

MÓDULO 1

Fundamentos de Codex como agente de desarrollo

Plan de Formación — OpenAI Codex para Desarrolladores

Nivel: Introductorio · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

Índice

1. Introducción y contexto

Este módulo introductorio establece los cimientos para trabajar con OpenAI Codex de forma productiva y segura. A diferencia de un simple chatbot que responde preguntas sobre código, Codex es un agente autónomo de ingeniería de software: lee ficheros, edita código, ejecuta comandos, corre tests y propone pull requests. Comprender esta diferencia fundamental es la clave para aprovechar su potencial sin perder el control del proceso de desarrollo.

Al finalizar este módulo, el alumno habrá completado un ciclo completo de trabajo con Codex: solicitar una funcionalidad, revisar el diff generado, refinar el resultado, ejecutar validaciones y obtener un entregable limpio listo para integrarse en un repositorio real.

2. Objetivos de aprendizaje

#	Objetivo	Evidencia de logro
O1	Entender qué es Codex como agente (lee/edita/ejecuta código) y diferenciarlo de un chat de ayuda.	El alumno explica las capacidades agente vs. chat con ejemplos concretos.
O2	Conocer los cuatro clientes: CLI, extensión IDE, app de escritorio y Codex Cloud.	El alumno describe cuándo usar cada superficie según el escenario.
O3	Adoptar el flujo “humano en control”: revisión de diffs, aprobaciones y trazabilidad.	El alumno completa el laboratorio con ciclo agente→diff→validación.
O4	Aplicar buenas prácticas para formular tareas acotadas y verificables.	El alumno redacta prompts con objetivo, restricciones y definición de done.

3. Contenidos teóricos

3.1 Qué es Codex: de chatbot a agente de desarrollo

Codex es el agente de programación con IA de OpenAI. Su modelo fundacional actual es GPT-5.3-Codex, una versión de GPT-5 optimizada específicamente para tareas de ingeniería de software mediante reinforcement learning sobre tareas reales de codificación.

3.1.1 Chat vs. agente: diferencias fundamentales

Para entender el salto de paradigma, es necesario distinguir claramente entre dos modos de interacción con IA para desarrollo:

Dimensión	Chat de ayuda (ej. ChatGPT)	Agente de desarrollo (Codex)
Acceso al código	Solo ve lo que pegas en el prompt	Lee todo el repositorio, navega la estructura de directorios
Ejecución	No ejecuta nada; sugiere código para copiar/pegar	Ejecuta comandos (build, test, lint), scripts y herramientas
Escritura	No modifica ficheros	Crea, edita y borra ficheros directamente en tu workspace
Validación	No puede verificar si su código funciona	Ejecuta tests, comprueba errores y corrige iterativamente
Contexto	Limitado a la ventana de conversación	Conoce AGENTS.md, skills, ficheros del proyecto y herramientas MCP
Autonomía	Responde una vez y espera	Puede trabajar autónomamente durante horas en tareas complejas
Resultado	Texto con código embebido	Diffs, commits, pull requests, ficheros creados directamente

3.1.2 Capacidades base de Codex

Según la documentación oficial de OpenAI, Codex puede:

- **Escribir código:** Describe lo que quieras construir y Codex genera código adaptándose a la estructura y convenciones de tu proyecto existente.
- **Comprender codebases desconocidas:** Puede leer y explicar código complejo o legacy, ayudándote a entender cómo los equipos organizan sus sistemas.
- **Revisar código:** Analiza código para identificar bugs potenciales, errores lógicos y edge cases no manejados.
- **Depurar y corregir problemas:** Cuando algo falla, Codex ayuda a trazar el error, diagnosticar causas raíz y sugerir correcciones específicas.
- **Automatizar tareas de desarrollo:** Ejecuta flujos repetitivos como refactorizaciones, renombrados, migraciones, tests y configuración para que el desarrollador se centre en tareas de mayor nivel.

💡 Concepto clave

Codex no es un autocompletador de código (como Copilot inline). Es un **compañero de equipo autónomo** que recibe tareas, las descompone, las ejecuta y entrega resultados verificados. Tu rol como desarrollador pasa de “escribir cada línea” a “dirigir, revisar y validar”.

