

Como una foto del estado de tu proyecto con una etiqueta describiéndola.

8.0.8 GitHub

Una web donde puedes guardar tus repositorios en internet, compartirlos y colaborar con otras personas.

8.0.9 Branch (rama)

Una línea de trabajo paralela en git. Te permite hacer cambios sin tocar la versión principal hasta que estés seguro.

Como hacer una copia para experimentar sin estropear el original.

8.0.10 Pull Request (PR)

Una propuesta para añadir tus cambios al proyecto principal, para que alguien los revise antes de aceptarlos.

8.0.11 API

Application Programming Interface. Una forma de que dos programas hablen entre sí. Por ejemplo, una web que consulta el tiempo usa la API de un servicio meteorológico.

Como el camarero de un restaurante: tú pides, él trae lo que la cocina prepara, sin que entres a la cocina.

8.0.12 API key

Una contraseña secreta que identifica quién usa un servicio. Claude Code necesita tu API key de Anthropic para funcionar.

Como tu carné personal para entrar a un servicio.

8.0.13 Frontend

La parte de una aplicación que ve y usa el usuario: botones, textos, colores, formularios.

El escaparate y la sala de una tienda.

8.0.14 Backend

La parte que no se ve: la lógica, la base de datos, los cálculos. Funciona 'por detrás'.

El almacén y la trastienda donde se gestiona todo.

8.0.15 Base de datos

Un sitio organizado donde se guardan los datos de una aplicación (usuarios, productos, mensajes...).

Como un archivador gigante, ordenado y consultable al instante.

8.0.16 Framework / Librería

Código ya hecho por otras personas que reutilizas para no empezar de cero. Por ejemplo, React es una librería para construir interfaces.

Como usar muebles de IKEA en vez de fabricar cada tornillo.

8.0.17 Dependencia

Una librería externa que tu proyecto necesita para funcionar. Se instalan normalmente con npm o pip.

8.0.18 npm

El gestor de paquetes de Node.js. Sirve para instalar librerías de JavaScript con un comando.

8.0.19 Paquete

Una librería empaquetada y lista para instalar. 'Instalar un paquete' = añadir código de otra persona a tu proyecto.

8.0.20 Función

Un bloque de código con nombre que hace una tarea concreta y puedes reutilizar las veces que quieras.

Como una receta: la escribes una vez y la usas cuando quieras.

8.0.21 Variable

Un nombre que guarda un valor (un número, un texto...) para usarlo después.

Como una caja etiquetada donde guardas algo.

8.0.22 Bug

Un error o fallo en el código que hace que no funcione como debería.

8.0.23 Debug (depurar)

El proceso de encontrar y arreglar bugs.

8.0.24 Stack trace / Traceback

El texto largo que aparece cuando algo falla. Muestra dónde y por qué se rompió el programa. Es muy útil: cópialo entero al pedir ayuda.

8.0.25 Deploy (desplegar)

Publicar tu proyecto en internet para que otras personas puedan usarlo. Vercel, por ejemplo, despliega webs.

8.0.26 Servidor

Un ordenador (normalmente en internet) que ejecuta tu aplicación y responde a las peticiones de los usuarios.

8.0.27 localhost

Tu propio ordenador actuando como servidor, para probar tu proyecto antes de publicarlo. Suele verse como 'localhost:3000'.

8.0.28 JSON

Un formato de texto para guardar y compartir datos de forma organizada. Lo usan casi todas las aplicaciones para comunicarse.

8.0.29 Entorno (environment)

El conjunto de programas y configuraciones donde se ejecuta tu código. Tu ordenador es un entorno; un servidor es otro.

8.0.30 Variable de entorno

Un valor de configuración que vive fuera del código (como una API key), para no escribir datos secretos directamente en los archivos.

8.0.31 Hook

En Claude Code, un script que se ejecuta solo cuando ocurre algo (antes o después de una acción). Sirve para automatizar.

8.0.32 MCP

Model Context Protocol. La forma en que Claude Code se conecta a herramientas externas como bases de datos o el navegador.

Como un enchufe universal para conectar herramientas a la IA.

Chapter 9

Claude Code para pymes y oficina

Aunque Claude Code es una herramienta para programar, su verdadero superpoder —trabajar con TUS archivos y automatizar tu ordenador— lo hace utilísimo para autónomos y pequeñas empresas, aunque no sepas programar. Aquí tienes casos reales de oficina que te ahorran horas, con el prompt listo para copiar.

Nota

Para seguir esta página necesitas tener Claude Code instalado (Instalación) y abrirlo escribiendo claude dentro de la carpeta donde tienes tus archivos.

Atención

Seguridad: trabaja siempre con COPIAS de tus archivos importantes, y para tareas que modifican archivos añade al prompt "primero enséñame qué harías sin tocar nada".

En esta página

Hojas de cálculo y datos Documentos y plantillas Automatizar tareas repetitivas Informes y análisis Comunicación y clientes Tu presencia online

9.1 Hojas de cálculo y datos

Para sacar conclusiones rápidas de un CSV o preparar datos antes de enviarlos a otra herramienta.

9.1.1 Resumen de ventas para una reunión

Prompt:

```
Analiza este archivo [ventas.csv]: dime el total de ventas, el mejor mes, el
↪ producto más vendido y hazme un resumen en 5 puntos que pueda enseñar en una
↪ reunión.
```

9.1.2 Limpiar una lista de contactos

Prompt:

```
Limpia este archivo [contactos.csv]: unifica los formatos de teléfono, quita
↪ filas duplicadas y corrige los nombres que están en mayúsculas. Enséñame un
↪ ejemplo de antes y después.
```

9.1.3 Cruzar clientes y pedidos

Prompt:

```
Cruza estos dos archivos: [clientes.csv] y [pedidos.csv]. Dime qué clientes no
↪ han hecho ningún pedido en los últimos 6 meses para poder contactarlos.
```

9.2 Documentos y plantillas

Úsalo para convertir, resumir o generar documentos repetitivos sin copiar y pegar a mano.

9.2.1 Facturas desde una plantilla

Prompt:

```
Tengo una plantilla de factura en [plantilla.docx] y los datos de clientes en
↪ [clientes.csv]. Genera una factura por cada cliente rellenando la plantilla.
↪ Empieza con las 3 primeras para que las revise.
```

9.2.2 Resumir PDFs de una carpeta

Prompt:

```
Convierte todos los PDF de esta carpeta a texto y hazme un resumen de media
↪ página de cada uno.
```

9.2.3 Crear una plantilla de presupuesto

Prompt:

```
Créame una plantilla profesional de presupuesto en un documento, con mis datos:
↪ [nombre empresa, CIF, logo opcional], que pueda reutilizar y rellenar
↪ fácilmente.
```

9.3 Automatizar tareas repetitivas

Ideal para trabajos aburridos que se repiten cada semana: ordenar, renombrar, generar resúmenes o preparar archivos.

9.3.1 Organizar una carpeta caótica

Prompt:

```
Organiza esta carpeta metiendo cada archivo en subcarpetas por tipo (Facturas,
↪ Imágenes, Documentos, etc.). Primero enséñame el plan de qué movería, sin
↪ mover nada.
```

9.3.2 Renombrar archivos por lote

Prompt:

```
Renombra todos los archivos de esta carpeta para que empiecen por la fecha y un
↪ nombre descriptivo: [evento o proyecto]. Muéstrame cómo quedarían los 5
↪ primeros antes de hacerlo.
```

9.3.3 Programa pequeño para pedidos diarios

Prompt:

```
Crea un pequeño programa que, cuando lo ejecute, lea [pedidos.csv] y me genere
↪ un resumen de los pedidos nuevos del día. Explícamelo como si no supiera
↪ programar.
```

9.4 Informes y análisis

Pídele que convierta datos en gráficos, documentos y conclusiones claras para compartir.

9.4.1 Gráfico de gastos por categoría

Prompt:

```
Con los datos de [gastos.csv], crea un gráfico de barras de gastos por categoría
↪ y guárdalo como imagen para meterlo en mi informe.
```

9.4.2 Informe mensual de ventas

Prompt:

```
Genera un informe mensual en un documento a partir de [ventas.csv]: incluye
↪ totales, comparación con el mes anterior, y 3 conclusiones claras.
```

9.5 Comunicación y clientes

También puede ayudarte a detectar patrones en opiniones de clientes y preparar respuestas profesionales.

9.5.1 Detectar problemas repetidos en reseñas

Prompt:

```
Lee las reseñas de [resenas.csv] y dime los 5 problemas que más repiten los
↪ clientes y una propuesta de mejora para cada uno.
```

9.5.2 Plantillas de email

Prompt:

```
Redáctame 3 plantillas de email para responder a [situación: p. ej. una
↪ reclamación], en tono cercano pero profesional, de menos de 150 palabras
↪ cada una.
```

9.6 Tu presencia online sin saber programar

Si necesitas una web sencilla para validar presencia online, puede crear una primera versión lista para revisar.

