Taller de Claude Code

Web Reactiva Premium

Descubriendo el asistente de programación con IA
Anthropic junto a la Comunidad Malandriner

Lo básico

🤖 Asistentes agénticos en terminal

- Más rápidos al carecer de interfaz
- Independientes del IDE
- Mejor experiencia de desarrollo
- Muchos son Open Source
- Flexibles
- Pensados para devs

Alternativas a Claude Code

- **Aider** → https://aider.chat/
  - El original
- **Gemini CLI** → https://github.com/google-gemini/gemini-cli
  - Tier gratuito y Acceso con cuenta de Google o API
- **Codex CLI** → https://github.com/openai/codex
  - Acceso con cuenta de ChatGPT
- **Qwen Code** → https://github.com/QwenLM/qwen-code
- **OpenCode** → https://opencode.ai/
- **Claude Code Router** → https://github.com/musistudio/claude-code-router

¿Qué es Claude Code?

- De los creadores de Claude Sonnet: Anthropic
- Herramienta de línea de comandos para programación agéntica
- Diseño bajo nivel y sin dogmas
- Acceso directo al modelo sin forzar workflows específicos
- Flexible y personalizable
- Doc en español: https://docs.claude.com/es/docs/claude-code/overview

💰 Precios de Claude Code

- **Plan Pro** ($20/mes) → Recomendado para desarrollo
- **Plan Max** ($100-200/mes) → Para uso intensivo
- **API** (pay-per-use) → Si no quieres cuenta de Claude

👉 Nota: Opus 4 usa 5x créditos que Sonnet 4

👉 Reset del límite: Cada 5 horas

📦 Instalación de Claude Code

npm (recomendado):

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

Homebrew (macOS):

```
brew install claude-code
```

Verificar instalación:

```
claude --version
claude -h
```

🚀 Arrancando

Comando básico para iniciar:

```
claude
```

Creando el fichero CLAUDE.md

```
/init
```

👀 CLAUDE.md

- Archivo brújula que resume el proyecto
- **Raíz del repo** → `CLAUDE.md` (recomendado)
- **Home folder** → `~/.claude/CLAUDE.md` (global)
- Soporte para anidar carpetas padres e hijas

Importante: Debes mantenerlo actualizado.

⌨ Atajos de teclado esenciales

Presiona `?` en Claude Code para ver todos los atajos:

```
! for bash mode          double tap esc to clear input      ctrl + _ to undo
/ for commands           shift + tab to auto-accept edits   ctrl + z to suspend
@ for file paths         ctrl + o for verbose output        ctrl + v to paste images
# to memorize            ctrl + t to show todos             shift + ⏎ for newline
```

Tip: Memoriza `shift + tab` y `double esc` - los usarás constantemente

🔄 Modos de operación

- Cambiar entre modos con `shift + tab`:

1. Default
2. Plan mode
3. Accept edits
4. ¡Peligro!: bypassPermissions
5. ¡¡Más peligro!!: YOLO

```
claude  --permission-mode <mode>
Permission mode to use for the session
(choices: "acceptEdits",
"bypassPermissions", "default", "plan")
```

🚨 Modo YOLO

¡¡No lo hagas nunca!!

Establecer permisos totales y luego lanzarse a la ejecución sin control.

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

```
claude --continue --print "[YOUR PROMPT]"
--dangerously-skip-permissions
--verbose --output-format stream-json | jq
```

⚙ Permisos sobre herramientas

- `/permissions` para gestionar herramientas permitidas
- El archivo `.claude/settings.json` almacena los permisos:

```
{
  "allowedTools": [
    "Edit",
    "Bash(git commit:*)",
    "Bash(git push:*)",
    "mcp__puppeteer__puppeteer_navigate",
  ],
  "blockedTools": [
    "Bash(rm:*)",
    "Bash(sudo:*)"
  ]
}
```

🔦 Prevalencia del `settings.json`

1. **Políticas empresariales administradas** (`managed-settings.json`)
2. **Argumentos de línea de comandos**
3. **Configuraciones locales del proyecto** (`.claude/settings.local.json`)
4. **Configuraciones compartidas del proyecto** (`.claude/settings.json`)
5. **Configuraciones de usuario** (`~/.claude/settings.json`)

