APRENDE HACIENDO

La Tripulación Agéntica

Guía Práctica — Edición macOS

Rasmus Bornhøft Schlünsen

Marzo 2026

rev 95

Una Nota Antes de Empezar

Este libro es el compañero práctico de *La Tripulación Agéntica*. Mientras el libro principal explica las ideas, este las pone en práctica.

Cada capítulo es un ejercicio. Configurarás herramientas reales, trabajarás con repositorios reales y construirás cosas reales, empezando desde cero. Los ejercicios están escritos para macOS.

No necesitas ser programador. Sí necesitas estar dispuesto a escribir comandos en una terminal y ver qué pasa. Así es como se aprende esto, no leyendo sobre ello, sino haciéndolo.

Al final de este libro, tendrás un entorno de desarrollo funcional, experiencia práctica con Git y GitHub, y la confianza para colaborar en proyectos reales usando agentes de IA como copiloto.

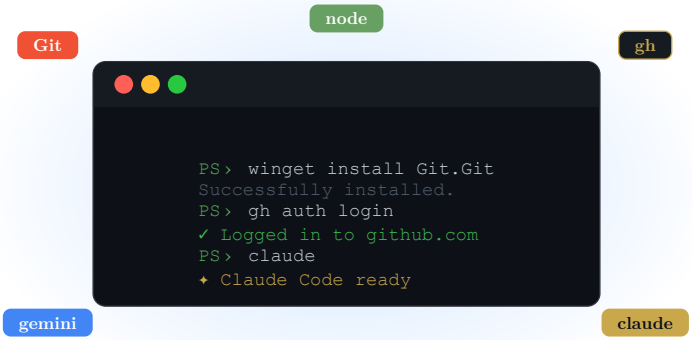
Rasmus Bornhøft Schlünsen
Marzo 2026

Ejercicios

Configuración de tu Entorno de Trabajo

Antes de poder contribuir a un proyecto, corregir un error o simplemente leer código fuente correctamente, necesitas un entorno de trabajo funcional. Este capítulo prepara tu máquina desde cero, cualquiera que sea el sistema operativo que uses.

Al final de este capítulo, tendrás una terminal, un gestor de paquetes, Git, el CLI de GitHub y, opcionalmente, un asistente de inteligencia artificial, todo listo para funcionar.



Your terminal: the workshop where everything begins

Abre tu Terminal

La terminal es tu línea de comandos: el lugar donde ejecutarás todo en este libro.

La Terminal está integrada. Ábrela desde Spotlight:

1. Presiona `Cmd + Espacio`
2. Escribe **Terminal** y presiona Intro

O encuéntrala en **Aplicaciones → Utilidades → Terminal**.

Verifica que usas un shell moderno:

```
echo $SHELL
```

Deberías ver `/bin/zsh` (predeterminado desde macOS Catalina) o `/bin/bash`. Ambos funcionan con esta guía.

> **¿Por qué la terminal?** Los agentes de programación IA viven en la terminal. No puedes trabajar en pareja con un agente si no tienes un lugar donde pueda operar. Piensa en esto como preparar tu taller antes de empezar a construir.

Instala un Gestor de Paquetes

Un gestor de paquetes te permite instalar software escribiendo un solo comando en lugar de descargar instaladores y hacer clic en asistentes. Cada plataforma tiene el suyo.

Homebrew
Homebrew es el gestor de paquetes estándar para macOS. Instálalo con:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.
com/Homebrew/install/HEAD/install.sh)"
```

Esto descarga y ejecuta el instalador oficial de Homebrew. Sigue las instrucciones: pedirá tu contraseña e instalará algunos elementos. Cuando termine, verifica:

```
brew --version
```

En Macs con Apple Silicon (M1/M2/M3), Homebrew se instala en `/opt/homebrew`. Si `brew` no se encuentra después de la instalación, busca una sección «Next steps» al final de la salida del instalador: mostrará dos comandos que debes ejecutar. Cópialos, ejecútalos y abre una nueva ventana de terminal.

Instala Git

Git es el control de versiones. Registra cada cambio en cada archivo de un proyecto y es la forma en que los equipos colaboran sin sobrescribir el trabajo de los demás. Todos los proyectos de este libro usan Git.

Git viene incluido con las Xcode Command Line Tools, que probablemente ya tienes. Prueba:

```
git --version
```

Si ves un número de versión, ya está listo. Si macOS te pide instalar las herramientas de desarrollo, haz clic en **Instalar** y espera.

Para obtener una versión más reciente con Homebrew:

```
brew install git
```

—

Verifica:

```
git --version
```

Deberías ver un número de versión como `git version 2.47.1`.

Ahora configura tu identidad para que Git sepa quién realiza los cambios:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu@correo.com"
```

Usa el mismo correo que en tu cuenta de GitHub.