3.1.3 Razonamiento adaptativo

Los modelos Codex ajustan dinámicamente el tiempo que dedican a “pensar” según la complejidad de la tarea. Una consulta simple recibe una respuesta rápida, mientras que un refactor complejo activa razonamiento profundo que puede extenderse durante horas de forma autónoma. Esto se controla con el parámetro reasoning effort (low / medium / high / xhigh):

Nivel	Uso recomendado	Características
low	Consultas rápidas, Q&A sobre código	Respuesta inmediata, consume menos tokens
medium (default)	Desarrollo interactivo diario	Mejor balance entre inteligencia y velocidad
high	Bugs complejos, refactors multiarchivo	Razonamiento profundo, más lento
xhigh	Migraciones masivas, tareas de días	Máxima capacidad, alto consumo de tokens/rate limits

3.2 Los cuatro clientes de Codex

Codex es un único agente unificado, pero dispone de múltiples superficies (clientes) para adaptarse al lugar donde cada desarrollador trabaja. Todos comparten la misma cuenta ChatGPT, configuración y contexto, permitiéndote mover trabajo sin perder contexto.

3.2.1 Codex CLI (terminal)

Agente open source basado en Rust que se ejecuta directamente en tu terminal. Dispone de una interfaz TUI (Terminal User Interface) a pantalla completa donde puedes iterar conversacionalmente mientras revisas las acciones del agente en tiempo real.

- **Instalación:** npm i -g @openai/codex
- **Plataformas:** macOS, Linux, Windows (nativo experimental o WSL2)
- **Sandbox:** Mecanismos a nivel de OS (Seatbelt en macOS, Landlock/Bubblewrap en Linux)
- **Punto fuerte:** Máximo control, scriptable con `codex exec`, integrable en CI/CD

Ejemplo de inicio rápido:

```
# Modo interactivo (TUI)
cd ~/mi-proyecto
codex

# Modo full-auto (escritura en workspace + aprobación on-request)
codex --full-auto

# Ejecución no interactiva (para scripts/CI)
codex exec "Run tests and fix failures" --json
```

3.2.2 Extensión IDE (VS Code, JetBrains y compatibles)

Panel lateral integrado en tu editor que comparte la misma configuración config.toml que el CLI. Disponible para VS Code, Cursor, Windsurf y JetBrains (IntelliJ, PyCharm, WebStorm, Rider).

- **Instalación VS Code:** Marketplace → buscar “Codex” (publisher: OpenAI)
- **Contexto automático:** Ficheros abiertos, selecciones de código y `@archivo` como referencias
- **Tres modos:** Chat (solo conversación), Agent (lee/escribe/ejecuta con aprobación), Agent Full Access (sin aprobaciones)
- **Punto fuerte:** Contexto visual inmediato, drag & drop de imágenes, delegación a cloud desde el IDE

3.2.3 Codex App (escritorio — macOS)

Aplicación de escritorio exclusiva para macOS (Apple Silicon) que funciona como centro de mando para programación con agentes. Incluye:

- Barra lateral de proyectos con worktrees integrados.
- Lista de threads para ejecutar múltiples agentes en paralelo.
- Panel de revisión de diffs antes de aplicar cambios.
- Lanzamiento de tareas cloud directamente desde la app.
- **Punto fuerte:** Gestión multi-agente, productividad en equipos grandes

3.2.4 Codex Cloud (web)

Accesible desde chatgpt.com/codex. Cada tarea se ejecuta en un contenedor aislado gestionado por OpenAI, preconfigurado con tu repositorio GitHub. Permite trabajar en tareas en background (incluso en paralelo) y proponer pull requests directamente.

