Cuando hablamos de usar Codex, lo normal es pensar en abrirlo desde la terminal, desde una extensión del editor o desde la interfaz oficial. Pero Codex también se puede integrar de formas menos directas en nuestros propios flujos de trabajo. Esto no significa que tengamos un modelo local funcionando como si fuera Ollama, llama.cpp o vLLM. Codex puede ejecutarse en nuestra máquina, leer nuestro proyecto, modificar archivos y lanzar comandos, pero la inteligencia del modelo sigue dependiendo del servicio de OpenAI. Lo que sí podemos hacer es utilizar Codex como pieza intermedia dentro de scripts, automatizaciones, servidores locales, clientes personalizados o herramientas conectadas.

Estas son cinco formas indirectas de usar Codex.

1. Usar Codex desde la terminal con Codex CLI

La forma más básica de usar Codex fuera de una interfaz gráfica es mediante **Codex CLI**. En este caso ejecutamos Codex directamente desde la terminal, dentro del directorio de un proyecto, y le pedimos que analice código, proponga cambios, ejecute pruebas o modifique archivos.

```
codex
```

Esta forma no convierte a Codex en una API, pero sí nos permite usarlo de forma nativa en Debian o en cualquier entorno Linux compatible. Es especialmente útil cuando trabajamos directamente sobre repositorios locales y queremos que Codex tenga contexto real del proyecto.

- **Pros:** es la forma más simple de empezar; funciona bien en proyectos locales; permite trabajar desde Debian sin depender de una interfaz web; puede leer, modificar y ejecutar código en el directorio seleccionado.
- **Contras:** no está pensado como API; depende de interacción humana si lo usamos en modo normal; no es ideal para integrarlo directamente con aplicaciones externas; sigue necesitando conexión con OpenAI para usar el modelo.

2. Usar codex exec en scripts y automatizaciones

La segunda forma es usar Codex en modo no interactivo mediante **codex exec**. Este modo está pensado para lanzar tareas desde scripts, jobs de integración continua o automatizaciones donde no queremos abrir la interfaz interactiva.

```
codex exec "Analiza este repositorio y dime si hay errores evidentes"
```

También podemos integrarlo en scripts Bash, tareas programadas, pipelines o sistemas internos que necesiten pedirle a Codex una revisión puntual. Por ejemplo, podríamos ejecutar Codex después de descargar un repositorio, antes de lanzar tests o como paso previo a una revisión manual.

- **Pros:** es fácil de automatizar; encaja bien con scripts; no requiere construir un cliente propio; sirve para tareas repetibles; puede devolver resultados sin abrir una sesión interactiva.
- **Contras:** ofrece menos control fino que el SDK o el app-server; no está pensado para mantener una interfaz rica de conversación; puede ser incómodo si necesitamos gestionar eventos, aprobaciones, historial o streaming avanzado.

3. Controlar Codex desde código con el SDK

Otra opción es usar el **Codex SDK**. Esta vía es más interesante cuando queremos integrar Codex dentro de una aplicación propia escrita en Python o JavaScript, sin tener que hablar directamente con el protocolo interno del app-server. El SDK nos permite iniciar conversaciones, lanzar tareas, recibir resultados y controlar Codex desde nuestro propio código. En el caso de Python, el SDK controla un app-server local de Codex mediante JSON-RPC, pero nos oculta buena parte de esa complejidad.

```
#!/usr/bin/env python3

from openai_codex import Codex, Sandbox
```

```
def fMain():
  with Codex() as vCodex:
    vThread = vCodex.thread_start(
      model="gpt-5.4",
      sandbox=Sandbox.workspace_write
    )

    vResultado = vThread.run(
      "Analiza este proyecto y resume los puntos más importantes."
    )

    print(vResultado.final_response)


if __name__ == "__main__":
  fMain()
```

Esta opción es más limpia que invocar comandos desde Bash cuando queremos construir una integración seria. Por ejemplo, podríamos crear una herramienta interna que mande tareas a Codex, guarde los resultados en una base de datos y genere informes para un equipo de desarrollo o ciberseguridad.

