Inteligencia Artificial

En los últimos años hay un montón de cambios. En estas cosas, como siempre desde que empece en informática, y mas en los últimos años hay un montón de vende humos, vende cursos. Al final, lo mejor es la práctica, y la lectura de gente que de verdad aporta algo.

- Tutorial n8n 2026 - Tom Crawshaw
- Claude Code: Mejores Prácticas

Tutorial n8n 2026 - Tom Crawshaw

Resumen Ejecutivo

Autor: Tom Crawshaw (@tomcrawshaw01)
Contexto: 8+ años en automatizaciones, $25M en ingresos de clientes con herramientas como Klaviyo, Zapier, GoHighLevel. Últimos 2 años centrado en n8n.
Público objetivo: Principiantes y usuarios intermedios que quieren resultados rápidos.

Valoración rápida (para decidir si seguir leyendo)

Aspecto	Puntuación	Comentario
Utilidad práctica	8/10	Enfoque 80/20 bien ejecutado
Profundidad técnica	5/10	Superficie, no profundiza en edge cases
Aplicabilidad sysadmin	7/10	Muchos casos de uso válidos para infraestructura
Curva de aprendizaje	Baja	Diseñado para resultados rápidos
Coste de entrada	Bajo	Desde ~6€/mes self-hosted

Conclusión adelantada: Vale la pena dedicarle 2-3 horas iniciales para evaluar si encaja en tu stack. La integración con Claude (MCP) es el punto más interesante para ese caso.

---

1. Qué es n8n y por qué considerarlo

Definición

n8n es una plataforma de automatización de flujos de trabajo visual. Cuando ocurre algo en una aplicación, n8n puede ejecutar acciones automáticas en otras aplicaciones.

Diferencia clave con Claude Code (según el autor)

> "Claude Code te ayuda a **construir** cosas. n8n **ejecuta** cosas, 24/7, sin que lo toques."

Esta distinción es importante:

- **Claude Code**: Herramienta de desarrollo, requiere supervisión
- **n8n**: Motor de ejecución autónomo, funciona desatendido

Ventajas argumentadas

1. **Canvas visual**: Ves el flujo de datos nodo a nodo. Útil para depuración y comprensión.
2. **Propiedad de datos**: Self-hosted = tus workflows, tu servidor, tu control.
3. **400+ integraciones**: Gmail, Slack, Notion, Airtable, OpenAI, APIs arbitrarias.
4. **IA integrada**: Conexión directa con GPT, Claude, LLMs locales.
5. **Coste**: Self-hosted desde ~6€/mes. Cloud más barato que Zapier a escala.

---

2. Metodología propuesta: Planificar antes de construir

Las 8 preguntas previas (traducción)

Antes de tocar n8n, responder:

1. **¿Qué lo dispara?** ¿Un email? ¿Un formulario? ¿Una hora específica?
2. **¿Cuáles son las entradas?** ¿Qué datos necesitas? ¿De dónde vienen?
3. **¿Quién está involucrado?** ¿El proceso pasa entre personas o departamentos?
4. **¿Qué herramientas se usan?** CRM, email, hojas de cálculo, Slack...
5. **¿Qué ocurre en cada etapa?** Cada paso, cada decisión, cada transformación.
6. **¿Cuál es el resultado ideal?** ¿Email enviado? ¿Registro actualizado? ¿Notificación?
7. **¿Qué puede fallar?** Puntos de fallo, datos faltantes, timeouts de API.
8. **¿Cuánto tiempo consume manualmente?** Para priorizar qué automatizar.

Criterios para decidir si automatizar

Pregunta	Si la respuesta es...	Entonces...
¿Es repetitivo?	Una vez al año	No automatizar
¿Es repetitivo?	50 veces al día	Automatizar ya
¿Sigue un proceso definido?	Sí, pasos predecibles	Buen candidato
¿Requiere iteración humana?	Sí, ida y vuelta	Diferente enfoque
¿El proceso es estable?	Cambia cada semana	Estabilizar primero
¿Coste de fallo?	Bajo (ej: redes sociales)	Experimentar
¿Coste de fallo?	Alto (ej: facturación)	Construir con cuidado

---

3. Opciones de despliegue

Opción 1: n8n Cloud

- **URL**: n8n.cloud
- **Precio**: ~20€/mes plan Starter
- **Ventaja**: Funcionando en 10 minutos
- **Recomendado para**: Aprender sin complicaciones de servidor

Opción 2: Self-Hosted

- **Coste**: Desde ~6€/mes en VPS (ej: Hostinger)
- **Instalación**: One-click, sin terminal, sin Docker manual
- **Ventajas**: Sin límites de ejecución, control total, propiedad de datos

Nota para tu caso: Con un Mac Studio M4, podrías correr n8n localmente para desarrollo y pruebas, y tener una instancia en VPS para producción 24/7.