Git en Acción

Git está instalado, pero todavía no lo has usado. Hagamos una prueba rápida de 60 segundos para que Git no sea un misterio cuando llegue el Capítulo 2.

Crea una carpeta de práctica y entra en ella:

```
mkdir test-repo && cd test-repo
```

Dile a Git que empiece a rastrear esta carpeta:

```
git init
```

Comprueba el estado:

```
git status
```

Deberías ver algo como: `On branch main — nothing to commit`. Git te está diciendo que está vigilando esta carpeta y que aún no hay nada nuevo que registrar.

En el Capítulo 2 usarás `git status` constantemente: es como verificas qué ha cambiado. Ahora ya sabes cómo se ve cuando todo está limpio.

Vuelve a tu carpeta de inicio cuando hayas terminado:

```
cd ..
```

> **¿Qué acaba de pasar?** `git init` creó una carpeta oculta `.git` dentro de `test-repo`. Ahí es donde Git almacena todo su historial. Todos los proyectos que usan Git tienen una. Nunca necesitas tocarla directamente.

Instala el CLI de GitHub

El CLI de GitHub (`gh`) te permite hacer forks de repositorios, crear pull requests y gestionar issues, todo sin salir de la terminal.

```
brew install gh
```

—

Verifica:

```
gh --version
```

Crea una Cuenta de GitHub

Si aún no tienes una, regístrate en github.com/signup. Es gratuita. La necesitarás para todo a partir del Capítulo 2.

Autentícate

Ahora conecta tu terminal a tu cuenta de GitHub:

```
gh auth login
```

Cuando se te pida, elige:

- **GitHub.com**
- **HTTPS**
- **Login with a web browser**

Te dará un código de un solo uso y abrirá tu navegador. Pega el código, autoriza y ya estarás conectado.

Verifica que funcionó:

```
gh auth status
```

Deberías ver `Logged in to github.com`.

(Opcional) Instala un Asistente de Programación IA

Este libro trata sobre trabajar con agentes de IA. Aunque no es estrictamente necesario para los ejercicios, tener un asistente IA en tu terminal hace que la experiencia sea real.

Tanto Claude Code como Gemini CLI requieren Node.js. Instálalo primero.

Instala Node.js

```
brew install node
```

Cierra y vuelve a abrir tu terminal, luego verifica:

```
node --version
npm --version
```

Opción A: Claude Code

Claude Code es el agente de programación IA de Anthropic. Se ejecuta en tu terminal y puede leer, escribir y razonar sobre código.

Instálalo usando el instalador oficial (el método recomendado):

```
curl -fsSL https://claude.ai/install.sh | bash
```

O mediante npm:

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

Ejecútalo:

```
claude
```

Se te pedirá que te autentiques con tu cuenta Anthropic la primera vez.

Opción B: Gemini CLI
Gemini CLI es el agente de programación IA de Google. Mismo concepto, modelo diferente.

```
npm install -g @google/gemini-cli
gemini
```

Necesitarás una clave API de Google. Configúrala en tu sesión actual:

```
export GEMINI_API_KEY="tu-clave-aquí"
```

Para hacerla permanente, añade la línea `export` a tu archivo de configuración del shell. Este archivo se ejecuta automáticamente cuando se inicia tu terminal:

Abre `~/.zshrc` con:

```
open -e ~/.zshrc
```

Añade `export GEMINI_API_KEY="tu-clave-aquí"` en una nueva línea al final, guarda y cierra. Luego ejecuta `source ~/.zshrc` para aplicar el cambio en la sesión actual.

Si no sabes dónde obtener una clave API de Google, ve a aistudio.google.com, inicia sesión y crea una clave en **Get API key**. Es gratuita para empezar.

¿Cuál elegir?
Cualquiera funciona para este libro. Claude Code destaca en razonamiento de código y ediciones de múltiples archivos. Gemini CLI tiene una fuerte integración con el ecosistema de Google. Prueba ambos si quieres: son gratuitos para empezar. Los ejercicios mostrarán prompts que funcionan con cualquiera de los dos.

Verifica Todo

Ejecuta esta lista de verificación rápida:

```
git --version
gh --version
gh auth status
```

Si los tres comandos funcionan, estás listo. Tu entorno de trabajo está configurado, tu identidad está ajustada y tienes conexión directa a GitHub.