- **Pros:** es más cómodo que hablar JSON-RPC directamente; permite integrar Codex en aplicaciones propias; ofrece más control que codex exec; es adecuado para automatizaciones complejas.
- **Contras:** dependemos del SDK y de sus abstracciones; no es una API HTTP compatible con OpenAI; requiere programar; si queremos control total sobre eventos, aprobaciones y clientes ricos, puede quedarse corto frente al app-server directo.

4. Exponer Codex localmente con Codex App Server

Codex App Server es la forma más potente de controlar Codex desde otro programa. No expone una API REST compatible con OpenAI, sino un servidor local que habla JSON-RPC. Está pensado para clientes ricos, como extensiones de editor, interfaces personalizadas o herramientas internas que necesiten controlar threads, turns, aprobaciones, eventos y cambios de archivos.

```
codex app-server --listen ws://127.0.0.1:4500
```

El app-server permite construir un cliente propio que envía mensajes JSON-RPC a Codex. Por ejemplo, podemos crear un thread, iniciar un turn, recibir eventos en streaming, revisar cambios de archivos, aceptar o rechazar aprobaciones y mostrar todo eso en una interfaz web propia. No debemos confundirlo con una API local compatible con OpenAI. No podemos apuntar directamente una aplicación que use **/v1/chat/completions** o **/v1/responses** contra el app-server y esperar que funcione. El protocolo es distinto.

- **Pros:** es la opción con más control; permite crear clientes personalizados; soporta eventos en streaming; gestiona conversaciones, aprobaciones, historial y cambios de archivos; es ideal para integraciones avanzadas.
- **Contras:** es más complejo; hay que implementar un cliente JSON-RPC; no es compatible directamente con la API REST de OpenAI; no conviene exponerlo en red sin autenticación y aislamiento adecuados.

5. Usar Codex como herramienta mediante MCP

La quinta forma es usar Codex dentro de flujos basados en **MCP**, el Model Context Protocol. MCP permite conectar modelos con herramientas, documentación, navegadores, Figma, repositorios u otros servicios. En el caso de Codex, podemos usar servidores MCP para ampliar el contexto y las capacidades disponibles.

```
codex mcp-server
```

Esta vía es interesante cuando queremos que Codex participe en un ecosistema de agentes o herramientas. En vez de usar

Codex como aplicación final, lo usamos como una pieza conectable dentro de una arquitectura mayor. Por ejemplo, podríamos tener un agente principal que delegue tareas de programación en Codex, o podríamos dar a Codex acceso a documentación técnica interna mediante un servidor MCP. También podríamos conectar Codex con herramientas de desarrollo para reducir trabajo manual repetitivo.

- **Pros:** permite conectar Codex con herramientas externas; encaja bien en arquitecturas de agentes; sirve para ampliar contexto; permite reutilizar integraciones MCP existentes.
- **Contras:** añade complejidad; requiere entender MCP; puede ser excesivo para tareas simples; hay que controlar muy bien qué herramientas y permisos damos al agente.

Comparación rápida

FORMA	USO PRINCIPAL	COMPLEJIDAD	CUÁNDO USARLA
Codex CLI	Uso manual desde terminal	Baja	Cuando queremos trabajar directamente sobre un repo local
codex exec	Automatización simple	Baja-media	Cuando queremos lanzar Codex desde scripts o CI
Codex SDK	Integración desde código	Media	Cuando queremos controlar Codex desde Python o JavaScript
Codex App Server	Clientes personalizados	Alta	Cuando queremos construir una interfaz o backend propio
MCP	Conectar Codex con herramientas	Media-alta	Cuando queremos integrar Codex dentro de un ecosistema de agentes

En la práctica, si solo queremos automatizar tareas simples, probablemente nos baste con **codex exec**. Si queremos programar una integración mantenible, el **SDK** es mejor punto de partida. Si queremos construir nuestro propio cliente completo, entonces tiene sentido mirar **Codex App Server**. Y si queremos conectar Codex con herramientas externas o agentes, la vía natural es **MCP**.