Claude Code: Mejores Prácticas

> Traducción del artículo original de Anthropic: Claude Code Best Practices

---

1. Personaliza tu configuración

Claude Code es un asistente de codificación agéntico que incorpora contexto automáticamente a las indicaciones. Esta recopilación de contexto consume tiempo y tokens, pero puedes optimizarla mediante el ajuste del entorno.

a. Crear archivos CLAUDE.md

`CLAUDE.md` es un archivo especial que Claude pone automáticamente en contexto al iniciar una conversación. Es el lugar ideal para documentar:

- Comandos bash comunes
- Archivos principales y funciones de utilidad
- Pautas de estilo de código
- Instrucciones de pruebas
- Etiqueta del repositorio (nomenclatura de ramas, merge vs rebase, etc.)
- Configuración del entorno del desarrollador (pyenv, compiladores, etc.)
- Comportamientos inesperados o advertencias particulares del proyecto
- Cualquier información que quieras que Claude recuerde

No existe un formato requerido. Recomendamos mantenerlos concisos y legibles:

```
# Comandos Bash

- npm run build: Compilar el proyecto
- npm run typecheck: Ejecutar el verificador de tipos

# Estilo de código

- Usar sintaxis ES modules (import/export), no CommonJS (require)
- Desestructurar imports cuando sea posible (ej. import { foo } from 'bar')
```

```
# Flujo de trabajo

- Verificar tipos al terminar una serie de cambios
- Preferir ejecutar tests individuales, no la suite completa
```

Ubicaciones válidas para CLAUDE.md:

Ubicación	Uso
Raíz del repositorio	Uso más común. Nombrar `CLAUDE.md` y commitear (recomendado)
Raíz como `.local`	`CLAUDE.local.md` y añadir a `.gitignore`
Directorios padre	Útil para monorepos
Directorios hijo	Claude los incorporará bajo demanda
Carpeta home	`~/.claude/CLAUDE.md` - aplica a todas las sesiones

> **Tip:** Ejecuta `/init` para que Claude genere automáticamente un `CLAUDE.md`.

b. Afina tus archivos CLAUDE.md

Tus archivos `CLAUDE.md` forman parte de los prompts de Claude, así que deben perfeccionarse como cualquier prompt. Un error común es agregar contenido extenso sin iterar sobre su efectividad.

Puedes añadir contenido manualmente o presionar `#` para darle a Claude una instrucción que incorporará automáticamente. En Anthropic, ocasionalmente pasamos los `CLAUDE.md` por el *prompt improver* y afinamos las instrucciones (agregando énfasis con "IMPORTANTE" o "DEBES") para mejorar la adherencia.

c. Gestiona la lista de herramientas permitidas

Por defecto, Claude Code solicita permiso para cualquier acción que pueda modificar tu sistema. Puedes personalizar esta lista:

Cuatro formas de gestionar permisos:

1. Seleccionar "Permitir siempre" cuando se solicite durante una sesión
2. Usar `/permissions` para agregar o eliminar herramientas
3. Editar manualmente `.claude/settings.json` o `~/.claude.json`
4. Usar `--allowedTools` para permisos específicos de sesión

Ejemplos de permisos:

- `Edit` - Permitir siempre edición de archivos
- `Bash(git commit:*)` - Permitir commits de git
- `mcp__puppeteer__puppeteer_navigate` - Permitir navegación con Puppeteer MCP

d. Si usas GitHub, instala la CLI `gh`

Claude sabe usar `gh` para interactuar con GitHub: crear issues, abrir PRs, leer comentarios y más.

---

2. Dale más herramientas a Claude

Claude tiene acceso a tu entorno shell y puede aprovechar herramientas más complejas vía APIs MCP y REST.

a. Usa Claude con herramientas bash

Claude hereda tu entorno bash. Aunque conoce utilidades comunes, no sabrá de tus herramientas personalizadas sin instrucciones:

- Dile el nombre de la herramienta con ejemplos de uso
- Dile que ejecute `--help` para ver la documentación
- Documenta las herramientas frecuentes en `CLAUDE.md`

b. Usa Claude con MCP

Claude Code funciona como servidor y cliente MCP. Como cliente, puede conectarse a servidores MCP de tres maneras:

1. **Configuración del proyecto** - Disponible al ejecutar Claude en ese directorio
2. **Configuración global** - Disponible en todos los proyectos
3. **Archivo** `.mcp.json` - Disponible para cualquiera que trabaje en el código base

> **Tip:** Usa `--mcp-debug` para identificar problemas de configuración.

c. Usa comandos de barra personalizados

Para flujos repetitivos, almacena plantillas en `.claude/commands/`. Estarán disponibles vía `/` al escribir.