9.6.1 Web de una sola página

Prompt:

```
Créame una web sencilla de una sola página para mi negocio de [tipo de negocio],
↪ con mi nombre, qué ofrezco, horarios, ubicación y un botón de contacto por
↪ WhatsApp. Que se vea moderna y funcione en el móvil. Al terminar dime cómo
↪ publicarla gratis.
```

Nota

El patrón ganador para una pyme: pon todos los archivos de la tarea en una carpeta, abre Claude Code ahí, y descríbele el resultado que quieres en lenguaje normal. Para más ejemplos, mira Recetas prácticas y Escribir buenos prompts.

Chapter 10

Para perfiles técnicos y equipos

Recetas concretas para exprimir Claude Code en proyectos serios y en equipo: revisión de código, refactors grandes, testing, CI/CD y estandarización. Cada una con el prompt o comando listo.

En esta página

Bases de código grandes Revisión de código automatizada Refactors y migraciones Testing y TDD CI/CD y modo headless Estandarizar el equipo Integraciones

10.1 Bases de código grandes

Usa `CLAUDE.md` en la raíz y por módulo para dar contexto;`/init` para generarlo; `/compact` en sesiones largas. Ajusta más detalles en Configuración (`https://claude-rho-snowy.vercel.app/configuracion`).

10.1.1 Generar contexto inicial del repositorio

Prompt:

```
Explora este repositorio y genera un CLAUDE.md con el stack, los comandos clave
↪ (build, test, lint), la estructura de carpetas y las convenciones. Mantenlo
↪ conciso.
```

10.2 Revisión de código automatizada

Usa `/code-review` y `/security-review` sobre el diff; en CI puedes automatizarlo con `claude-p`.

10.2.1 Revisar el diff local

Prompt:

```
Revisa el diff actual en busca de bugs, problemas de seguridad y deuda técnica.
↪ Ordena los hallazgos por gravedad y propón el arreglo de los críticos.
```

10.2.2 Revisión desde GitHub CLI

```
gh pr diff 123 | claude -p "haz un code review y comenta problemas por gravedad"
```

10.3 Refactors y migraciones a gran escala

Combina Plan Mode para revisar antes de tocar código con subagentes en paralelo. Profundiza en Flujos de trabajo pro (https://claude-rho-snowy.vercel.app/flujos) y Subagentes (https://claude-rho-snowy.vercel.app/subagentes).

10.3.1 Migración con plan previo

Prompt:

```
Quiero migrar todos los componentes de clase a componentes de función con hooks.
↪ Primero hazme un plan por fases y dime los riesgos. No cambies nada hasta
↪ que lo apruebe.
```

10.4 Testing y TDD

Pide pruebas concretas y haz que Claude Code las ejecute para cerrar el ciclo con evidencia.

10.4.1 Tests unitarios para un módulo

Prompt:

```
Genera tests unitarios para [módulo], cubriendo casos límite y errores.
↪ Ejecútalos y, si fallan, arregla el código o el test explicándome la causa.
```

10.4.2 Trabajar en TDD

Prompt:

```
Trabajemos en TDD: escribe primero un test que falle para [funcionalidad], y
↪ luego implementa el código mínimo para que pase.
```

10.5 CI/CD y modo headless

Usa `claude-p` para scripts y `--output-formatjson` cuando necesites parsear la salida desde otro proceso.

10.5.1 Ejemplo en GitHub Actions

```
- name: Claude review
  run: claude -p "revisa los cambios del PR y resume riesgos" --output-format
  ↪ json > review.json
  env:
    ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
```

10.6 Estandarizar el equipo

Versiona la carpeta `.claude/` en el repo con settings, skills y agents para que todo el equipo comparta configuración, permisos y flujos. Empaqueta lo común como plugin interno. Mira Skills (https://claude-rho-snowy.vercel.app/skills) y Plugins (https://claude-rho-snowy.vercel.app/plugins).

10.6.1 Skill interna para antes del PR

Prompt:

```
Crea en .claude/ una skill de 'revisión previa a PR' que compruebe lint, tests y
↪ que no haya console.log ni secretos. Que el equipo pueda invocarla con
↪ /pre-pr.
```

10.7 Integraciones

Conecta MCP a tu base de datos u observabilidad; usa hooks para formatear al guardar. Sigue con Servidores MCP (https://claude-rho-snowy.vercel.app/mcp) y Hooks (https://claude-rho-snowy.vercel.app/hooks).

10.7.1 Consultar datos mediante MCP

Prompt:

```
Conéctate a la base de datos vía MCP y dime el esquema de la tabla [pedidos],
↪ luego escríbeme una consulta para ver los 10 pedidos de mayor importe del
↪ último mes.
```

Consejo

El cambio de mentalidad para equipos: pasar de escribir cada línea a dirigir y revisar; estandariza con .claude/ versionado para que el equipo trabaje igual.

Chapter 11

Skills

Las Skills son la forma más potente de enseñarle a Claude Code a hacer tareas concretas *a tu manera*: revisiones, despliegues, debugging, flujos repetitivos... Las defines una vez y Claude las usa cuando hacen falta.

11.1 ¿Qué es una Skill?

Una Skill es un conjunto reutilizable de instrucciones y procedimientos (checklists, flujos de varios pasos, comportamientos especializados) que amplían lo que Claude sabe hacer. Siguen el estándar abierto **Agent Skills** más las extensiones de Claude Code.

Claude carga una Skill **automáticamente** cuando es relevante (según su descripción), o tú la invocas **manualmente** con `/nombre-skill`. Puede incluir archivos auxiliares: scripts, plantillas, documentos de referencia.

Nota

Novedad importante: los antiguos comandos personalizados (.claude/commands/) se han unificado dentro de las Skills. Los archivos .md antiguos siguen funcionando, pero las Skills son la forma recomendada porque admiten frontmatter y archivos auxiliares.

11.2 Estructura de una Skill

Una Skill es una carpeta con un archivo `SKILL.md` dentro. Ese archivo tiene dos partes: un **frontmatter** en YAML (metadatos) y el cuerpo en Markdown (las instrucciones).

```
.claude/skills/
+-- deploy/
    +-- SKILL.md          <- instrucciones + metadatos
    +-- checklist.md      <- archivo auxiliar (opcional)
    +-- scripts/
        +-- deploy.sh     <- script auxiliar (opcional)
```

11.2.1 Ejemplo de SKILL.md

```
---
name: deploy
description: Despliega la aplicación a producción. Úsala cuando el usuario pida
↪  "subir", "desplegar" o "hacer deploy".
disable-model-invocation: true
```

```
argument-hint: "[entorno]"
---

Despliega la aplicación al entorno $ARGUMENTS siguiendo estos pasos:

1. Ejecuta los tests. Si fallan, detente y avisa.
2. Comprueba que la rama es 'main' y está actualizada.
3. Ejecuta el build de producción.
4. Lanza el deploy con el script scripts/deploy.sh.
5. Verifica que el sitio responde y resume el resultado.
```

11.2.2 Campos del frontmatter

Campo	Para qué sirve
`name`	Nombre de la skill (así la invocas: /name). Obligatorio.
`description`	Cuándo usarla. Claude lee esto para decidir si la activa sola. Obligatorio.
`disable-model-invocation`	Si es true, solo se activa manualmente (ideal para acciones con efectos como deploy).
`argument-hint`	Pista de qué argumentos espera, p. ej. "[entorno]".

11.3 Dónde se guardan

- **Personal (todos tus proyectos):** `~/.claude/skills/<nombre>/SKILL.md`
- **Proyecto (recomendado, versionable):** `.claude/skills/<nombre>/SKILL.md`
- **Vía plugin:** dentro del plugin, con nombre namespaced como `/plugin:skill`
- **Monorepos:** en subdirectorios, cualificadas como `/apps/web:deploy`

Consejo

Guarda las skills del proyecto en .claude/skills/ y añádelas a git: así todo tu equipo comparte los mismos flujos de trabajo.

11.4 Cómo invocar una Skill

11.4.1 Manualmente

```
# Sin argumentos
/deploy

# Con argumentos (llegan como $ARGUMENTS)
/deploy produccion
```

11.4.2 Automáticamente

Si no pones `disable-model-invocation:true`, Claude activará la skill por su cuenta cuando tu petición encaje con su `description`. Por eso la descripción es tan importante: escríbela pensando en *cuándo* debe usarse.

11.5 Crear tu primera Skill (sin saber)

Deja que Claude Code la cree por ti:
Prompt:

```
Quiero crear una skill de Claude Code para mi proyecto que haga siempre lo mismo
↪ cuando le pida "revisar": comprobar que el código no tiene console.log
↪ olvidados, que los nombres de variables son claros y que hay tests. Crea la
↪ carpeta y el SKILL.md en .claude/skills/ y explícame cómo invocarla.
```

11.6 Skills oficiales incluidas

Claude Code trae skills listas para usar. Algunas que verás disponibles:

- `/code-review` — revisión de código del diff actual.
- `/security-review` — revisión de seguridad de tus cambios.
- `/init` — genera el `CLAUDE.md` de tu proyecto.

