BIG school {m}

CURSO DE DESARROLLO CON IA

Programa con agentes

Índice

AI-first IDE

El paradigma AI-First prioriza herramientas donde el LLM es el núcleo de la arquitectura del editor, no un plugin añadido.

1. Contexto completo:
El modelo ve todo tu proyecto, no sólo el archivo actual.

2. Edición multi-archivo:
Cambios coordinados en múltiples ficheros a la vez.

3. Ejecución real:
Comandos, test y builds dentro del proyecto.

4. Iteración autónoma:
Detectan errores y se auto-corrigen.

EL DESARROLLADOR PASA
A SER EL DIRECTOR DEL PROCESO

OpenCode

¿Qué es?

Es un agente de código de IA de código abierto. Está disponible como interfaz basada en terminal, aplicación de escritorio o extensión IDE. Es la alternativa a Claude Code.

Características

- Open-source
- Agente de su categoría más utilizado
- Agnóstico al modelo
- Tiene una capa de uso gratis

Web Oficial: https://opencode.ai
Repositorio Oficial: https://github.com/anomalyco/opencode

Primeros pasos en OpenCode

Instalación y configuración

- **Descarga:** https://opencode.ai/es/download
- **Documentación Oficial:** https://opencode.ai/docs

Comandos y atajos más utilizados

- **opencode web** → Abre una interfaz web en local para usar en lugar de la terminal
- **/help** → Guía de referencia rápida y catálogo de funciones
- **/connect** → Selecciona los proveedores de IA a utilizar
- **/models** → Selecciona los modelos a utilizar en cada proveedor de IA
- **/init** → Analiza el proyecto y genera automáticamente el archivo agents.md
- **/agents** o **TAB** → Cambia entre modo Build y Plan
- **/status** → Ver el estado de la sesión
- **/new** → Abre un nuevo chat
- **/session** → Lista las sesiones de chat
- **exit** → Cierra la sesión actual
- **CTRL + P** → Muestra en detalle todos los comandos de OpenCode
- **CTRL + T** → Cambia el nivel de esfuerzo del modelo de IA elegido

Build vs. Plan

Build Mode

El agente actúa directamente sobre el código. Ejecución inmediata. Ideal para tareas atómicas, prototipado rápido o corrección de bugs evidentes.

Plan Mode

El agente primero elabora un plan detallado de los cambios que va a hacer, te lo muestra para que lo revises, y sólo ejecuta cuando tú das el OK. Evita cambios destructivos y permite ajustar la estrategia antes de tocar el código.

Cuándo usarlo	Cuándo NO usarlo
✓ Tareas grandes	✗ Tareas pequeñas
✓ Refactors	✗ Correcciones obvias
✓ Supervisión de estrategia	✗ Prototipado rápido

Arneses (Harness engineering)

El indeterminismo intrínseco de los LLM requiere una estructura de control profesional: el Arnés. En ingeniería de software con IA, un arnés es el conjunto de "guardarraíles" que garantiza que el agente trabaje bajo convenciones estrictas y estándares de producción.

AGENTS.md y reglas

¿Qué son?

Archivos de texto que actúan como system prompts persistentes para tu proyecto. Para evitar el ruido y la ineficiencia, el agent.md no debe exceder las 500 líneas.

El agente lo consulta antes de cada acción.

AGENT.md → Genérico	CLAUDE.md	.cursorrules	copilot-instructions.md

¿Qué incluir?

Stack tecnológico	Convenciones de código	Patrones	Prohibiciones
Estructuras de proyecto	Flujo de trabajo	Testing, CI/CD	Estilo de commits y PRs