Los comandos pueden incluir `$ARGUMENTS` para pasar parámetros.

Ejemplo: `.claude/commands/fix-github-issue.md`

```
Analiza y corrige el issue de GitHub: $ARGUMENTS.

Sigue estos pasos:

1. Usa `gh issue view` para obtener los detalles

2. Comprende el problema descrito

3. Busca archivos relevantes en el código

4. Implementa los cambios necesarios

5. Escribe y ejecuta tests para verificar

6. Asegura que el código pase linting y verificación de tipos

7. Crea un mensaje de commit descriptivo

8. Push y crea un PR

Recuerda usar la CLI de GitHub (`gh`) para todas las tareas relacionadas con GitHub.
```

Uso: `/project:fix-github-issue 1234`

---

3. Prueba flujos de trabajo comunes

Claude Code no impone un flujo específico. Estos patrones han emergido como exitosos:

a. Explorar, Planificar, Codificar, Confirmar

Flujo versátil para muchos problemas:

1. **Explorar:** Pide a Claude que lea archivos, imágenes o URLs relevantes, diciéndole explícitamente que **no escriba código todavía**. Usa subagentes para verificar detalles o investigar preguntas.
2. **Planificar:** Pide a Claude que haga un plan. Usa "pensar" para activar el modo de pensamiento extendido:
   - "pensar" < "pensar duro" < "pensar más duro" < "pensar ultra"
   - Cada nivel asigna progresivamente más presupuesto de reflexión
3. **Codificar:** Pide que implemente la solución, verificando la razonabilidad durante la implementación.
4. **Confirmar:** Pide que haga commit y cree un PR. Si es relevante, que actualice README o changelog.

> **Importante:** Los pasos 1 y 2 son cruciales. Sin ellos, Claude tiende a saltar directamente a codificar.

b. Escribir tests, confirmar; codificar, iterar, confirmar

TDD se vuelve más poderoso con codificación agéntica:

1. Pide a Claude que escriba tests basados en pares entrada/salida esperados. Sé explícito sobre TDD para evitar implementaciones simuladas.
2. Dile que ejecute los tests y confirme que fallan. Dile explícitamente que **no escriba código de implementación**.
3. Pide que haga commit de las pruebas.
4. Pide que escriba código que pase los tests, **sin modificar los tests**. Dile que continúe hasta que pasen todos.
5. Pide verificación con subagentes de que la implementación no se sobreajusta a los tests.
6. Pide que haga commit del código.

c. Escribe código, captura de pantalla, itera

Para objetivos visuales:

1. Dale a Claude forma de tomar capturas de pantalla (servidor MCP de Puppeteer, simulador iOS, o copiar/pegar manualmente)
2. Dale un mockup visual (copiar/pegar, arrastrar, o ruta del archivo de imagen)
3. Pide que implemente el diseño, tome capturas de pantalla e itere hasta que coincida
4. Pide que haga commit cuando estés satisfecho

>

> La primera versión puede ser buena, pero después de 2-3 iteraciones normalmente se ve mucho mejor.

d. Modo YOLO seguro

Usa `claude --dangerously-skip-permissions` para eludir todas las comprobaciones de permisos. Funciona bien para corregir errores de linting o generar código repetitivo.