- **Requisito:** Conectar cuenta GitHub para acceso a repositorios
- **Internet:** Deshabilitado por defecto durante ejecución (habilitado durante setup de dependencias)
- **@codex en PRs:** Mencionar `@codex` en un comentario de GitHub lanza una tarea cloud de revisión/corrección
- **Punto fuerte:** Tareas largas autónomas, revisiones de PR, sin consumo de recursos locales

3.2.5 Tabla comparativa de superficies

Característica	CLI	IDE Extension	App (macOS)	Cloud (web)
Sistema operativo	macOS/Linux/Win	macOS/Linux (Win exp.)	macOS ARM	Navegador/iOS
Ejecución	Local	Local	Local	Remota (container)
Multi-agente	No	No	Sí (threads)	Sí (tareas paralelas)
CI/CD integrable	Sí (codex exec)	No directamente	No	Sí (GitHub Action)
Contexto de ficheros	CWD + <code>@file</code>	Ficheros abiertos + <code>@file</code>	Proyecto seleccionado	Repo GitHub completo
Sandbox	OS-level configurable	OS-level (misma config)	OS-level	Container aislado
Requiere GitHub	No	No	No	Sí
Open source	Sí	Parcial	No	No

3.3 Filosofía de trabajo: humano en control

El paradigma fundamental de trabajo con Codex es que el humano mantiene el control en todo momento. Codex es un agente poderoso, pero opera dentro de límites explícitos definidos por el desarrollador. Esta filosofía se materializa en tres mecanismos:

3.3.1 Sistema de aprobaciones

Antes de ejecutar un comando generado, Codex puede detenerse y pedir aprobación al usuario. El comportamiento depende de la política configurada:

Política	Comportamiento	Cuándo usarla
untrusted	Solo comandos read-only auto-ejecutan; todo lo demás pide permiso	Repos desconocidos, primeros días con Codex
on-request (default)	El modelo decide cuándo pedir permiso según el riesgo	Desarrollo diario normal
on-failure	Auto-ejecuta todo en sandbox; pide permiso solo si falla	Desarrolladores experimentados con tests sólidos
never	Nunca pide (PELIGROSO)	Solo en contenedores CI aislados externamente

3.3.2 Sandbox y aislamiento

El sandbox controla lo que Codex puede hacer técnicamente cuando ejecuta comandos:

- **read-only**: Solo puede leer el filesystem. No ejecuta comandos ni escribe. Ideal para exploración.
- **workspace-write (recomendado)**: Puede leer y escribir dentro del directorio de trabajo. Red deshabilitada por defecto. Es el modo estándar para desarrollo.
- **danger-full-access**: Sin restricciones de filesystem ni red. Solo usar en entornos ya aislados (containers de CI).

⚠️ Advertencia de seguridad

Al arrancar, Codex detecta si el directorio está bajo control de versiones y recomienda automáticamente: **workspace-write + on-request** para repos con Git. Si el directorio no está versionado, puede empezar en **read-only** hasta que lo marques como trusted.

3.3.3 Revisión de diffs y trazabilidad

Todo cambio que Codex realiza se presenta como un diff revisable antes de aplicarse. El flujo recomendado es:

1. Codex propone cambios y muestra el plan de acción.
2. El desarrollador revisa el diff (en TUI, IDE o panel de revisión de la app).
3. Si el cambio es correcto, se aprueba y aplica.
4. Si requiere ajustes, se proporciona feedback directo (“Cambia el nombre de esta función”, “Añade manejo de errores”).
5. Se ejecutan tests/lint para validar el resultado.
6. Se crea un checkpoint Git para trazabilidad.

💡 Consejo práctico

Crea **checkpoints Git** antes y después de cada tarea de Codex. Esto permite revertir fácilmente cualquier cambio (`git stash` o `git reset --hard`). Trata cada sesión de Codex como si fuera un PR de un colega: revisas, comentas, pides cambios y apruebas.

3.4 Tareas pequeñas vs. tareas largas

Codex es versátil: puede resolver una consulta rápida en segundos o trabajar autónomamente durante horas en migraciones complejas. La clave está en elegir la estrategia adecuada:

Dimensión	Tarea pequeña / interactiva	Tarea larga / autónoma
Ejemplo	Añadir endpoint, fix bug específico, escribir tests para un módulo	Migrar JS a TS, refactor completo de capa de datos, auditoría de seguridad
Superficie preferida	CLI interactivo / IDE Extension	Codex Cloud / App (threads paralelos)
Reasoning effort	medium (rápido y suficiente)	high / xhigh (profundo y autónomo)
Interacción	Conversacional: prompt → revisión → ajuste	Delegar y monitorizar: lanzar → revisar resultado final
Validación	Tests locales inmediatos	Codex ejecuta tests en cloud; revisar antes de merge
Duración típica	Minutos	Minutos a horas (hasta 7+ horas observadas)

La recomendación general para equipos que empiezan con Codex es comenzar con tareas pequeñas e interactivas para desarrollar confianza y comprensión del flujo, y escalar progresivamente hacia tareas más autónomas a medida que se establezcan buenas prácticas de AGENTS.md, skills y revisión.