Existe además un plugin `skill-creator` que te ayuda a crear, iterar y evaluar tus propias skills.

11.7 Edición en vivo

Puedes editar un `SKILL.md` mientras Claude Code está abierto y los cambios tienen efecto **inmediato**, sin reiniciar. Ideal para ir afinando una skill mientras la pruebas.

Nota

Buena práctica 2026: mantén un CLAUDE.md ligero (contexto general del proyecto) y mueve los procedimientos concretos a Skills específicas. Así Claude solo carga lo que necesita en cada momento.

Chapter 12

Subagentes

Los subagentes son ”ayudantes especializados” que Claude Code puede lanzar para trabajar en paralelo: uno revisa código, otro investiga, otro escribe tests... mientras el agente principal coordina y tú solo revisas resultados.

12.1 ¿Para qué sirven?

En tareas grandes, en vez de hacerlo todo de forma lineal, Claude Code puede repartir el trabajo entre varios subagentes con roles concretos (revisor, planificador, depurador, investigador). Ventajas:

- **Paralelismo:** varios trabajando a la vez = más rápido.
- **Contexto limpio:** cada subagente tiene su propia ”memoria”, sin mezclar.
- **Especialización:** cada uno con sus instrucciones y herramientas.
- **Background:** pueden correr de fondo sin bloquearte.

Nota

Un patrón muy comentado en 2026: lanzar 7 o más subagentes en paralelo (imágenes, auditoría de seguridad, importación de datos, tests...) mientras tú solo asignas tareas y revisas los diffs. El rol del programador cambia: de escribir código a asignar y revisar.

12.2 Crear un subagente

Un subagente es un archivo Markdown con frontmatter YAML (configuración) y un cuerpo que es su *system prompt* (sus instrucciones de personalidad y rol).

```
.claude/agents/
+-- revisor.md
+-- depurador.md
+-- investigador.md
```

12.2.1 Ejemplo: un subagente revisor de código

```
---
name: revisor
```

```
description: Revisa código en busca de bugs y malas prácticas. Úsalo tras
↪  escribir o cambiar código.
tools: Read, Grep, Glob
model: sonnet
color: blue
---

Eres un revisor de código senior. Tu trabajo es leer los cambios y encontrar:
- Bugs potenciales y casos límite no cubiertos.
- Nombres poco claros y código difícil de mantener.
- Problemas de seguridad.

No edites archivos: solo informa de lo que encuentres, ordenado por gravedad.
Sé directo pero constructivo.
```

12.2.2 Campos del frontmatter

Campo	Para qué sirve
`name`	Nombre del subagente. Obligatorio.
`description`	Cuándo usarlo. El agente principal lo lee para delegar. Obligatorio.
`tools`	Lista blanca de herramientas que puede usar (p. ej. Read, Grep). Limítalas por seguridad.
`model`	Modelo a usar: sonnet, opus, fable o inherit (heredar del principal).
`permissionMode`	Modo de permisos para este subagente.
`color`	Color con el que se muestra en la interfaz.
`hooks`	Hooks específicos de este subagente.
`skills`	Skills que se precargan al lanzarlo.
`background`	Si es true, corre en segundo plano.
`effort`	Nivel de esfuerzo/razonamiento.

12.3 Dónde se guardan

- **Proyecto (recomendado):** `.claude/agents/<nombre>.md`
- **Personal:** `~/.claude/agents/<nombre>.md`
- **Vía plugin:** dentro del plugin (`agents/...`)
- **Solo para una sesión:** con el flag `--agents'{...}'` (no se guarda en disco)

12.4 Cómo invocarlos

12.4.1 Delegación natural

Simplemente pídeselo al agente principal:
Prompt:

```
Usa el subagente "revisor" para revisar los cambios que acabas de hacer y dime
↪  qué encuentra.
```

12.4.2 Mención directa

```
@"revisor (agent)" revisa src/pagos.ts
```

12.4.3 Asistente interactivo

Usa el comando `/agents` para abrir un asistente que te guía en la creación y gestión de subagentes sin escribir el YAML a mano.

12.4.4 Desde la terminal

```
# Lanzar con un subagente concreto
claude --agent revisor

# Ver, monitorizar y gestionar subagentes en paralelo
claude agents
```

12.5 Crear uno sin saber YAML

Prompt:

```
Quiero crear un subagente para mi proyecto que se dedique solo a escribir tests.
↪ Debe poder leer y editar archivos, usar el modelo sonnet, y centrarse en
↪ cubrir casos límite. Créalo en .claude/agents/ y explícame cómo lanzarlo.
```

12.6 Subagentes en paralelo (ejemplo real)

Prompt:

```
Tengo que añadir una función de notificaciones a la app. Reparte el trabajo en
↪ subagentes en paralelo: uno que investigue cómo está montado el sistema
↪ actual, otro que escriba el código, y otro que prepare los tests.
↪ Coordínalos tú y al final enséñame un resumen con los cambios para que los
↪ revise.
```

Consejo

Consejo de seguridad: limita el campo tools de cada subagente a lo mínimo. Un revisor solo necesita leer (Read, Grep); no le des permiso para editar o ejecutar comandos si no hace falta.

Chapter 13

Plugins

Los plugins son paquetes que añaden funcionalidad a Claude Code de golpe: agrupan skills, subagentes, comandos, hooks y servidores MCP. Es la forma más rápida de potenciar tu instalación con trabajo ya hecho por otros.

13.1 ¿Qué es un plugin?

Un plugin empaqueta varias extensiones en una sola unidad instalable:

- **Skills** — flujos y procedimientos especializados.
- **Subagentes** — ayudantes con roles concretos.
- **Comandos** — slash commands listos para usar.
- **Hooks** — automatizaciones de eventos.
- **Servidores MCP** — conexiones a herramientas externas.

Un **marketplace** es un repositorio (normalmente en GitHub) con un registro de plugins. Añades el marketplace una vez y luego instalas los plugins que quieras de él.

13.2 Añadir un marketplace

El marketplace oficial de Anthropic:

```
/plugin marketplace add anthropics/claude-plugins-official
```

También puedes añadir uno de la comunidad o uno privado de tu empresa:

```
/plugin marketplace add https://github.com/usuario/mi-marketplace
```

13.3 Instalar un plugin

```
# Desde un marketplace añadido
/plugin install github@claude-plugins-official

# Directamente desde un repo
/plugin install https://github.com/usuario/mi-plugin
```

O usa la interfaz interactiva: escribe `/plugin` y entra en **Discover** para explorar, ver qué incluye cada plugin y su coste estimado de contexto antes de instalar.

13.4 Gestionar plugins

```
/plugin           # Abre el panel: listar, activar, desactivar
/plugin list      # Ver plugins instalados
```

13.5 Plugins útiles que la gente instala

- Integración con **GitHub** (issues, PRs, commits).
- **Toolkits de revisión de PRs** (pr-review-toolkit).
- Flujos de **commit y despliegue**.
- Bundles de **skills científicas** (biología, química con bases de datos).
- `skill-creator` para crear y evaluar tus propias skills.

Nota

Coste de contexto: cada plugin que activas consume parte del contexto de Claude (sus skills, agentes y descripciones se cargan). Activa solo los que vayas a usar; el panel /plugin te muestra el coste estimado.

13.6 Empezar rápido con plugins

Prompt:

```
Soy nuevo con Claude Code. Añade el marketplace oficial de Anthropic, enséñame
↪ qué plugins hay disponibles que sean útiles para alguien que está
↪ aprendiendo a programar, y recomiéndame 2 o 3 para empezar explicándome qué
↪ hace cada uno.
```

13.7 Crear tu propio plugin

Si tienes skills, subagentes o comandos que usas en varios proyectos, puedes empaquetarlos en un plugin y compartirlo (con tu equipo o públicamente) creando un repositorio con la estructura de marketplace. Pídeselo a Claude Code:

Prompt:

```
Tengo varias skills y subagentes en .claude/ que quiero reutilizar en otros
↪ proyectos. Ayúdame a empaquetarlos como un plugin de Claude Code en un
↪ repositorio nuevo, con la estructura correcta para poder instalarlo con
↪ /plugin install. Explícame los pasos.
```

Consejo

Verifica siempre lo que instalas. Un plugin puede traer hooks y comandos que se ejecutan en tu máquina. Instala solo de fuentes en las que confíes (oficial, tu empresa, autores conocidos) y revisa qué incluye.

Chapter 14

Flujos de trabajo pro

Las funciones y rutinas que diferencian a quien usa Claude Code de vez en cuando de quien lo exprime de verdad: planificar antes de actuar, deshacer sin miedo y trabajar en paralelo.

14.1 Plan Mode (modo planificación)

En Plan Mode, Claude **explora e investiga tu código y propone un plan detallado, pero sin tocar ningún archivo**. Tú lo revisas y, si te convence, le das luz verde para ejecutarlo. Es la mejor red de seguridad para tareas grandes.