---

4. Integración con Claude (MCP Server)

Esto es lo más relevante para tu situación

Existe un servidor MCP (Model Context Protocol) que conecta Claude con n8n:

- Acceso a los 537+ nodos
- Documentación de cada nodo
- Requisitos de configuración

Flujo de trabajo propuesto:

1. Describes lo que quieres construir
2. Claude genera el primer borrador del workflow
3. Tú lo refinas en n8n

Recursos citados

- **Video setup MCP**: https://youtu.be/O_yyQAIU-zU
- **GitHub MCP Server**: https://github.com/czlonkowski/n8n-mcp
- **GitHub n8n Skills para Claude**: https://github.com/czlonkowski/n8n-skills
- **Post X sobre setup**: https://x.com/tomcrawshaw01/status/2008538584085594458

---

5. Los 12 nodos esenciales (el 20% que hace el 80%)

Nodos de Disparo (Triggers)

Nodo	Función	Caso de uso
Manual Trigger	Click para ejecutar	Testing y desarrollo
Webhook Trigger	URL que recibe datos	Formularios, pagos, integraciones
Schedule Trigger	Ejecución programada	Reportes diarios, backups semanales
Form Trigger	Formularios nativos	Sin herramienta externa de forms

Nodos de Lógica

Nodo	Función	Caso de uso
IF	Bifurcación condicional	Si amount > 1000, hacer X
Switch	Múltiples caminos	Enrutar según tipo de ticket
Filter	Filtrar datos	Eliminar registros inválidos

Nodos de Datos

Nodo	Función	Caso de uso
Edit Fields (Set)	Crear/modificar campos	Transformar estructura de datos
Split Out	Array → items individuales	Procesar listas
Merge	Combinar ramas	Reunir datos de múltiples fuentes

Nodos de Integración

Nodo	Función	Caso de uso
HTTP Request	Llamadas a cualquier API	Cuando no hay nodo específico
Gmail/Slack/Sheets	Integraciones comunes	Los "big three"

Nodos de IA

Nodo	Función	Caso de uso
Basic LLM Chain	Prompt → Respuesta	Integración simple de IA
AI Agent	IA con herramientas	Decisiones complejas

---

6. Funcionalidades ocultas que ahorran tiempo

6.1 Pin Data (Fijar datos)

- **Cómo**: Click derecho en nodo → Pin data
- **Efecto**: Bloquea la salida para testear nodos posteriores sin re-ejecutar todo
- **Utilidad**: Crítico para depuración

6.2 Cargar ejecuciones anteriores

- **Cómo**: Pestaña Executions → Seleccionar ejecución pasada
- **Efecto**: Cargar datos reales de producción en el canvas
- **Utilidad**: Testear con datos reales sin disparar el workflow

6.3 URLs de Webhook: Test vs Producción

- **URL de Test**: Solo funciona con "Test Workflow"
- **URL de Producción**: Solo funciona con workflow activo
- **Trampa común**: Confundir cuál usar en cada contexto

6.4 Editor de expresiones

- **Cómo**: Click en el icono de engranaje en campos de input
- **Efecto**: Editor completo con autocompletado
- **Consejo**: No escribir expresiones complejas en el input pequeño

6.5 Nombrar nodos

- **Malo**: "HTTP Request 45"
- **Bueno**: "Fetch_Customer_Data"
- **Por qué**: A las 2am cuando algo falla, agradecerás nombres descriptivos

---

7. Errores comunes a evitar

Error 1: Construir antes de planificar

Síntoma: 3 horas después tienes un espagueti que no funciona.
Solución: 5 minutos de esquema en papel ahorran 5 horas de debugging.

Error 2: Ignorar mensajes de error

Ejemplo: `Cannot read property 'email' of undefined`
Significado: Intentas acceder a un campo que no existe.
Solución: Leer el error, mirar los datos, la respuesta suele estar ahí.