4. Buenas prácticas

4.1 Pedir resultados verificables

Cada solicitud a Codex debe incluir criterios que permitan verificar automáticamente el resultado. Esto transforma la IA de una “caja negra que genera código” a una herramienta cuyo output es comprobable:

- **“Incluye comandos para ejecutar tests”**: Codex no solo genera el código, sino que lo ejecuta y reporta resultados.
- **“Muestra el diff antes de aplicar”**: En modo interactivo, Codex muestra el plan y los cambios propuestos.
- **“Explica los supuestos que has tomado”**: Si Codex necesita tomar decisiones de diseño, las explica en lugar de asumirlas silenciosamente.
- **“Ejecuta lint después de los cambios”**: Verifica que el código generado cumple con las reglas de estilo del proyecto.

Ejemplo de prompt verifiable vs. no verifiable:

```
# X Prompt no verificable
"Añade una función de autenticación"

# ✓ Prompt verificable
"Implementa middleware de autenticación JWT en auth_middleware.py.
Debe verificar el header Authorization con formato Bearer <token>.
Si el token es inválido, devuelve 401 con body {"error": "Unauthorized"}.
Crea tests con pytest que cubran: token válido, token expirado,
token malformado y header ausente. Ejecuta los tests y muestra resultados.
Ejecuta flake8 para verificar que no hay warnings."
```

4.2 Mantener tareas acotadas

Cada tarea que envías a Codex debe tener tres elementos claros:

1. Objetivo concreto: qué quieres que haga (una función, un fix, un test, no “todo”).
2. Restricciones explícitas: qué NO debe tocar, qué convenciones seguir, qué dependencias usar.
3. Definición de “done”: cómo saber que la tarea está completa (tests pasan, lint limpio, endpoint responde).

Este enfoque reduce drásticamente los problemas de “Codex tocó demasiados archivos” o “cambió el estilo sin permiso”.

4.3 Revisar siempre como si fuera un PR

El código generado por IA requiere el mismo nivel de revisión que un pull request de un colega:

- Revisar cada diff línea por línea antes de aplicar.
- Verificar que los tests cubren los casos críticos, no solo el happy path.
- Comprobar que no se han añadido dependencias innecesarias.
- Buscar secrets hardcodeados, consultas SQL sin parametrizar u otros patrones inseguros.
- Ejecutar el linter del proyecto para verificar consistencia de estilo.

💡 Regla de oro

Si no lo aprobarías en un PR de un compañero, no lo apruebes a Codex. La IA genera código con alta calidad, pero puede introducir errores sutiles, dependencias innecesarias o patrones inseguros que solo un humano con contexto de negocio puede detectar.

4.4 Usar Git como red de seguridad

- Antes de cada tarea: git add -A && git stash o crear una rama dedicada.
- Después de cada tarea validada: git commit con mensaje descriptivo.
- Si Codex produce un resultado incorrecto: git checkout -- . o git stash pop.
- Para tareas grandes en cloud: Codex propone PRs que puedes revisar antes de merge.

5. Errores comunes y cómo evitarlos

5.1 “Hazlo todo” sin criterios de aceptación

El problema: Prompts como “Crea una aplicación web completa” sin especificar tecnologías, endpoints, modelo de datos ni criterios de calidad producen resultados impredecibles y difíciles de integrar.

La solución: Descomponer en tareas atómicas. En lugar de “Crea una app”, pedir secuencialmente: “Crea el modelo de datos” → “Implementa el endpoint de creación” → “Añade validación” → “Genera tests”.

5.2 No fijar límites: el agente toca demasiados archivos

El problema: Sin restricciones explícitas, Codex puede refactorizar ficheros que no le has pedido, cambiar imports, formatear código que no es parte del scope o añadir dependencias no deseadas.