14.1.1 Cómo activarlo

- Pulsa `Shift+Tab` para alternar el modo (vuelve a pulsar para salir).
- O al iniciar: `claude--permission-modeplan`

Consejo

Patrón recomendado: usa un modelo potente (Opus) para planificar y uno rápido (Sonnet) para ejecutar. Primero planificas con calma, luego ejecutas con velocidad.

14.2 Checkpoints y Rewind (deshacer)

Cada vez que envías un mensaje, Claude Code crea un **checkpoint**(un punto de guardado). Si un cambio sale mal, puedes **volver atrás**tanto la conversación como el código a un estado anterior.

14.2.1 Cómo deshacer

- Comando `/rewind` para elegir a qué punto volver.
- Pulsa `Esc Esc` (dos veces) como ”botón de pánico”.

Atención

El rewind es tu seguro para refactorizaciones arriesgadas: si Claude se desvía o rompe algo, vuelves al estado bueno en segundos. Aun así, trabajar con git sigue siendo la mejor red de seguridad.

14.3 Tareas en background

Puedes lanzar tareas o subagentes que corran **en segundo plano**mientras tú sigues trabajando en otra cosa en la sesión principal. Perfecto para procesos largos (tests pesados, builds, investigación extensa).

14.3.1 Cómo usarlo

- Lanzar en background: `claude--bg"tutarea"`
- Pulsa `Ctrl+B` o pídele "ejecuta esto en background".
- Gestiona los procesos con `claudeagents` (ver, adjuntar, logs, parar).

14.4 Output styles (formato de salida)

Controla cómo responde Claude Code, útil sobre todo para scripts y automatizaciones:

```
# Salida en texto normal (por defecto)
claude -p "resume los cambios" --output-format text

# Salida en JSON (para procesar con scripts)
claude -p "lista los TODO" --output-format json

# Salida en streaming JSON (para integraciones en vivo)
claude -p "analiza el repo" --output-format stream-json
```

También puedes definir estilos personalizados (p. ej. un modo "explicativo" que enseñe más, ideal para aprender) mediante la configuración o el system prompt.

14.5 Los flujos que más se recomiendan

Esto es lo que la comunidad de Claude Code está compartiendo y recomendando como mejores prácticas:

1. **Plan Mode primero, siempre.** Antes de cualquier tarea seria:`Shift+Tab` revisar el plan ejecutar. Evita sorpresas.
2. **Subagentes en paralelo + background.** Delega investigación, código, tests y revisión a la vez. Monitoriza con `claudeagents`.
3. **Skills estandarizadas.** Crea o instala skills (`https://claude-rho-snowy.vercel.app/skills`) para revisar, desplegar, testear. Usa `disable-model-invocation:true` en las que tengan efectos (como deploy) para que no se lancen solas.
4. **Rewind / Esc Esc como botón de pánico** cuando algo se tuerce.
5. **Hooks + MCP** para integrar tu entorno: formateo automático tras editar, conexión con Slack o tu gestor de tickets.
6. **Revisión intensiva antes de producción.** Una pasada de revisión exigente (con un modelo potente) antes de subir cambios importantes.
7. **`CLAUDE.md` ligero + skills específicas**, y versiona la carpeta `.claude/` en tu repositorio para que el equipo comparta la misma configuración.

Nota

El gran cambio de mentalidad: el flujo que domina ahora mismo no es ”escribir código línea a línea”, sino asignar tareas y revisar los cambios que proponen Claude y sus subagentes. Tu valor está en dirigir bien y revisar con criterio.

14.6 Comprueba tu versión

Claude Code se actualiza muy a menudo (releases frecuentes). Si una función de aquí no te aparece, actualiza:

```
claude --version
npm update -g @anthropic-ai/claude-code
```

Chapter 15

Comandos

Referencia completa de slash commands y flags de CLI disponibles en Claude Code.

15.1 Slash commands

Los slash commands se escriben dentro de la sesión interactiva de Claude Code. Empiezan con `/` y se ejecutan al instante.

Comando	Descripción
`/help`	Muestra la lista de comandos disponibles y ayuda general.
`/clear`	Limpia la pantalla del terminal.
`/reset`	Reinicia la conversación actual (borra el contexto de la sesión).
`/exit`	Sale de Claude Code.
`/plan`	Activa el modo planificación: Claude muestra el plan antes de actuar.
`/permissions`	Abre el diálogo de gestión de permisos interactivo.
`/config`	Abre la configuración de Claude Code.
`/agents`	Lista y gestiona los subagentes activos en la sesión.
`/doctor`	Diagnostica la instalación, autenticación y entorno.
`/hooks`	Gestiona los hooks de automatización de la sesión actual.
`/memory`	Muestra o edita el archivo de memoria (CLAUDE.md).
`/status`	Muestra el estado actual de la sesión, modelo y costos.
`/fast`	Alterna entre modo rápido (Opus con más velocidad) y normal.
`/compact`	Compacta el contexto de la conversación para ahorrar tokens.
`/review`	Solicita a Claude que revise el código actual o los cambios recientes.
`/init`	Inicializa CLAUDE.md en el proyecto actual con instrucciones base.

`/rewind`	Vuelve a un punto anterior (checkpoint): deshace conversación y/o código.
`/plugin`	Gestiona plugins: añadir marketplaces, instalar, activar y desactivar.
`/mcp`	Muestra y gestiona los servidores MCP conectados a la sesión.
`/skills`	Lista las skills disponibles (también las invocas por su nombre: /nombre).

15.2 Flags de la CLI

Los flags se pasan al ejecutar `claude` desde el terminal, antes o después del prompt.

```
claude [flags] [prompt]
claude --model claude-opus-4-8 "refactoriza este archivo"
claude -p "¿qué hace esta función?" < mi_archivo.py
```

Flag	Descripción
`-p/--print`	Modo no interactivo: responde una vez y sale. Ideal para scripts.
`--dangerously-skip-permissions`	Omite todas las confirmaciones de permisos. Usar con precaución.
`--model<id>`	Especifica el modelo a usar. Ej: –model claude-opus-4-8
`--max-tokens<n>`	Limita los tokens de salida por respuesta.
`--no-color`	Desactiva el color en la salida del terminal.
`--version`	Muestra la versión instalada de Claude Code.
`--help`	Muestra la ayuda de la CLI.
`--verbose`	Activa salida detallada para depuración.
`--output-formatjson`	Devuelve la respuesta en JSON. Útil para scripts.

15.3 Modelos disponibles

Puedes cambiar el modelo con `--model` o en la configuración. Claude Code usa `claude-sonnet-4-6` por defecto (velocidad óptima), pero puedes elegir:

Modelo	ID	Velocidad	Ideal para
Claude Fable 5	`claude-fable-5`	Lento	Tareas de máxima complejidad
Claude Opus 4.8	`claude-opus-4-8`	Medio	Razonamiento complejo, análisis profundo
Claude Sonnet 4.6	`claude-sonnet-4-6`	Rápido	Uso general (por defecto en Claude Code)

Claude Haiku 4.5	`claude-haiku-4-5`	Muy rápido	Tareas simples, alta frecuencia, bajo costo

15.4 Atajos de teclado en sesión interactiva

Atajo	Acción
`↑/↓`	Navegar por el historial de prompts
`Ctrl+C`	Interrumpir la respuesta actual o salir
`Ctrl+D`	Salir de Claude Code
`Ctrl+L`	Limpiar pantalla
`Tab`	Autocompletar slash commands
`Shift+Enter`	Nueva línea sin enviar el mensaje
`Esc`	Cancelar edición actual

15.5 Comandos de ejemplo

15.5.1 Sesión headless en CI/CD

```
# Generar un resumen del PR en JSON
claude -p --output-format json "resume los cambios del último commit"

# Revisar seguridad de un archivo
claude -p "revisa posibles vulnerabilidades en src/api/auth.ts" < /dev/null
```

15.5.2 Cambiar modelo para una sesión

```
claude --model claude-opus-4-8
# Ahora usas Opus 4.8 para la sesión completa
```

15.5.3 Ver diagnóstico de instalación

```
# Dentro de Claude Code:
/doctor

# Desde la terminal:
claude doctor
```

Chapter 16

Configuración

Claude Code es altamente configurable. Aprende a personalizar su comportamiento, modelo, memoria y preferencias globales o por proyecto.

16.1 Archivo de configuración principal

Claude Code guarda su configuración en `~/.claude/settings.json`(configuración global) y en `.claude/settings.json` dentro de cada proyecto (configuración local, tiene prioridad).

```
# Ver configuración actual dentro de Claude Code:
/config

# O editar directamente:
nano ~/.claude/settings.json
```

16.1.1 Ejemplo de settings.json

```
{
  "model": "claude-sonnet-4-6",
  "theme": "dark",
  "autoUpdates": true,
  "permissions": {
    "allow": [
      "Bash(npm:*)",
      "Bash(git:*)",
      "Read(**)",
      "Edit(**)"
    ],
    "deny": [
      "Bash(rm -rf:*)",
      "WebFetch(domain:evil.com)"
    ]
  },
  "env": {
    "NODE_ENV": "development"
  }
}
```

Nota

La configuración del proyecto (.claude/settings.json) sobreescribe la global. Puedes añadir este archivo al repositorio para compartir configuración con tu equipo.