📜 Historial

Vuelve a las conversaciones pasadas

```
/resume
```

Limipia el contexto

```
/clear
```

Arranca desde la última conversación

```
claude --continue
```

🌐 Interfaces visuales alternativas

Claude Code UI https://claudecodeui.com/

- Mejor visualización de conversaciones

Claudia (opcode) https://claudia.so/**

- Dashboard y agentes

Claude Code WebUI https://github.com/sugyan/claude-code-webui

- UI open source

¿Qué diablos significa esto?

Herramientas nativas de Claude Code

⚡ Comandos slash: atajos poderosos

¿Qué son?

Prompts reutilizables almacenados como archivos Markdown

- **Personalizados de proyecto** → `.claude/commands/`
- **Personalizados de usuario** → `~/.claude/commands/`
- **Argumentos dinámicos** → `$ARGUMENTS`, `$1`, `$2`
- **Ejecución de bash** → Prefijo `!` para comandos
- **Referencias de archivos** → Prefijo `@`

Ejemplo: Optimizador

```
mkdir -p .claude/commands
echo "Analiza este código en busca de
problemas de rendimiento y sugiere
optimizaciones:" > .claude/commands/optimizar.md
```

Luego:

```
/optimizar
```

Ejemplo práctico 🎹

🔗 Hooks: automatización inteligente

¿Qué son?

Comandos shell que se ejecutan automáticamente en puntos específicos del flujo de trabajo

Algunos eventos disponibles:

- **PreToolUse** → Antes de ejecutar herramientas
- **PostToolUse** → Después de usar herramientas
- **Notification** → Cuando se necesita atención del usuario
- **Stop** → Al finalizar respuestas
- **SessionStart/End** → Inicio/fin de sesiones

Ejemplo: Log de comandos bash

```
{
  "PreToolUse": [
    {
      "matcher": "Bash",
      "hooks": [
        {
          "command": "jq -r '.tool_input.command'
          >> ~/.claude/bash-log.txt"
        }
      ]
    }
  ]
}
```

Ejemplo práctico 🎹

🤖 Agentes: especialistas personalizados

¿Qué son?

Asistentes de IA **especializados** que Claude Code puede invocar para tareas específicas

Características clave:

- **Contexto separado** → No contaminan la conversación principal
- **Herramientas específicas** → Solo acceden a lo necesario
- **Prompts personalizados** → Instrucciones especializadas
- **Reutilizables** → Se pueden usar en múltiples proyectos

Comando: `/agents` para gestionar todos tus subagentes

Ejemplo: Revisor de código

```
---
name: code-reviewer
description: Expert code review specialist.
Use immediately after writing code.
tools: Read, Grep, Bash
---

You are a senior code reviewer. When invoked:
1. Run git diff to see recent changes
2. Focus on security, performance, readability
3. Provide actionable feedback with examples

Review checklist:
- No exposed secrets
- Proper error handling
- Clear naming conventions
- Performance considerations
```

Ejemplo práctico 🎹

🎨 Estilos de salida: personalidades adaptables

¿Qué son?

Modificaciones del prompt del sistema que cambian cómo responde Claude

Estilos integrados:

- **Predeterminado** → Ingeniería de software eficiente
- **Explicativo** → Incluye insights educativos
- **Aprendizaje** → Colaborativo con TODOs para el usuario

Personalización:

- Se guardan en `~/.claude/output-styles/` o `.claude/output-styles/`

Comando: `/output-style` para seleccionar el estilo

Ejemplo: Mentor de código

```
---
name: Code Mentor
description: Teaching-focused style with detailed explanations
---

# Mentor de Código

Eres un mentor experimentado que no solo resuelve problemas,
sino que SIEMPRE explica el razonamiento detrás de cada decisión.

## Comportamientos específicos:
- Explica el "por qué" de cada elección técnica
- Sugiere alternativas y trade-offs
- Incluye recursos para profundizar
- Usa analogías para conceptos complejos

Antes de cada implementación, pregunta:
"¿Te gustaría que implemente esto paso a paso explicando cada parte?"
```

🔍 Tabla comparativa

Característica	Propósito	Contexto	Persistencia
Subagentes	Tareas especializadas	Separado	Por sesión
Estilos de salida	Personalidad/modo	Modifica principal	Configuració
Hooks	Automatización	Eventos del sistema	Permanente
Comandos slash	Prompts reutilizables	Principal	Archivos estáticos

MCP en Claude Code

🧠 MCP Server

Amplia las capacidades más allá de la IA

¿Qué son?