La solución: Incluir restricciones en el prompt: “Solo modifica ficheros en src/auth/. No toques otros directorios. No añadas dependencias nuevas.” Usar el sandbox `workspace-write` que limita la escritura al directorio de trabajo.

5.3 Confiar en explicaciones sin ejecutar tests/lint

El problema: Codex puede generar código que “parece correcto” y explicar convincentemente por qué funciona, pero contener errores sutiles (off-by-one, race conditions, edge cases no cubiertos).

La solución: Siempre pedir ejecución de tests como parte del prompt. Incluir en AGENTS.md las instrucciones de test y lint del proyecto. Usar `/review` después de cada cambio significativo.

5.4 Errores adicionales frecuentes

Error	Consecuencia	Prevención
Usar --yolo en la máquina de desarrollo	El agente puede ejecutar cualquier comando sin sandbox ni aprobación	Reservar exclusivamente para contenedores CI aislados
No crear checkpoints Git	Imposible revertir cambios no deseados	git commit antes y después de cada tarea
Ignorar el contexto del AGENTS.md	Codex no conoce las convenciones del proyecto	Crear AGENTS.md con comandos de test, lint y convenciones
Subir reasoning effort sin necesidad	Agota rate limits rápidamente	Usar medium por defecto; high solo para tareas complejas
No revisar dependencias añadidas	Codex puede instalar paquetes innecesarios o inseguros	Especificar dependencias permitidas o pedir confirmación

6. Casos de uso reales en desarrollo de software

6.1 Onboarding a repositorios grandes

Escenario: Un desarrollador se incorpora a un equipo con un monorepo de 500+ ficheros. Necesita entender la arquitectura, los puntos de entrada, las dependencias entre módulos y los patrones utilizados.

Cómo usar Codex: Abrir el proyecto en Codex CLI o IDE y pedir:

```
"Analiza la estructura de este repositorio. Genera un mapa de módulos mostrando: directorios principales con su propósito, puntos de entrada de la aplicación, dependencias entre módulos y patrones de diseño utilizados. Presenta el resultado como un documento ARCHITECTURE.md."
```

Resultado: Codex lee toda la estructura, analiza imports y exports, identifica patrones y genera documentación que normalmente llevaría días de trabajo manual.

6.2 Hotfix guiado con pruebas

Escenario: Alerta de producción: el endpoint /api/orders devuelve 500 para pedidos con descuento > 100%. El equipo necesita un fix urgente con pruebas.

Cómo usar Codex:

```
"El endpoint /api/orders devuelve 500 cuando discount_percentage > 100. El error está en src/services/order_service.py.  
1. Diagnóstica la causa raíz analizando el código.  
2. Implementa el fix con validación adecuada (cap discount a 0-100).  
3. Añade tests que cubran: descuento 0%, 50%, 100%, 150% y -10%.  
4. Ejecuta pytest y muestra resultados.  
5. Ejecuta flake8 para verificar estilo."
```

Resultado: Codex diagnostica el bug (división que produce valor negativo), aplica el fix, genera tests exhaustivos y ejecuta la validación. Todo en minutos, revisable como diff.

6.3 Generación de documentación técnica del repositorio

Escenario: Proyecto interno sin documentación: las funciones no tienen docstrings, no hay README actualizado y la API no está documentada.

Cómo usar Codex:

```
"Añade docstrings Google-style a todas las funciones públicas de src/. Genera README.md con: descripción del proyecto, instalación, uso, arquitectura simplificada y ejemplos. Genera también un fichero CONTRIBUTING.md con guía de contribución siguiendo las convenciones del proyecto."
```

Resultado: Documentación completa generada en minutos, coherente con el código real del proyecto.

6.4 Caso adicional: triaje de issues con automatizaciones

Los equipos de OpenAI usan Codex internamente para clasificar on-call issues, planificar tareas al inicio del día y delegar trabajo de background. Codex puede actuar sin instrucciones explícitas mediante automatizaciones que monitorizan alertas, clasifican incidencias y proponen fixes.

7. Laboratorio práctico (L1) — Primer ciclo agente→diff→validación

⌚ Objetivo del laboratorio

Crear una funcionalidad pequeña en un repositorio de ejemplo usando Codex, completando el ciclo completo: solicitar → revisar diff → refinar → validar → entregar. Al finalizar, el alumno tendrá un PR local con un endpoint funcional, tests verdes y README generado.