https://agents.md

Plantilla AGENTS.md

```
# [Nombre del proyecto]
[Una o dos frases: qué es el proyecto y para quién. En lenguaje claro.]

## Stack
- Lenguaje: [p. ej. TypeScript estricto]
- Framework / runtime: [p. ej. Node 22 + Express 5]
- Base de datos: [p. ej. PostgreSQL con Prisma]
- Tests: [p. ej. Vitest]

## Comandos
- `[comando dev]`    — arranca el servidor en local
- `[comando test]`   — ejecuta los tests (deben pasar antes de cada commit)
- `[comando lint]`   — revisa el estilo (antes de cada PR)
- `[comando build]`  — compila para producción

## Estructura del proyecto
- `[carpeta]/` — [qué contiene]
- `[carpeta]/` — [qué contiene]
- `[carpeta]/` — [qué contiene]
```

```
## Convenciones
- [Estilo de nombres, p. ej. camelCase para variables y funciones.]
- [Dónde van los tests, p. ej. al lado del archivo: `foo.ts` +
`foo.test.ts`.]
- [Manejo de errores, p. ej. clases propias en `src/errors/`.]
- [Patrón a seguir, p. ej. validar toda entrada del usuario antes de
usarla.]

## No hagas
- [Límite duro, p. ej. no instalar dependencias sin avisar.]
- [Zona prohibida, p. ej. no tocar `src/legacy/`, está congelada.]
- [Regla de seguridad, p. ej. no subir archivos `.env*` al
repositorio.]
- [Antipatrón, p. ej. no usar `any` en TypeScript sin justificarlo.]

## Flujo de trabajo
- Antes de una tarea no trivial, propón un plan y espera mi OK.
- Una tarea a la vez; al terminar, dime qué cambiaste para que lo
revise.
- Si no estás seguro al 80%, pregunta. No inventes.

## Documentación
- Referencias a más reglas, contexto, documentación y especificaciones.
```

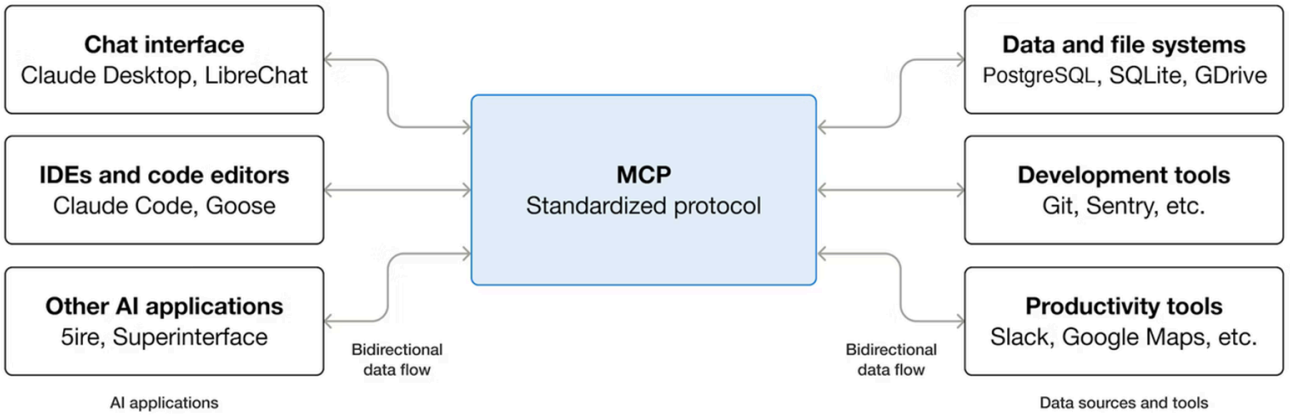
MCP (Model Context Protocol)

Cliente MCP

Interfaz estándar para conectar cualquier herramienta.

Ventajas

- ✓ Estándar abierto
- ✓ Reutilizable entre IDEs
- ✓ Seguridad por diseño
- ✓ Ecosistema creciente



https://modelcontextprotocol.io

Directorio de MCPs:

https://mcp.directory/

https://mcpservers.org/

Documentación Oficial OpenCode sobre MCP:

https://opencode.ai/docs/mcp-servers/

Context7

Context7 es un servidor MCP que proporciona a asistentes de IA acceso a documentación actualizada y ejemplos de código directamente desde las fuentes oficiales de las librerías y frameworks.
Su objetivo principal es evitar que la IA genere código basado en información obsoleta o APIs inexistentes.

¿Para qué sirve?

- Obtener documentación de la versión exacta de una librería.
- Generar código compatible con APIs actuales.
- Reducir las "alucinaciones" de los LLMs al programar.
- Consultar ejemplos oficiales sin salir del editor.
- Mejorar la calidad de herramientas como Cursor, Claude Code, Windsurf o VS Code con MCP.

Ventajas

- **Documentación actualizada**: No depende únicamente del entrenamiento del modelo.
- **Información específica por versión**: Evita ejemplos de versiones antiguas.
- **Menos APIs inventadas**: Reduce errores causados por alucinaciones.
- **Integración con IA**: Funciona directamente dentro del flujo de trabajo del asistente.
- **Fuentes oficiales**: Obtiene información desde documentación real y mantenida.