16.2 CLAUDE.md — Memoria del proyecto

El archivo `CLAUDE.md` en la raíz de tu proyecto es la ”memoria” de Claude Code. Cuando inicias una sesión, Claude lee este archivo automáticamente para entender el contexto de tu proyecto.

16.2.1 Crear CLAUDE.md automáticamente

```
# Dentro de Claude Code:
/init
```

Claude Code analizará tu proyecto y generará un CLAUDE.md con la estructura, stack tecnológico, comandos importantes y convenciones.

16.2.2 Estructura recomendada de CLAUDE.md

````
# Proyecto: Mi App

## Stack
- Frontend: Next.js 16 + TypeScript + Tailwind CSS
- Backend: Node.js + Express + PostgreSQL
- Tests: Vitest + Playwright

## Comandos esenciales
```bash
npm run dev          # Iniciar dev server (puerto 3000)
npm run test         # Ejecutar tests
npm run build        # Build de producción
npm run db:migrate   # Ejecutar migraciones
```

## Estructura del proyecto
- /app - Páginas Next.js (App Router)
- /components - Componentes reutilizables
- /lib - Utilidades y configuración
- /prisma - Schema de base de datos

## Convenciones
- Usa kebab-case para nombres de archivos
- Los componentes llevan sufijo .tsx
- Los tests van junto al archivo que prueban (*.test.ts)

## Notas importantes
- La rama main está protegida, trabaja en feature branches
- La DB local está en localhost:5432, usuario: dev, sin contraseña
````

16.3 Variables de entorno

Claude Code lee variables de entorno de tu shell. Las más importantes:

Variable	Descripción
ANTHROPIC_API_KEY	Tu API key de Anthropic (obligatoria)
ANTHROPIC_MODEL	Modelo por defecto para la sesión
ANTHROPIC_BASE_URL	URL base para proxies o endpoints personalizados
CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC	Desactiva telemetría (pon 1)
NO_COLOR	Desactiva el color en la salida (pon 1)
CLAUDE_MAX_TOKENS	Tokens máximos por respuesta

16.4 Modelo por defecto

Puedes cambiar el modelo que Claude Code usa en cada sesión. El modelo por defecto es `claude-sonnet-4-6` (equilibrio velocidad/calidad).

```
# En settings.json:
{ "model": "claude-opus-4-8" }

# O como variable de entorno:
export ANTHROPIC_MODEL="claude-opus-4-8"

# O al iniciar Claude Code:
claude --model claude-opus-4-8
```

16.5 Integración con VS Code

Instala la extensión **Claude Code** desde el Marketplace de VS Code. Tras instalarla, Claude Code aparece en el panel lateral y funciona con el mismo contexto de tu proyecto abierto.

- Atajos de teclado: `Cmd+Shift+P` "Claude Code"
- Inline suggestions al escribir código
- Panel de chat integrado con contexto del archivo activo
- Diff view para revisar cambios propuestos

16.6 Integración con JetBrains

Disponible en el Marketplace de JetBrains para IntelliJ IDEA, PyCharm, WebStorm, etc. Mismas capacidades que la extensión de VS Code.

16.7 Configuración de proxy

Si trabajas detrás de un proxy corporativo:

```
export HTTPS_PROXY="https://proxy.empresa.com:8080"
export HTTP_PROXY="http://proxy.empresa.com:8080"
# Luego inicia Claude Code normalmente
claude
```

16.8 Múltiples perfiles / proyectos

Para usar diferentes API keys o configuraciones en distintos proyectos, crea un `.claude/settings.json` en cada proyecto con sus propios valores. Claude Code los detecta automáticamente.

Chapter 17

Servidores MCP

El Model Context Protocol (MCP) permite conectar Claude Code con herramientas externas: bases de datos, APIs, navegadores, servicios en la nube y mucho más.

17.1 ¿Qué es MCP?

MCP (Model Context Protocol) es un estándar abierto creado por Anthropic que define cómo los modelos de IA se comunican con herramientas externas. Es como un "USB para IA": un conector universal que cualquier herramienta puede implementar.

Con MCP, Claude Code puede acceder a recursos que van más allá de tu sistema de archivos local: bases de datos PostgreSQL, APIs REST, el navegador web, Slack, GitHub, Jira, y cualquier servicio que tenga un servidor MCP.

Nota

MCP en Claude Code 2026: Claude Code ahora soporta MCP nativo sin necesidad de configuración manual. Muchos servidores se activan automáticamente según el contexto del proyecto.

17.2 Cómo funciona

Un servidor MCP es un proceso (local o remoto) que expone:

- **Herramientas (tools):** funciones que Claude puede llamar (ej: ejecutar SQL, buscar en Slack).
- **Recursos (resources):** datos que Claude puede leer (ej: esquema de la DB, documentación).
- **Prompts:** instrucciones especializadas para tareas concretas.

17.3 Añadir un servidor MCP

Los servidores MCP se configuran en `.claude/settings.json`:

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres",
               "postgresql://localhost/midb"],
```

```
      "env": {}
    },
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_xxxxxxxx"
      }
    }
  }
}
```

O añade servidores directamente desde la CLI:

```
claude mcp add <nombre> -- <comando> [args...]

# Ejemplo: servidor de PostgreSQL
claude mcp add postgres -- npx -y @modelcontextprotocol/server-postgres \
  "postgresql://localhost/midb"

# Ejemplo: servidor de filesystem (solo lectura)
claude mcp add files -- npx -y @modelcontextprotocol/server-filesystem \
  /ruta/al/directorio
```

17.4 Servidores MCP populares

Servidor	Paquete npm	Para qué sirve
PostgreSQL	`@modelcontextprotocol/server-postgres`	Consultas SQL directas a tu base de datos
SQLite	`@modelcontextprotocol/server-sqlite`	Base de datos SQLite local
GitHub	`@modelcontextprotocol/server-github`	Issues, PRs, repos de GitHub
Brave Search	`@modelcontextprotocol/server-brave-search`	Búsquedas web en tiempo real
Puppeteer	`@modelcontextprotocol/server-puppeteer`	Control de navegador web
Slack	`@modelcontextprotocol/server-slack`	Leer/enviar mensajes de Slack
Google Drive	`@modelcontextprotocol/server-gdrive`	Acceso a archivos de Google Drive
Filesystem	`@modelcontextprotocol/server-filesystem`	Acceso controlado al sistema de archivos
Memory	`@modelcontextprotocol/server-memory`	Memoria persistente entre sesiones
Sentry	`@modelcontextprotocol/server-sentry`	Errores y performance de Sentry

17.5 Gestionar servidores MCP

```
# Listar servidores configurados
claude mcp list

# Ver detalles de un servidor
claude mcp get postgres

# Eliminar un servidor
claude mcp remove postgres

# Dentro de Claude Code, ver servidores activos:
/mcp
```

17.6 Ámbito de los servidores MCP

Los servidores MCP tienen tres ámbitos posibles:

- **Local** (por defecto): disponible solo en el proyecto actual. Se guarda en `.claude/settings.json`.
- **Usuario**: disponible en todos tus proyectos. Se guarda en `~/.claude/settings.json`. Añade `--scopeuser` al comando.
- **Proyecto**: compartido con el equipo via control de versiones.

```
# Añadir servidor a nivel de usuario (global)
claude mcp add --scope user github -- npx -y @modelcontextprotocol/server-github
```

17.7 Crear tu propio servidor MCP

Cualquier script que implemente el protocolo MCP puede ser un servidor. Anthropic proporciona SDKs para Python y TypeScript:

```
# TypeScript
npm install @modelcontextprotocol/sdk

# Python
pip install mcp
```

17.7.1 Ejemplo mínimo en TypeScript

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

const server = new Server({ name: "mi-servidor", version: "1.0.0" }, {
  capabilities: { tools: {} }
});

server.setRequestHandler("tools/list", async () => ({
  tools: [{
    name: "saludar",
```

```
    description: "Saluda al usuario",
    inputSchema: {
      type: "object",
      properties: { nombre: { type: "string" } },
      required: ["nombre"]
    }
  }]
}));

server.setRequestHandler("tools/call", async (request) => {
  if (request.params.name === "saludar") {
    return { content: [{ type: "text", text: `Hola,
    ↪  ${request.params.arguments.nombre}!` }] };
  }
});

const transport = new StdioServerTransport();
await server.connect(transport);
```

17.8 MCP en la nube vs local

Los servidores MCP pueden ejecutarse localmente (como proceso en tu máquina) o en la nube (conectados via HTTP/SSE). Claude Code soporta ambos:

```
# Servidor local (stdio)
{
  "command": "node",
  "args": ["./mi-servidor-mcp.js"]
}

# Servidor remoto (HTTP/SSE)
{
  "url": "https://mi-servidor.com/mcp",
  "headers": { "Authorization": "Bearer TOKEN" }
}
```

Chapter 18

Hooks

Los hooks son scripts de shell que se ejecutan automáticamente en respuesta a eventos de Claude Code. Te dan control total sobre el comportamiento de la herramienta.