Error 3: No usar plantillas

Dato: n8n tiene 1000+ plantillas gratuitas.
Método: Copiar → Modificar → Desplegar → Cobrar.

Error 4: Sobre-ingeniería prematura

Trampa: Añadir manejo de errores, reintentos, notificaciones... antes de que funcione.
Método: Primero que funcione feo. Luego mejorar.

> "Mi workflow más bonito tardó 2 semanas y nunca se desplegó. El más feo generó $5K."

Error 5: Saltarse los básicos de JSON

Mínimo necesario:

```
{
  "name": "John",
  "email": "john@example.com"
}
```

- `name` y `email` son etiquetas
- `"John"` y `"john@example.com"` son valores
- Para acceder en n8n: `{{ $json.email }}`

---

8. Ruta de aprendizaje propuesta

Semana 1: Básicos (5-10 horas)

Construir estos 3 workflows:

1. Webhook → Mensaje de Slack
2. Envío de formulario → Notificación por email
3. Schedule → Actualización de Google Sheets

Objetivo: Que los datos fluyan de A a B. Sin complejidad.

Semana 2: Añadir lógica (5-10 horas)

Tomar los workflows de Semana 1 y añadir:

- Nodos IF (enrutar según condiciones)
- Nodos Filter (solo procesar lo relevante)
- Edit Fields (transformar datos antes de enviar)

Semana 3: Integraciones API (10-15 horas)

- Usar HTTP Request para APIs externas
- Parsear respuestas JSON
- Manejar autenticación (API keys, OAuth)

Ejercicio: Extraer datos de un sistema, transformarlos, enviarlos a otro.

Semana 4: Integración IA (5-10 horas)

- Basic LLM Chain para prompts simples
- AI Agent para tareas complejas
- Procesar y enrutar datos según clasificación de IA

Meses 2-3: Patrones de producción

- Sub-workflows para reutilización
- Manejo de errores con notificaciones
- Lógica de reintentos para APIs inestables
- Optimización de velocidad y coste

---

9. Debugging: El método IA

Cuando te atasques:

1. Copiar el mensaje de error
2. Copiar los datos JSON que lo causaron
3. Exportar el workflow como JSON
4. Pegar todo en Claude
5. Preguntar: "¿Qué está mal y cómo lo arreglo?"

Ejemplo:

```
ERROR: "Cannot read property 'email' of undefined"

JSON DATA:
{
  "user_email": "john@example.com",
  "name": "John Smith"
}
```

Respuesta de Claude: "Intentas acceder a 'email' pero tus datos tienen 'user_email'. Cambia `{{ $json.email }}` por `{{ $json.user_email }}`"

Recursos de comunidad

- **Foro n8n**: community.n8n.io (buscable, mayoría de preguntas ya respondidas)
- **Plantillas**: n8n.io/workflows (1000+ ejemplos)
- **Reddit**: r/n8n

---

10. Análisis crítico: Fortalezas y debilidades del artículo

Lo que hace bien

1. **Enfoque 80/20**: Reduce 400+ nodos a los 12 esenciales. Pragmático.
2. **Metodología antes que herramienta**: Las 8 preguntas previas son aplicables a cualquier automatización.
3. **Integración Claude-n8n**: El MCP server es un multiplicador de productividad real.
4. **Ruta de aprendizaje estructurada**: Progresión lógica de 4 semanas.
5. **Errores comunes documentados**: Ahorra tiempo de aprendizaje por prueba y error.

Lo que falta o simplifica

1. **Seguridad**: No menciona:
   - Almacenamiento de credenciales
   - Exposición de webhooks
   - Rate limiting
   - Auditoría de accesos
2. **Alta disponibilidad**: ¿Qué pasa si el servidor n8n cae? No hay mención de:
   - Redundancia
   - Backups de workflows
   - Recuperación ante desastres
3. **Escalabilidad**: "Self-hosted desde 6€/mes" asume cargas ligeras. No discute:
   - Límites de ejecuciones concurrentes
   - Optimización de recursos
   - Cuándo escalar vertical vs horizontal
4. **Casos de error complejos**: El debugging con Claude es útil para errores simples. No cubre:
   - Condiciones de carrera
   - Datos corruptos

- Fallos parciales en workflows largos

5. **Integración con infraestructura existente**: No menciona:
   - Cómo integrar con sistemas de monitorización (ej: Zabbix)
   - Logs centralizados
   - Integración con CI/CD