El Enfoque Basado en Prompts

Una vez instalado Claude Code o Gemini CLI, no tienes que recordar cada comando: puedes describir lo que quieres en español natural. Así es como pedirías a un agente IA que verifique toda tu configuración.

Abre tu asistente IA en la terminal:

```
claude
```

Luego prueba prompts como estos:

- *«Comprueba si Git está instalado y correctamente configurado con nombre y correo electrónico.»*
- *«¿Está instalado y autenticado el CLI de GitHub? Muéstrame el estado.»*
- *«Ejecuta una lista de verificación rápida: git, gh y gh auth status — dime qué funciona y qué no.»*

El agente ejecutará los comandos, interpretará la salida y te dirá en lenguaje natural qué está listo y qué aún necesita atención. Si algo falta o está roto, pregúntale:

- *«Git no está configurado con mi correo electrónico — ¿cómo lo arreglo?»*
- *«Guíame paso a paso para autenticar el CLI de GitHub.»*
- *«Estoy en macOS y Homebrew no está en mi PATH — ¿cómo lo soluciono?»*

> **Comandos vs. prompts.** Ambos enfoques te llevan al mismo lugar. Los comandos son rápidos y precisos una vez que los conoces. Los prompts son más tolerantes: te encuentran donde estás. A medida que ganes experiencia, cambiarás entre los dos de forma natural.

En el próximo capítulo, pondremos todo en práctica: harás un fork de un repositorio real, leerás un capítulo real de un libro real, escribirás una reseña honesta y enviarás tu primer pull request.

Tu Primer Pull Request

Este capítulo es un ejercicio práctico. Al final, habrás hecho un fork de un repositorio real, creado una rama, escrito una reseña de un capítulo del libro y enviado un pull request: la forma estándar en que los colaboradores de código abierto proponen cambios.

Esto no es una simulación. Tu pull request aparecerá en GitHub para que personas reales lo lean.

fork → branch → pull request: the open-source contribution loop

Lo que Necesitarás

Todo lo del Capítulo 1:

- Una terminal (abierta y lista)
- Git instalado y configurado

- CLI de GitHub autenticado
- Una cuenta de GitHub

Lo que Está a Punto de Ocurrir

Antes del primer comando, aquí tienes el camino completo que vas a recorrer: siete pasos, uno tras otro:

1. **Fork** — Crea tu propia copia del repositorio del libro en GitHub
2. **Clone** — Descarga esa copia en tu máquina
3. **Branch** — Crea una versión aislada y segura donde puedas hacer cambios
4. **Write** — Añade tu reseña como un nuevo archivo
5. **Commit** — Guarda tus cambios con un mensaje (el equivalente de Git a guardar)
6. **Push** — Envía tus cambios de vuelta a tu copia en GitHub
7. **Pull Request** — Pide al autor que integre tus cambios en el original

Este es el flujo de trabajo del código abierto. Cada contribución a cada proyecto importante del mundo sigue estos mismos siete pasos. Los habrás completado todos al final de este capítulo.

Aquí están los términos clave que verás: léelos una vez ahora y tendrán sentido a medida que avances:

Término	**Qué significa**
Repositorio	Una carpeta de proyecto que Git está rastreando. A menudo llamado «repo».
Fork	Tu copia personal del repositorio de otra persona, almacenada en GitHub.

Clone	Descargar un repositorio de GitHub a tu máquina.
Branch	Una versión paralela del proyecto donde trabajas sin tocar la copia principal.
Commit	Un snapshot guardado de tus cambios, con un mensaje que describe lo que hiciste.
Push	Enviar tus commits desde tu máquina a GitHub.
Pull Request	Una solicitud formal de integrar tu rama en el proyecto original.

No necesitas memorizar estos términos. Se volverán naturales a medida que los uses.

Fork y Clone del Libro

Trabajaremos con *La Tripulación Agéntica*: el libro que estás leyendo ahora mismo. El código fuente vive en GitHub como repositorio público.

Ejecuta este único comando:

```
gh repo fork schlunsen/theagenticcrew --clone --remote
```

Esto hace tres cosas en un solo paso:
1. Crea tu propia copia (un «fork») en GitHub
2. La descarga en tu máquina (un «clone»)
3. Establece la conexión entre tu copia y el original

Entra en el directorio del proyecto:

```
cd theagenticcrew
```

Verifica tu configuración:

```
git remote -v
```

Deberías ver dos remotos:

- `origin` — tu fork (donde envías tus cambios)
- `upstream` — el repositorio original (de donde puedes obtener actualizaciones)

Explora el Proyecto

Antes de cambiar nada, mira a tu alrededor. El libro está escrito en Typst, un lenguaje de composición tipográfica moderno. Los archivos fuente son texto plano: puedes leerlos en cualquier editor.