18.1 ¿Qué son los hooks?

Los hooks te permiten interceptar y reaccionar a eventos del ciclo de vida de Claude Code: antes de ejecutar una herramienta, después de hacerlo, cuando Claude termina una respuesta, etc.

Casos de uso comunes:

- Formatear automáticamente el código tras cada edición.
- Registrar en un log todas las operaciones de Claude.
- Bloquear operaciones peligrosas con lógica personalizada.
- Enviar notificaciones cuando Claude termina una tarea larga.
- Ejecutar tests automáticamente después de cada cambio.

18.2 Tipos de hooks

Evento	Cuándo se dispara	Puede bloquear
`PreToolCall`	Antes de ejecutar cualquier herramienta	Sí
`PostToolCall`	Después de ejecutar cualquier herramienta	No
`Stop`	Cuando Claude termina una respuesta completa	No
`SubagentStop`	Cuando un subagente termina su tarea	No
`Notification`	Cuando Claude emite una notificación al usuario	No

18.3 Configurar hooks

Los hooks se configuran en `.claude/settings.json` (proyecto) o `~/.claude/settings.json` (global):

```
{
  "hooks": {
    "PostToolCall": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "prettier --write $CLAUDE_FILE_PATH"
          }
        ]
      }
    ],
    "Stop": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification "Claude terminó" with
            ↪  title "Claude Code"'"
          }
        ]
      }
    ]
  }
}
```

18.4 Variables de entorno disponibles en hooks

Variable	Descripción
`CLAUDE_TOOL_NAME`	Nombre de la herramienta que se ejecutó
`CLAUDE_TOOL_INPUT`	Input de la herramienta (JSON)
`CLAUDE_TOOL_OUTPUT`	Output de la herramienta (PostToolCall)
`CLAUDE_FILE_PATH`	Ruta del archivo afectado (Edit/Write/Read)
`CLAUDE_SESSION_ID`	ID único de la sesión actual
`CLAUDE_PROJECT_DIR`	Directorio raíz del proyecto
`CLAUDE_HOOK_TYPE`	Tipo de hook (PreToolCall, PostToolCall, Stop...)

18.5 El matcher

El campo `matcher` es una expresión regular que se compara con el nombre de la herramienta. Si no se especifica, el hook se aplica a todas las herramientas.

```
# Solo edición de archivos TypeScript
"matcher": "Edit"

# Herramientas de bash y edición
"matcher": "Bash|Edit|Write"
```

```
# Cualquier herramienta de lectura
"matcher": "^Read"
```

18.6 Hook de tipo command

El tipo más común. Ejecuta un comando de shell. El hook puede leer información del evento via variables de entorno o via stdin (JSON):

```
{
  "type": "command",
  "command": "bash /ruta/a/mi-hook.sh",
  "timeout": 30
}
```

18.6.1 Ejemplo: hook que lee datos del evento por stdin

```
#!/bin/bash
# mi-hook.sh - recibe el evento completo como JSON por stdin
EVENT=$(cat)
TOOL=$(echo $EVENT | jq -r '.tool_name')
FILE=$(echo $EVENT | jq -r '.tool_input.path // ""')

echo "Herramienta: $TOOL, Archivo: $FILE" >> ~/.claude/hook.log
```

18.7 Bloquear operaciones con PreToolCall

Un hook `PreToolCall` puede devolver un código de salida distinto de 0 para bloquear la operación:

```
#!/bin/bash
# Bloquear rm -rf
TOOL=$(cat | jq -r '.tool_name')
CMD=$(cat | jq -r '.tool_input.command // ""')

if [[ "$CMD" == *"rm -rf"* ]]; then
  echo "BLOQUEADO: rm -rf no permitido"
  exit 1  # exit 1 = bloquear la operación
fi

exit 0  # exit 0 = permitir
```

Configúralo así en settings.json:

```
{
  "hooks": {
    "PreToolCall": [{
      "matcher": "Bash",
      "hooks": [{
        "type": "command",
        "command": "bash ~/.claude/hooks/bloquear-rm.sh"
```

```
      }]
    }]
  }
}
```

18.8 Ejemplos prácticos

18.8.1 Formatear con Prettier tras edición

```
{
  "hooks": {
    "PostToolCall": [{
      "matcher": "Edit|Write",
      "hooks": [{
        "type": "command",
        "command": "npx prettier --write "$CLAUDE_FILE_PATH" 2>/dev/null || true"
      }]
    }]
  }
}
```

18.8.2 Ejecutar tests tras cambios

```
{
  "hooks": {
    "Stop": [{
      "hooks": [{
        "type": "command",
        "command": "npm test --watchAll=false 2>&1 | tail -20"
      }]
    }]
  }
}
```

18.8.3 Log de todas las operaciones

```
{
  "hooks": {
    "PostToolCall": [{
      "hooks": [{
        "type": "command",
        "command": "echo "$(date) - $CLAUDE_TOOL_NAME: $CLAUDE_FILE_PATH" >>
        ↪  ~/.claude/audit.log"
      }]
    }]
  }
}
```

Consejo

Consejo: Usa 2>/dev/null || true al final de tus comandos de hook para evitar que errores menores interrumpan el flujo de Claude.

18.9 Gestionar hooks desde la CLI

```
# Ver hooks configurados (dentro de Claude Code)
/hooks

# O abre settings.json directamente:
claude config
```

Chapter 19

Permisos

El sistema de permisos de Claude Code garantiza que nada ocurra sin tu conocimiento. Aprende a configurar qué puede y qué no puede hacer Claude.

19.1 Filosofía de permisos

Claude Code sigue el principio de **mínimo privilegio**: por defecto pide confirmación para cualquier acción potencialmente irreversible o que afecte a recursos fuera de tu proyecto. Tú decides cuándo darle más autonomía.

19.2 Herramientas y sus permisos

Claude Code tiene acceso a estas herramientas, cada una con su nivel de riesgo por defecto:

Herramienta	Qué hace	Confirmación por defecto
`Read`	Leer archivos de tu proyecto	No
`Edit`	Editar archivos existentes	Sí (muestra diff)
`Write`	Crear nuevos archivos	Sí
`Bash`	Ejecutar comandos de terminal	Sí
`WebFetch`	Hacer peticiones HTTP	Sí
`WebSearch`	Buscar en internet	No
`TodoRead/Write`	Gestionar lista de tareas interna	No

19.3 Permitir y denegar operaciones

Configura permisos en `.claude/settings.json` para no tener que confirmar operaciones frecuentes:

```
{
  "permissions": {
    "allow": [
      "Bash(npm run:*)",
      "Bash(git:*)",
      "Bash(npx:*)",
      "Read(**)",
```

```
      "Edit(**)",
      "Write(**)",
      "WebFetch(domain:api.github.com)"
    ],
    "deny": [
      "Bash(rm -rf:*)",
      "Bash(sudo:*)",
      "WebFetch(domain:*.evil.com)"
    ]
  }
}
```

19.3.1 Sintaxis de permisos

Los permisos usan el formato `Herramienta(patrón:valor)`:

- `Bash(npm:*)` — permite cualquier comando npm
- `Bash(gitcommit:*)` — permite git commit con cualquier argumento
- `Read(**)` — permite leer cualquier archivo (`**` = cualquier ruta)
- `WebFetch(domain:api.example.com)` — permite fetch solo a ese dominio
- `Edit(src/**)` — permite editar solo archivos bajo `src/`

19.4 Gestión interactiva de permisos

Dentro de Claude Code, usa el comando:

```
/permissions
```

Esto abre un panel interactivo donde puedes ver, añadir o revocar permisos de la sesión actual.

19.5 Permisos durante una sesión

Cuando Claude Code solicita hacer algo que no tienes pre-autorizado, te mostrará un diálogo como este:

```
Claude quiere ejecutar:
  git push origin main

¿Permitir? [s/N/siempre/nunca] _
```

- `s` — permitir solo esta vez.
- `N` — denegar (Claude buscará otra alternativa).
- `siempre` — añadir a la lista de permisos permanentes.
- `nunca` — añadir a la lista de denegados permanentes.

19.6 Modo sin permisos (peligroso)

Para entornos controlados (CI, Docker, sandboxes), puedes desactivar todas las confirmaciones:

```
claude --dangerously-skip-permissions "implementa los tests unitarios"
```

Atención

Solo para entornos aislados. Con este flag, Claude puede ejecutar cualquier comando sin confirmación. Úsalo en contenedores Docker o VMs donde el daño potencial esté contenido.