7.1 Prerrequisitos

- Node.js v22+ y npm instalados (para Codex CLI).
- Python 3.10+ con pip (para el servicio de ejemplo).
- Git instalado y configurado.
- Codex CLI instalado: npm i -g @openai/codex
- Sesión autenticada: ejecutar codex y completar el flujo de login (ChatGPT o API key).
- Opcional: VS Code con extensión Codex instalada (para la variante IDE).

7.2 Paso 1 — Crear el repositorio temporal con servicio mínimo

Tiempo estimado: 5 minutos

```
# Crear directorio aislado
mkdir /tmp/codex-lab01 && cd /tmp/codex-lab01

# Inicializar repositorio Git
git init

# Crear estructura mínima Python/FastAPI
python3 -m venv .venv
source .venv/bin/activate    # Linux/macOS
# .venv\Scripts\activate     # Windows

pip install fastapi uvicorn httpx pytest

# Crear fichero principal mínimo
cat > main.py << 'EOF'
from fastapi import FastAPI

app = FastAPI(title="Lab01 - Codex Intro")

@app.get("/health")
def health_check():
    return {"status": "ok"}
EOF

# Checkpoint inicial
git add -A && git commit -m "chore: scaffold inicial FastAPI"
```

[i] Nota para variante IDE

Si prefieres usar la extensión VS Code: abre el directorio `/tmp/codex-lab01` en VS Code (`code /tmp/codex-lab01`), abre el panel Codex desde la barra lateral y selecciona modo **Agent**. Los pasos siguientes son idénticos, pero usas el panel de chat en lugar del TUI de terminal.

7.3 Paso 2 — Solicitar funcionalidad a Codex

Tiempo estimado: 5–10 minutos

Lanza Codex en modo interactivo desde el directorio del proyecto:

```
cd /tmp/codex-lab01
source .venv/bin/activate
codex --full-auto
```

Introduce el siguiente prompt (adaptable según preferencia):

Implementa un endpoint POST `/api/tasks` en `main.py` que permita crear tareas.

Especificaciones:

- Modelo de datos Task con campos: `id` (UUID generado), `title` (str, obligatorio), `description` (str, opcional) y `done` (bool, default false).
- Validación con Pydantic: `title` no puede estar vacío ni superar 200 caracteres.
- Almacenamiento en memoria (dict global, no BD).
- Respuesta 201 con el objeto Task creado.
- Respuesta 422 si la validación falla.

Además:

- Implementa un endpoint GET `/api/tasks` que devuelva todas las tareas.
- Crea tests completos en `test_main.py` usando `httpx + pytest`:
 - Crear tarea válida (verificar 201 y campos)
 - Crear tarea sin `title` (verificar 422)
 - Crear tarea con `title > 200 chars` (verificar 422)
 - Listar tareas (verificar que devuelve las creadas)
- Ejecuta `pytest` y muestra resultados.
- Genera un `README.md` básico con: descripción, cómo instalar, cómo ejecutar y cómo testear.

Observa cómo Codex:

1. Lee la estructura actual del proyecto (`main.py`, `requirements`).
2. Planifica las acciones necesarias.
3. Edita `main.py` añadiendo el modelo Pydantic, las rutas y el almacén en memoria.
4. Crea `test_main.py` con los tests solicitados.
5. Ejecuta `pytest` y muestra el output.
6. Genera `README.md`.

7.4 Paso 3 — Revisar el diff y pedir ajustes

Tiempo estimado: 5–10 minutos

Una vez que Codex termine, revisa los cambios:

```
# Ver el diff completo
git diff

# O usar el comando de revisión integrado de Codex
/review
```

Revisa críticamente y solicita ajustes si es necesario. Ejemplos de feedback:

```
# Ejemplo 1: Naming
"Renombra la variable `db` a `task_store` para mayor claridad."

# Ejemplo 2: Estructura
"Mueve el modelo Task a un fichero separado models.py e importa desde main.py."

# Ejemplo 3: Tests adicionales
"Añade un test que verifique que el campo done es false por defecto."

# Ejemplo 4: Mid-turn steering (mientras Codex trabaja)
# Pulsa Enter durante la ejecución para inyectar:
"Añade también un endpoint DELETE /api/tasks/{task_id} con su test."
```