---

11. Aplicabilidad a tu contexto

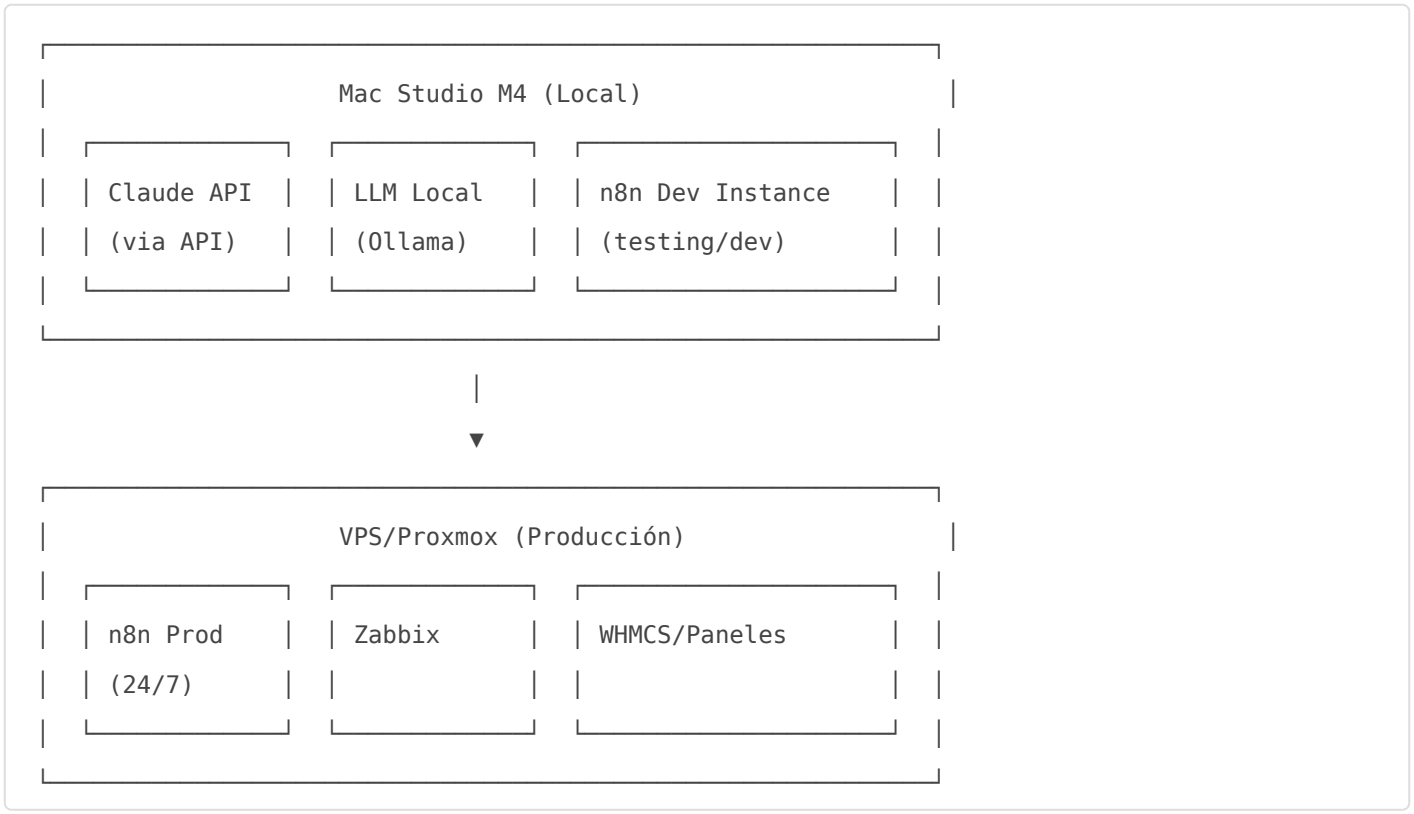
Tu situación

- Usuario de Claude
- Mac Studio M4 disponible para IA local
- Interés en automatización
- No quieres perder tiempo
- 30 años de experiencia sysadmin
- Infraestructura existente (Proxmox, cPanel, DirectAdmin, WHMCS, Zabbix)