- Un servidor MCP actúa como intermediario entre la IA y aplicaciones externas "del mundon real"
- Locales y remotos (Se comunican con aplicaciones de IA a través de internet usando HTTP o SSE)
- Herramientas que puede invocarn la IA durante conversaciones, incluyendo lectura de datos, creación, modificación o eliminación de datos en aplicaciones conectadas

🧠 MCP Server en Claude Code

- Conecta múltiples servidores MCP
- Configuración por proyecto o global
- `.mcp.json`
- `claude mcp` o dentro de claude con `/mcp`

Sequential Thinking

- Resolver problemas de forma estructurada
- https://mcp.so/server/sequentialthinking

```
claude mcp add sequential-thinking --scope project --
npx -y @modelcontextprotocol/server-sequential-thinking
```

Context7

- Documentación a la medida
- https://context7.com

```
claude mcp add context7 --scope project --transport http
https://mcp.context7.com/mcp --header "CONTEXT7_API_KEY: YOUR_API_KEY"
```

Github

- Acceso a todas las características de Github con tu proyecto
- https://github.com/github/github-mcp-server#remote-github-mcp-server

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp
-H "Authorization: Bearer $(grep GITHUB_PAT .env | cut -d '=' -f2)"
--scope project
```

Últimos trucos

🧠⚡ Pensamiento extendido

Palabras clave

- `piensa` o `think` para que reflexiones más a fondo
- A más pensamiento más coste del presupuesto de razonamiento = más tokens
- `think` < `think hard` < `think harder` < `ultrathink`

⚡ Workflow EPCC

1. Explore
2. Plan
3. Code
4. Commit

Otro:

- TDD con Claude Code

Workflow visual: Mock → Código → Screenshot → Iterar

1. **Dale a Claude capacidad de screenshots** (Puppeteer MCP)
2. **Proporciona un mock visual** (drag & drop)
3. **Pide implementación** del diseño
4. **Itera** hasta que coincida con el mock

Resultado: Diseños mucho mejores tras 2-3 iteraciones

Trabajo por Git worktrees

- Separa múltiplas ramas en directorios diferentes

```
# Create worktrees for parallel work
git worktree add ../project-feature-a -b feature-a
git worktree add ../project-bugfix -b bugfix-123

# Run Claude Code in each directory
cd ../project-feature-a && claude
cd ../project-bugfix && claude

# Manage worktrees
git worktree list
git worktree remove ../project-feature-a
```

🔄 Workflows multi-Claude

Patrón 1: Escritor + Revisor

1. Claude escribe código
2. `/clear` o nuevo terminal
3. Segundo Claude revisa el trabajo
4. Tercer Claude edita basado en feedback

Patrón 2: Múltiples checkouts de repo

- Crear 3-4 checkouts en carpetas separadas
- Abrir cada carpeta en pestañas de terminal separadas
- Iniciar Claude en cada carpeta con tareas diferentes

📚 Recursos

- Documentación completa (EN ESPAÑOL):

  https://docs.claude.com/es/docs/claude-code/overview
- Claude Code Course (Free): https://anthropic.skilljar.com/claude-code-in-action
- Mejores prácticas: https://www.anthropic.com/engineering/claude-code-best-practices
- Awesome Claude Code: https://github.com/hesreallyhim/awesome-claude-code
- Claude Code Templates: https://www.aitmpl.com
- SuperClaude Framework: https://superclaude.netlify.app/
- ClaudeLog: https://claudelog.com

¡Gracias malandriners 🧡!