19.7 Buenas prácticas de seguridad

- **Usa git:** trabajar en un repositorio git te permite revertir cualquier cambio que Claude haya hecho con `gitrestore`.
- **Limita el scope en producción:** en servidores de producción, no le des permisos de escritura a Claude.
- **Revisa los diffs:** Claude siempre muestra un diff antes de editar. Léelo antes de aceptar.
- **Permisos por dominio:** si usas WebFetch, especifica los dominios exactos en lugar de `WebFetch(*)`.
- **Variables de entorno:** Claude no puede leer tus secretos a menos que estén en el entorno o se los pases explícitamente.

19.8 Configuración de permisos para equipo

Añade `.claude/settings.json` a tu repositorio para que todo el equipo use la misma configuración de permisos base:

```
# .gitignore - NO ignorar settings.json del equipo
# (sí ignorar settings.local.json para configs personales)
.claude/settings.local.json
```

Nota

Existe un archivo settings.local.json para configuraciones personales que no deben compartirse (como tu API key o rutas locales). Claude Code lo lee y tiene prioridad sobre settings.json.

Chapter 20

Uso avanzado

Técnicas avanzadas para sacar el máximo partido a Claude Code: subagentes, worktrees, integración en CI/CD, modo headless y más.

20.1 Subagentes

Claude Code puede lanzar **subagentes**: instancias paralelas de Claude que trabajan de forma independiente en subtareas. Esto es especialmente útil para tareas que se pueden paralelizar.

Ejemplos donde los subagentes brillan:

- Refactorizar 20 archivos en paralelo.
- Generar tests para múltiples módulos simultáneamente.
- Investigar distintas soluciones a la vez y elegir la mejor.

Lanzar subagentes con el SDK de Claude:

```
# Dentro de Claude Code, Claude decide cuándo paralelizar
> "Genera tests unitarios para cada archivo en src/components. Hazlo en
↪  paralelo."

# Claude lanzará un subagente por archivo automáticamente
```

Nota

Los subagentes se gestionan automáticamente. Claude decide cuándo crear uno basándose en la complejidad y paralelizabilidad de la tarea. Puedes ver los activos con /agents.

20.2 Git worktrees

Los **git worktrees** permiten trabajar en múltiples ramas del mismo repositorio simultáneamente, en directorios distintos. Claude Code los soporta nativamente y los usa para aislar cambios de subagentes:

```
# Crear un worktree para una feature branch
git worktree add ../mi-proyecto-feature feature/nueva-api

# Claude Code con isolation de worktree
claude --isolation worktree "implementa la nueva API de pagos en una branch
↪  separada"
```

Con `--isolationworktree`, Claude Code crea automáticamente un worktree temporal, hace los cambios ahí y te propone un PR al terminar. Si no acepta los cambios, el worktree se limpia solo.

20.3 Modo headless / CI-CD

Claude Code puede ejecutarse sin interfaz interactiva, ideal para pipelines de CI/CD, scripts y automatizaciones:

```
# Modo headless básico
claude -p "revisa el código en busca de vulnerabilidades SQL"

# Con output JSON para parsear
claude -p --output-format json "lista todos los TODO del proyecto" | jq '.result'

# En un Makefile o script CI
claude -p --dangerously-skip-permissions \
  "ejecuta los tests, si hay fallos corrígelos y haz commit"

# En GitHub Actions
- name: Claude Code Review
  run: |
    claude -p "revisa los cambios del PR y comenta problemas de seguridad" \
      --output-format json > review.json
  env:
    ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
```

20.4 Modo /fast (Opus acelerado)

El modo fast usa Claude Opus con mayor velocidad de respuesta. Actívalo o desactívalo durante la sesión:

```
# Dentro de Claude Code
/fast

# O al iniciar
claude --fast
```

20.5 Sesiones largas y compactación

En sesiones largas, el contexto se llena y Claude Code puede volverse más lento o perder coherencia. Soluciones:

20.5.1 Compactar el contexto

```
# Dentro de Claude Code
/compact

# Claude resume la conversación hasta el momento
# y la reemplaza por un resumen compacto
```

20.5.2 Reanudar una sesión

```
# Ver sesiones recientes
claude resume

# Continuar la última sesión
claude resume --last

# El contexto se restaura automáticamente
```

20.6 CLAUDE.md en subdirectorios

Puedes tener archivos `CLAUDE.md` en subdirectorios de tu proyecto. Claude los leerá automáticamente cuando trabaje en esa carpeta, dándole instrucciones específicas para esa parte del código:

```
mi-proyecto/
+-- CLAUDE.md              <- instrucciones globales
+-- frontend/
|   +-- CLAUDE.md          <- instrucciones para el frontend
+-- backend/
|   +-- CLAUDE.md          <- instrucciones para el backend
+-- docs/
    +-- CLAUDE.md          <- instrucciones para documentación
```

20.7 Flujos de trabajo con git

20.7.1 Crear feature branches automáticamente

```
> "Implementa el sistema de notificaciones push. Crea una feature branch,
   haz los cambios necesarios y al terminar prepara el PR."

# Claude hará:
# 1. git checkout -b feature/push-notifications
# 2. Implementar los cambios
# 3. git add + git commit con mensaje descriptivo
# 4. Preparar el cuerpo del PR para que lo apruebes
```

20.7.2 Code review automatizado

```
# Revisar el diff del último commit
claude -p "revisa este diff en busca de bugs y problemas de rendimiento" \
  <<< "$(git diff HEAD~1)"

# Revisar todo un PR
gh pr diff 123 | claude -p "haz un code review completo"
```

20.8 Integración con tmux / pantallas divididas

Un flujo de trabajo popular es tener Claude Code en un panel y tu editor en otro:

```
# Crear sesión tmux con dos paneles
tmux new-session -d -s dev
tmux split-window -h
tmux send-keys -t dev:0.0 "claude" Enter  # Claude Code a la izquierda
tmux send-keys -t dev:0.1 "nvim ." Enter  # Editor a la derecha
```

20.9 Tips de productividad

- **Sé específico con el contexto:** menciona el archivo, la función y el error exacto. Cuanto más específico, mejor resultado.
- **Un alias útil:** `aliasai="claude-p"`para consultas rápidas sin entrar al modo interactivo.
- **Memoria entre sesiones:** mantén tu CLAUDE.md actualizado con las decisiones de arquitectura y convenciones del proyecto.
- **Tareas grandes:** divide el trabajo en subtareas menores. Claude trabaja mejor con objetivos acotados y claros.
- **Revisar antes de aceptar:** usa siempre el diff view para entender exactamente qué va a cambiar antes de confirmar.

20.10 Exportar la sesión

```
# Guardar el transcript de la sesión actual
claude -p --output-format json "resumen del trabajo de hoy" > sesion-$(date
↪  +%Y%m%d).json
```

Chapter 21

Preguntas frecuentes

Las dudas que casi todo el mundo tiene al empezar con Claude Code, respondidas sin rodeos.

21.1 Coste y cuenta

21.2 Seguridad y privacidad

21.3 Uso y aprendizaje

21.4 Comparativas

Consejo

¿No está tu duda aquí? Pregúntasela directamente a Claude Code: escribe ”explícame [tu duda] sobre cómo funcionas”. O revisa la solución de problemas si algo no te funciona.

Chapter 22

Solución de problemas

Los tropiezos más habituales al empezar y cómo resolverlos. Si algo no te funciona, probablemente esté aquí.

Consejo

Lo primero que debes probar: el comando de diagnóstico. Dentro de Claude Code escribe /doctor (o claude doctor desde la terminal). Comprueba instalación, autenticación y entorno, y te dice qué falla.

22.1 Al instalar

22.1.1 "command not found: claude"

Instalaste Claude Code pero la terminal no lo encuentra. Causas habituales:

- La instalación de npm no terminó bien. Reinstala: `npminstall-g@anthropic-ai/claude-code`
- La carpeta global de npm no está en tu `PATH`. Cierra y abre la terminal de nuevo.
- Aún sin solución: comprueba dónde instala npm con `npmconfiggetprefix` y asegúrate de que esa ruta `/bin` esté en tu PATH.

22.1.2 "EACCES: permission denied" al instalar

npm no tiene permisos para instalar globalmente. **No uses `sudo`**(causa más problemas). En su lugar, lo ideal es instalar Node.js con un gestor de versiones como `nvm`, que evita estos problemas de permisos.
Prompt:

```
Al instalar Claude Code con npm me sale "EACCES: permission denied". Estoy en
↪  macOS. Ayúdame a arreglarlo de la forma correcta, sin usar sudo. Si
↪  recomiendas instalar nvm, guíame paso a paso.
```

22.1.3 "Node.js version too old" / versión incompatible

Claude Code necesita **Node.js 20 o superior**. Comprueba tu versión con `node--version`. Si es menor, actualiza Node.js (con nvm: `nvminstall20&&nvmuse20`).

22.2 Al iniciar sesión / autenticación

22.2.1 "Invalid API key" o errores de autenticación

- Comprueba que copiaste la clave entera, sin espacios al principio o final.
- Verifica que la variable está bien puesta: `echo$ANTHROPIC_API_KEY` debe mostrarla.
- Si la pusiste en `~/.zshrc`, recarga con `source~/.zshrc` o abre una terminal nueva.
- La clave puede estar revocada o agotada. Genera una nueva en `console.anthropic.com`.