Ejecuta la validación final:

```
# Ejecutar tests
pytest -v

# Ejecutar linter (si está instalado)
flake8 main.py test_main.py --max-line-length 120

# Verificar que el servidor arranca
uvicorn main:app --host 0.0.0.0 --port 8000 &
curl http://localhost:8000/health
curl -X POST http://localhost:8000/api/tasks -H "Content-Type: application/json" \
-d '{"title": "Mi primera tarea con Codex"}'
kill %1 # Detener servidor
```

7.5 Paso 4 — Crear el PR local

Tiempo estimado: 2 minutos

```
# Crear rama de feature
git checkout -b feature/task-api

# Commit del trabajo de Codex
git add -A
git commit -m "feat: endpoint POST/GET /api/tasks con validación y tests"

# Ver el log
git log --oneline
```

7.6 Resultado esperado

Al finalizar el laboratorio, el repositorio debe contener:

Fichero	Contenido	Validación
main.py	App FastAPI con /health, POST /api/tasks, GET /api/tasks, modelo Pydantic	unicorn arranca sin errores
test_main.py	Tests: crear válida, crear sin title, title largo, listar	pytest -v con 4+ tests verdes
README.md	Descripción, instalación, ejecución, testing	Contenido coherente con el código
Git log	2 commits: scaffold + feature	git log --oneline muestra ambos

7.7 Limpieza (OBLIGATORIA)

⚠️ Limpieza de recursos

Ejecutar al finalizar el laboratorio: No dejar residuos en el sistema.

```
# 1. Desactivar el entorno virtual
deactivate

# 2. Borrar el repositorio completo
rm -rf /tmp/codex-lab01

# 3. Si configuraste variables de entorno temporales:
unset OPENAI_API_KEY

# 4. Si creaste ficheros .env temporales, borrarlos:
# rm ~/.env (solo si fue creado para este lab)

# 5. Verificar limpieza
ls /tmp/codex-lab01 2>/dev/null && echo 'ERROR: directorio aún existe' || echo 'OK:
limpio'
```

8. Resumen del módulo y siguientes pasos

8.1 Conceptos clave aprendidos

Concepto	Detalle
Codex = agente	No es un chat: lee, escribe, ejecuta y valida código en tu repositorio
Cuatro superficies	CLI (terminal), IDE Extension (VS Code/JetBrains), App (macOS), Cloud (web/GitHub)
Humano en control	Sandbox + política de aprobación + revisión de diffs + checkpoints Git
Tareas verificables	Incluir criterios de aceptación, tests y lint en cada solicitud
Tareas acotadas	Objetivo + restricciones + definición de done
Reasoning effort	low/medium/high/xhigh según la complejidad de la tarea

8.2 Preparación para el siguiente módulo

En el Módulo 2, profundizaremos en la instalación y configuración avanzada del entorno: fichero config.toml, capas de configuración, perfiles, modos de sandbox y políticas de aprobación. Para prepararse:

- Tener Codex CLI instalado y autenticado (ya completado en el lab).
- Tener VS Code con la extensión Codex instalada.
- Revisar brevemente la página de configuración básica:
<https://developers.openai.com/codex/config-basic/>

9. Referencias y recursos

- Documentación oficial de Codex: <https://developers.openai.com/codex>
- Quickstart: <https://developers.openai.com/codex/quickstart>
- Guía de prompting: <https://developers.openai.com/codex/prompting>
- Workflows (recetas end-to-end): <https://developers.openai.com/codex/workflows/>
- Seguridad (sandbox y aprobaciones): <https://developers.openai.com/codex/security/>
- Extensión IDE: <https://developers.openai.com/codex/ide/>
- CLI Features: <https://developers.openai.com/codex/cli/features/>
- Codex Cloud: <https://developers.openai.com/codex/cloud>
- Modelos disponibles: <https://developers.openai.com/codex/models/>
- Changelog: <https://developers.openai.com/codex/changelog/>
- Repositorio open source (CLI): <https://github.com/openai/codex>
- OpenAI Academy — Codex for Builders:
<https://academy.openai.com/public/resources/codex-for-builders>