> **ADVERTENCIA:** Riesgoso. Puede provocar pérdida de datos o exfiltración. Usar en contenedor sin acceso a internet. Ver implementación de referencia con Docker Dev Containers.

e. Preguntas y respuestas del código base

Al incorporarte a un nuevo proyecto, usa Claude para explorar. Haz las mismas preguntas que le harías a otro ingeniero:

- ¿Cómo funciona el logging?
- ¿Cómo creo un nuevo endpoint de API?
- ¿Qué hace `async move { ... }` en la línea 134 de foo.rs?
- ¿Qué casos extremos maneja `CustomerOnboardingFlowImpl`?
- ¿Por qué llamamos `foo()` en lugar de `bar()` en la línea 333?
- ¿Cuál es el equivalente de la línea 334 de baz.py en Java?

f. Usa Claude para interactuar con git

Claude puede manejar eficazmente operaciones git. Muchos ingenieros de Anthropic lo usan para el 90%+ de sus interacciones con git:

- Buscar historial para responder preguntas
- Escribir mensajes de commit (analiza cambios e historial reciente)
- Manejar operaciones complejas (revertir, resolver conflictos de rebase, comparar parches)

g. Usa Claude para interactuar con GitHub

- Crear PRs (Claude entiende la abreviatura "pr")
- Implementar resoluciones para comentarios de revisión de código
- Corregir builds fallidos o advertencias de linter
- Categorizar y clasificar issues abiertos

h. Usa Claude para trabajar con Jupyter Notebooks

Claude puede leer y escribir notebooks, interpretar resultados incluyendo imágenes. Recomendamos tener Claude Code y el archivo `.ipynb` abiertos lado a lado en VS Code.

> **Tip:** Pide a Claude que haga el notebook o sus visualizaciones "estéticamente agradables" antes de mostrarlo a colegas.

---

4. Optimiza tu flujo de trabajo

a. Sé específico en tus instrucciones

La tasa de éxito mejora significativamente con instrucciones más específicas:

❌ Pobre	✅ Bueno
Añade tests para foo.py	Escribe un nuevo caso de prueba para foo.py, cubriendo el caso extremo donde el usuario cierra sesión. Evita mocks
¿Por qué ExecutionFactory tiene una API tan extraña?	Revisa el historial de Git de ExecutionFactory y resume cómo surgió su API
Añade un widget de calendario	Observa cómo se implementan los widgets existentes en la página de inicio. HotDogWidget.php es buen ejemplo. Sigue el patrón para implementar un nuevo widget de calendario que permita seleccionar mes y paginar año. Construye desde cero sin bibliotecas adicionales

b. Dale imágenes a Claude

Claude sobresale con imágenes mediante varios métodos:

- **Pegar capturas de pantalla:** `Cmd+Ctrl+Shift+4` en macOS para captura al portapapeles, `Ctrl+V` para pegar
- **Arrastrar y soltar** imágenes directamente
- **Proporcionar rutas** de archivos de imagen

Útil para mockups de diseño, gráficos visuales para análisis y depuración.

c. Menciona archivos específicos

Usa autocompletado con Tab para referenciar rápidamente archivos o carpetas.

d. Proporciona URLs a Claude

Pega URLs específicas junto con tus indicaciones. Usa `/permissions` para añadir dominios a la lista de permitidos.

e. Corrige el rumbo pronto y frecuentemente

Aunque el modo auto-accept (`Shift+Tab`) permite autonomía, obtendrás mejores resultados siendo colaborador activo.

Cuatro herramientas para corregir el rumbo:

1. **Pide un plan antes de codificar.** Dile explícitamente que no codifique hasta que confirmes que el plan es bueno.
2. **Presiona Escape** para interrumpir durante cualquier fase, preservando contexto para redirigir.
3. **Doble Escape** para volver al historial y editar un mensaje anterior.
4. **Pide deshacer cambios**, a menudo junto con #2 para un enfoque diferente.

f. Usa `/clear` para mantener el contexto enfocado

Durante sesiones largas, la ventana de contexto puede llenarse de contenido irrelevante. Usa `/clear` frecuentemente entre tareas.

g. Usa listas de verificación para flujos complejos

Para tareas grandes (migraciones, corrección de muchos errores de linting), haz que Claude use un archivo Markdown como lista de verificación:

1. Dile que ejecute el comando y escriba todos los errores en una lista Markdown
2. Encárgale abordar cada issue uno por uno, marcándolo antes de pasar al siguiente

h. Pasa datos a Claude

Varios métodos:

- **Copiar y pegar** directamente (más común)
- **Pipe al CLI:** `cat foo.txt | claude` (útil para logs, CSV, datos grandes)
- **Extraer vía comandos bash**, herramientas MCP o comandos de barra
- **Pedir que lea archivos** o busque URLs