22.2.2 "Rate limit exceeded" / "Too many requests"

Has hecho demasiadas peticiones en poco tiempo, o alcanzaste el límite de tu plan. Espera unos minutos. Si es recurrente, revisa los límites de tu plan o, en API, tu nivel de uso (tier) en el panel de Anthropic.

22.2.3 "Insufficient credit" / saldo agotado

Si usas API, se acabaron tus créditos. Añade saldo o un método de pago en `console.anthropic.com`. Si usas suscripción, puede que hayas alcanzado el límite de uso de tu plan; espera a que se renueve o sube de plan.

22.3 Durante el uso

22.3.1 Claude Code va lento o se "atasca"

- **Sesión muy larga:** el contexto se ha llenado. Usa `/compact` para resumirlo, o `/clear` para empezar limpio (perderás el contexto de la conversación, no tus archivos).
- **Tarea muy grande:** divídela en partes más pequeñas. Mira cómo escribir buenos prompts (`https://claude-rho-snowy.vercel.app/prompts`).
- **Conexión:** Claude Code necesita internet estable.

22.3.2 Hace cambios que yo no quería

- Usa `/rewind` (o `Esc Esc`) para deshacer al estado anterior. Ver Flujos de trabajo (`https://claude-rho-snowy.vercel.app/flujos`).
- Si usas git: `gitrestore.` revierte los cambios no guardados.
- Para el futuro: usa **Plan Mode** (`Shift+Tab`) y revisa el plan antes de que actúe.

22.3.3 No me pide permiso / me pide demasiado permiso

Ajusta la lista de permisos. Si te pregunta por cosas que siempre permites (como `npmrun`), añádelas a la lista `allow`. Si quieres más control, revisa tu configuración. Todo en Permisos (`https://claude-rho-snowy.vercel.app/permisos`).

22.3.4 Una función de la documentación no me aparece

Probablemente tienes una versión antigua. Claude Code se actualiza muy a menudo:

```
claude --version
npm update -g @anthropic-ai/claude-code
```

22.3.5 Un servidor MCP no conecta

- Revisa la lista con `/mcp` dentro de Claude Code.
- Comprueba que el comando del servidor es correcto y que tiene las variables de entorno necesarias (tokens, rutas).
- Mira los detalles en Servidores MCP (`https://claude-rho-snowy.vercel.app/mcp`).

22.4 El comodín que siempre funciona

Si te bloqueas con cualquier error, **pregúntale a Claude Code directamente**. Es literalmente experto en resolver problemas técnicos. Pégale el error completo:
Prompt:

```
Estoy teniendo este problema con Claude Code (o con mi proyecto):

[describe qué intentabas hacer y pega el error completo]

Explícame qué significa y cómo solucionarlo paso a paso. Soy principiante.
```

Nota

¿Sigue sin funcionar? Ejecuta /doctor y, si el problema persiste, busca en el repositorio oficial de Claude Code en GitHub (sección Issues) por si es un fallo conocido con solución.

Chapter 23

Recursos

Enlaces oficiales y de la comunidad, actualizados (2026), para profundizar en Claude Code.

Nota

Enlaces externos: estas páginas no dependen de esta guía y pueden cambiar, moverse o actualizarse con el tiempo.

23.1 Documentación oficial

- Quickstart oficial (`https://code.claude.com/docs/en/quickstart`)
- Documentación principal (`https://code.claude.com/docs`)
- Claude Code 101 (curso oficial Anthropic) (`https://anthropic.skilljar.com/claude-code-101`)
- Cómo usan Claude Code los equipos de Anthropic (PDF) (`https://www-cdn.anthropic.com/58284b19e702b49db9302d5b6f135ad8871e7658.pdf`)

23.2 Skills

- Guía completa para crear Skills (PDF oficial Anthropic) (`https://resources.anthropic.com/hubfs/The-Complete-Guide-to-Building-Skill-for-Claude.pdf`)
- Repositorio oficial de Skills (Anthropic) (`https://github.com/anthropics/skills`)
- Awesome Claude Skills (curado, +1000) (`https://github.com/ComposioHQ/awesome-claude-skills`)
- Colección de skills (337+) (`https://github.com/alirezarezvani/claude-skills`)
- superpowers (framework: convierte Claude Code en dev senior) (`https://github.com/obra/superpowers`)
- awesome-claude-code-toolkit (agentes + skills + hooks + MCP) (`https://github.com/rohitg00/awesome-claude-code-toolkit`)
- claude-mem (`https://github.com/thedotmack/claude-mem`)
- obsidian-skills (`https://github.com/kepano/obsidian-skills`)

23.3 Herramientas

- Repomix (empaqueta tu repo en un archivo para dar contexto) (https://github.com/yamadashy/repomix)

23.4 MCP (Model Context Protocol)

- Sitio oficial de MCP (https://modelcontextprotocol.io/)
- Repositorio oficial de servidores MCP (https://github.com/modelcontextprotocol/servers)
- GitHub MCP Server (el más usado) (https://github.com/github/github-mcp-server)

23.5 Cursos y tutoriales en español (YouTube)

- Claude Code 2026: Curso Completo (Benjamín Cordero) (https://www.youtube.com/watch?v=73eFWU-edO4)
- Claude Code Desde Cero, Curso Completo (Agustín Medina) (https://www.youtube.com/playlist?list=PLwjjPKgjif66EDzC_QaRANOjwjo-mmJ7_)
- Claude Code: Curso Completo 2 Horas (https://www.youtube.com/watch?v=K4e2l49RSrY)

23.6 Tutoriales en inglés

- The Claude Code Handbook (FreeCodeCamp) (https://www.freecodecamp.org/news/claude-code-handbook/)
- Net Ninja – Claude Code (playlist) (https://www.youtube.com/playlist?list=PL4cUxeGkcC9g4YJeBqChhFJwKQ9TRiivY)
- Programming with Mosh – Claude Code (https://www.youtube.com/watch?v=IuyVVtr1uhY)

23.7 Cuentas de X para estar al día

- @bcherny (creador de Claude Code) (https://x.com/bcherny)
- @trq212 (desarrollador de Claude Code) (https://x.com/trq212)
- @rubenhassid (guías gratuitas) (https://x.com/rubenhassid)
- @charliejhills (recopilaciones de skills) (https://x.com/charliejhills)

Chapter 24

Comparativa

En qué se diferencia Claude Code de otras herramientas de IA para programar, para ayudarte a elegir.

Herramienta	Tipo	Acceso a tu proyecto	Ideal para
Claude Code	CLI	Lee, edita y ejecuta comandos dentro de tu proyecto con permisos explícitos.	Cambios de varios archivos, refactors, investigación del código y automatización desde terminal.
Cursor	Editor	Trabaja dentro del editor, con contexto del workspace y asistencia integrada al flujo de edición.	Programar con IA sin salir del IDE, autocompletado, chat sobre el código y cambios guiados.
Windsurf	Editor	Accede al proyecto desde el editor y coordina ediciones con agentes y contexto del workspace.	Flujos visuales dentro del IDE, navegación asistida y desarrollo iterativo con ayuda contextual.
GitHub Copilot	Editor	Se integra en editores y repositorios de GitHub, con contexto del archivo y del proyecto según configuración.	Autocompletado rápido, ayuda puntual, generación de funciones y asistencia continua mientras escribes.
ChatGPT (web)	Chat	No actúa directamente sobre tu proyecto salvo que subas archivos o copies contexto manualmente.	Explicaciones, diseño de soluciones, aprendizaje, revisión de ideas y consultas sin tocar el repositorio.

24.1 ¿Cuándo elegir Claude Code?

Claude Code encaja especialmente bien cuando quieres que la IA trabaje sobre el repositorio real: leer varios archivos, proponer un plan, editar código, ejecutar tests, revisar errores y dejar cambios listos para inspeccionar. Es fuerte en tareas largas donde importa entender el proyecto completo, no solo completar la línea actual.

24.2 Claude Code + tu editor (no es o lo uno o lo otro)

No tienes que elegir una sola herramienta. Claude Code funciona desde la terminal y también se integra con VS Code y JetBrains, así que puede convivir con Cursor, Windsurf o Copilot. Puedes usar el editor para escribir y navegar, y Claude Code para encargos más amplios como refactors, migraciones, pruebas o análisis de bugs.

Consejo

Recomendación práctica: usa Claude Code para tareas con varios pasos y validación en el proyecto; usa un editor con IA para el trabajo fino y continuo mientras programas.

24.3 Resumen

Claude Code destaca como agente de terminal orientado a operar sobre tu código. Cursor, Windsurf y Copilot brillan dentro del editor. ChatGPT web es muy útil para pensar, aprender y explicar. La mejor elección depende de si necesitas conversación, autocompletado, edición asistida o ejecución real en el proyecto.

Chapter 25

Sugerencias y contacto

Si tienes sugerencias, dudas o quieres proponer mejoras para esta guía, puedes escribir a learntouseai@gmail.com.