Casos de uso potenciales para tu infraestructura

Caso	Trigger	Proceso	Beneficio
Alertas Zabbix enriquecidas	Webhook de Zabbix	n8n consulta logs, usa Claude para análisis, notifica Slack con contexto	Alertas más accionables
Gestión de tickets WHMCS	Nuevo ticket	Clasificación con IA, enrutamiento automático, respuesta inicial	Reducir tiempo de primera respuesta
Monitorización de certificados	Schedule diario	Revisar expiración SSL de dominios, alertar si < 30 días	Prevenir expiraciones
Backups verificados	Post-backup hook	Verificar integridad, notificar resultado, escalar si falla	Confianza en backups
Sincronización DNS	Cambio en panel	Propagar cambios a otros sistemas, verificar propagación	Consistencia multi-panel
Análisis de logs	Schedule o webhook	Extraer patrones con IA, generar resumen diario	Visibilidad sin esfuerzo

Arquitectura sugerida para tu caso



Integración con tu stack de monitorización (Zabbix)

n8n puede complementar Zabbix:

- **Zabbix**: Detección y alertas base
- **n8n**: Enriquecimiento, decisiones complejas, acciones multi-sistema

Ejemplo de workflow:

```
Zabbix Alert (webhook)
    ↓
n8n recibe alerta
    ↓
Consulta contexto adicional (logs, métricas relacionadas)
    ↓
Envía a Claude para análisis
    ↓
IF análisis indica problema conocido
    → Ejecuta runbook automático
    → Notifica con resolución
ELSE
```

```
    → Escala a humano con contexto enriquecido
```

---

12. Recomendación de aproximación

Fase 0: Evaluación (2-3 horas)

1. Instalar n8n localmente en Mac Studio (Docker o nativo)
2. Configurar MCP server para Claude
3. Construir un workflow trivial: webhook → log → notificación
4. Evaluar si la interfaz y el modelo mental encajan contigo

Fase 1: Caso de uso real simple (1 semana)

Elegir UN caso de uso de bajo riesgo:

- Sugerencia: Notificación enriquecida de alertas Zabbix
- Por qué: No modifica nada, solo observa y notifica

Fase 2: Iterar o descartar

Si Fase 1 demuestra valor → expandir gradualmente Si no → el coste ha sido bajo, lección aprendida

Lo que NO hacer

- No intentar automatizar todo de golpe
- No migrar procesos críticos antes de dominar la herramienta
- No asumir que "visual" = "simple" (los edge cases siguen existiendo)

---

13. Recursos adicionales

Del artículo original

- Masterclass 50 minutos: enlace en artículo original
- Newsletter "AI Operator's Playbook": 15 workflows listos para usar, playbooks de implementación

Documentación oficial

- **n8n Docs**: docs.n8n.io
- **n8n Community**: community.n8n.io
- **Templates**: n8n.io/workflows

Para integración con Claude

- **MCP Server n8n**: github.com/czlonkowski/n8n-mcp
- **n8n Skills**: github.com/czlonkowski/n8n-skills

Para tu stack específico

- **Zabbix webhooks**: documentación de media types webhook
- **WHMCS API**: developers.whmcs.com
- **cPanel API**: api.docs.cpanel.net

---

14. Checklist de decisión

Antes de invertir tiempo significativo, validar:

- ☐ ¿Tengo al menos 3 procesos repetitivos que consumen >30 min/semana?
- ☐ ¿Esos procesos son estables (no cambian cada semana)?
- ☐ ¿Tengo acceso API a los sistemas involucrados?
- ☐ ¿El coste de fallo es manejable mientras aprendo?
- ☐ ¿Tengo 5-10 horas en las próximas 2 semanas para experimentar?

Si ≥4 respuestas son "sí" → vale la pena probar n8n Si <4 → quizás mejor esperar o buscar alternativa

---

Notas para ampliación futura

Esta sección queda reservada para añadir información adicional según se vaya explorando la herramienta.

Pendientes de investigar

- [ ] Rendimiento de n8n en Mac Studio M4 vs VPS
- [ ] Integración específica con Zabbix 7.4
- [ ] Workflows para gestión de WHMCS
- [ ] Comparativa n8n vs alternativas (Make, Zapier, Activepieces)
- [ ] Backup y versionado de workflows

Workflows implementados

(Añadir según se vayan creando)

Lecciones aprendidas

(Documentar problemas encontrados y soluciones)

Documento generado: Enero 2026
Basado en: Tutorial n8n 2026 de Tom Crawshaw
Contexto: Evaluación para infraestructura de hosting y automatización sysadmin

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