---

5. Usa el modo headless para automatización

Claude Code incluye modo headless para contextos no interactivos: CI, hooks pre-commit, scripts de build, automatización.

```
claude -p "tu mensaje" --output-format stream-json
```

> **Nota:** El modo headless no persiste entre sesiones.

a. Usa Claude para clasificación de issues

El modo headless puede impulsar automatizaciones activadas por eventos de GitHub. El repositorio público de Claude Code usa Claude para inspeccionar nuevos issues y asignar etiquetas.

b. Usa Claude como linter

Claude puede proporcionar revisiones subjetivas de código más allá del linting tradicional: errores tipográficos, comentarios obsoletos, nombres engañosos, etc.

---

6. Flujos de trabajo multi-Claude

Algunas de las aplicaciones más potentes implican ejecutar múltiples instancias en paralelo.

a. Un Claude escribe código; otro verifica

1. Usa Claude para escribir código
2. Ejecuta `/clear` o inicia segundo Claude en otra terminal
3. El segundo Claude revisa el trabajo del primero
4. Inicia otro Claude para leer código y comentarios de revisión
5. Este Claude edita basándose en los comentarios

Puedes hacer que tus instancias se comuniquen mediante archivos compartidos.

b. Realiza múltiples checkouts del repositorio

1. Crea 3-4 checkouts con Git en carpetas separadas
2. Abre cada carpeta en pestañas de terminal separadas
3. Inicia Claude en cada carpeta con diferentes tareas
4. Cicla para comprobar progreso y aprobar/denegar solicitudes

c. Usa Git worktrees

Alternativa más liviana a múltiples checkouts:

```
# Crear worktree
git worktree add ../project-feature-a feature-a

# Iniciar Claude en cada worktree
cd ../project-feature-a && claude

# Limpiar al terminar
git worktree remove ../project-feature-a
```

Tips:

- Convenciones de nomenclatura consistentes
- Una pestaña de terminal por worktree
- En iTerm2, configurar notificaciones cuando Claude necesite atención
- Ventanas IDE separadas para diferentes worktrees

d. Usa modo headless con un arnés personalizado

`claude -p` integra Claude Code programáticamente en flujos de trabajo más grandes.

Patrón 1: Migraciones grandes o análisis

1. Pide a Claude que escriba un script para generar lista de tareas (ej. 2000 archivos a migrar de React a Vue)
2. Recorre las tareas, llamando a Claude programáticamente:

```
claude -p "migra foo.py de React a Vue. Al terminar, DEBES devolver OK si tuviste éxito, o FAIL si la tarea falló." --allowedTools Edit Bash(git commit:*)
```

3. Ejecuta varias veces y refina tu prompt

Patrón 2: Pipeline de datos

```
claude -p "<tu prompt>" --json | tu_comando_siguiente
```

> **Tip:** Usa `--verbose` para depurar. Desactívalo en producción.

Resumen de comandos útiles

Comando/Atajo	Descripción
`/init`	Genera CLAUDE.md automáticamente
`/permissions`	Gestiona lista de herramientas permitidas
`/clear`	Limpia ventana de contexto
`#`	Añade instrucción a CLAUDE.md
`Escape`	Interrumpe a Claude preservando contexto
`Escape` x2	Vuelve al historial para editar
`Shift+Tab`	Toggle modo auto-accept
`--dangerously-skip-permissions`	Modo YOLO (usar con precaución)
`-p "mensaje"`	Modo headless
`--output-format stream-json`	Salida JSON en streaming
`--mcp-debug`	Depurar configuración MCP
`--verbose`	Modo verbose para depuración
`--allowedTools`	Permisos específicos de sesión

Créditos

Escrito por Boris Cherny. Basado en las mejores prácticas de la comunidad de usuarios de Claude Code.

Agradecimientos especiales a: Daisy Hollman, Ashwin Bhat, Cat Wu, Sid Bidasaria, Cal Rueb, Nodir Turakulov, Barry Zhang, Drew Hodun y muchos otros ingenieros de Anthropic.

Documento formateado para lectura y relectura. Versión: Enero 2025

Revision #3
Created 2026-01-18 05:17:46 UTC by Abkrim
Updated 2026-01-18 05:32:08 UTC by Abkrim