Web oficial: https://context7.com/
Creación API KEY: https://context7.com/dashboard
Instalación en OpenCode:
https://context7.com/docs/resources/all-clients#opencode

Prompts utilizados en el ejemplo del video:

Crear el MCP de Context7:

```
Crea la configuración de un MCP para Context7 a nivel de proyecto
```

Usar el MCP:

```
Crea una web que muestre una línea del tiempo desde la aparición de
ChatGPT a la actualidad, donde aparezcan predicciones realizadas por
CEO y figuras relevantes de la tecnología que hablen sobre cómo los
programadores iban a desaparecer y cómo se equivocaron ya que sólo
buscaban favorecer la expectativa por sus empresas. Utiliza Next.js y
context7.
```

Skills: Conocimiento para agentes

Son instrucciones o mejores prácticas a seguir por el agente.
Se alojan en **.opencode/skills**.

Cliente MCP

- Acceso a herramientas
- Conexión en tiempo real
- Acciones sobre servicios externos
- API estándar (JSON-RPC)

Skills

- Instrucciones/mejores prácticas
- El agente las lee antes de actuar
- Archivos SKILL.md + scripts
- No siempre se usa

MCPs: Servicios externos
Skills: Conocimiento interno

Utilizando plataformas como skills.sh, podemos dotar al agente de habilidades ultra-específicas de forma modular. Esto mantiene el contexto principal limpio.

Por ejemplo, cargando el skill “Frontend Design Expert”, nos aseguramos que el agente solo cargue las reglas de diseño React cuando realmente está trabajando en un componente de frontend, optimizando así el uso de tokens y la precisión de la respuesta.

Carga:

```
npx skills add https://github.com/anthropics/skills --skill
frontend-design
```

Uso:

```
Mejora la web teniendo en cuenta las buenas prácticas de
/frontend-design
```

Documentación Oficial OpenCode sobre Skills: https://opencode.ai/docs/skills/

Comandos

Los comandos en herramientas como OpenCode y Claude Code son acciones específicas que el desarrollador invoca manualmente para interactuar con el agente y configurar el entorno de trabajo.

A diferencia de las habilidades (skills), que son conocimientos que el agente consulta de forma autónoma, los comandos solo se ejecutan cuando el usuario los solicita mediante el uso de la barra inclinada (/).

Tipos de comandos

Integrados

- /init
- /new
- /help
- …

Personalizados

- Atajo con /nombre
- Son archivos .md
- Estructura con descripción y pasos

Skills: Conocimiento interno (se invoca solo)
Comandos: Acciones repetitivas (lo invocas)

Los comandos personalizados se guardan en el directorio **.opencode/commands/** en formato Markdown (.md).

La estructura de un comando personalizado incluye una descripción, el tipo de agente que debe usar (habitualmente el modo build) y la instrucción específica que debe ejecutar.

Son ideales para acciones repetitivas que deben seguir un paso a paso estricto definido por el desarrollador.

Documentación Oficial OpenCode sobre Commands:
https://opencode.ai/docs/commands/

Memoria y verificación

La gestión de la memoria y la verificación es fundamental para pasar de un uso descontrolado de la inteligencia artificial a un desarrollo de software profesional y estructurado.

Gestión de la Memoria y Contexto

No se debe permitir que el agente consuma tokens sin criterio; en su lugar, se deben aplicar buenas prácticas de higiene del contexto:

- **Documentación de tareas**: Una técnica recomendada es crear una carpeta de documentos (por ejemplo, /docs) donde el agente guarde notas de la sesión y listas de tareas realizadas. Esto permite "particionar el cerebro" del agente, facilitando que este recupere el contexto de una tarea específica semanas después sin saturar la ventana de tokens.

- Comando **/compact**: Este comando resume el historial crítico, reduciendo el porcentaje de uso de tokens sin perder la información esencial para continuar trabajando y así evitar que la ventana de contexto se llene. Esto es preferible al comando **/new**, ya que preserva la continuidad del razonamiento agéntico.

Bucle de Verificación

La verificación se presenta como un proceso iterativo donde el desarrollador mantiene el control absoluto:

- Ciclo de trabajo: El proceso consiste en un bucle de escribir, testear y corregir. Es responsabilidad del desarrollador asegurarse de que el agente sea capaz de probar lo que construye.

- Rol del desarrollador: El humano debe actuar como el director del proceso, supervisando que el agente se autocorrija y valide sus resultados hasta que encajen con los requisitos reales del proyecto.

En resumen, la memoria se gestiona mediante la segmentación de documentos y la compactación de sesiones, mientras que la verificación se logra a través de un bucle continuo de pruebas y supervisión humana.

Súmate a nuestro directo del día 3

IR AL DIRECTO