Observa la estructura del proyecto:

```
ls
ls chapters/
```

Los capítulos están en el directorio `chapters/`, numerados y con nombres descriptivos. El que nos interesa es `01-introduction.typ`.

Lee el Capítulo 1

Ábrelo en un editor de texto:

```
open -e chapters/01-introduction.typ
```

O con VS Code en cualquier plataforma, si lo tienes instalado:

```
code chapters/01-introduction.typ
```

Verás marcado Typst: los encabezados comienzan con `=`, el énfasis usa `_guiones bajos_` y el resto es prosa. Se lee como un documento normal.

Tómate tu tiempo. Lee el capítulo completo. La introducción cubre:

- El viaje del autor descubriendo los flujos de trabajo agénticos
- Por qué el terreno está cambiando para los ingenieros de software
- Para quién es el libro y cómo leerlo
- Qué es el libro y qué no es

Forma tus propias opiniones mientras lees. Ese es el objetivo de este ejercicio.

Crea una Rama

En Git, no editas la copia principal directamente. Creas una **rama**: una versión paralela donde puedes hacer cambios sin afectar el trabajo de nadie más.

```
git checkout -b review/chapter-1-TU-NOMBRE-DE-USUARIO-GITHUB
```

Reemplaza `TU-NOMBRE-DE-USUARIO-GITHUB` con tu nombre de usuario real. Por ejemplo:

```
git checkout -b review/chapter-1-juanperez
```

> **Versiones más recientes de Git (2.23+)** ofrecen `git switch -c` como una alternativa más explícita a `git checkout -b`. Ambos hacen lo mismo. Usamos `checkout` aquí porque es lo que más se ve en tutoriales y documentación.

> **¿Por qué ramas?** Imagina diez personas editando el mismo documento de Google a la vez: caos. Las ramas permiten que cada persona trabaje de forma independiente y luego integre sus cambios uno a uno. Así operan todos los proyectos de software serios.

Escribe tu Reseña

Crea una carpeta `reviews` y tu archivo de reseña.

```
mkdir -p reviews
nano reviews/resena-capitulo-1-TU-NOMBRE-DE-USUARIO-
GITHUB.md
```

Esto abre un sencillo editor de texto en la terminal. Escribe (o pega) tu reseña directamente. Cuando termines, presiona `Ctrl+X` para salir, luego `Y` para confirmar el guardado y `Enter` para mantener el nombre del archivo.

En macOS o Linux también puedes usar cualquier editor que prefieras: `code`, `vim` o lo que te resulte más natural.

Aquí tienes una plantilla para empezar. Escríbela o cópiala de esta página y luego completa tus pensamientos reales bajo cada encabezado:

```
# Reseña — Capítulo 1: Introducción

**Reseñador/a:** @TU-NOMBRE-DE-USUARIO-GITHUB
**Fecha:** AAAA-MM-DD

## Primeras Impresiones
¿Qué te llamó la atención cuando leíste este capítulo por
primera vez?

## Lo que Funcionó Bien
¿Qué partes te resonaron? ¿Qué fue claro y atractivo?

## Lo que Podría Mejorar
Sé honesto/a pero constructivo/a. ¿Secciones confusas?
¿Contexto que faltaba? ¿Algo que no encajara?

## Quién Se Beneficiaría de Este Capítulo
Basándote en lo que leíste, ¿quién es el lector ideal?
```

```
## Puntuación
Tu puntuación general sobre 5, con un resumen de una
línea.
```

No lo pienses demasiado. Un párrafo genuino para cada sección vale más que un ensayo pulido.

El Enfoque Basado en Prompts

El enfoque manual comando a comando que aparece arriba es el flujo de trabajo completo. Una vez instalado Claude Code o Gemini CLI, puedes describir toda la tarea en español natural y dejar que el agente se encargue de la mayor parte.

Two paths, one destination — commands or prompts, the result is the same

Abre tu asistente IA desde dentro del directorio del proyecto clonado:

```
claude
```

O:

```
gemini
```

Deja que el agente lea el capítulo
En lugar de abrir el archivo tú mismo, pide al agente que lo lea y te dé un resumen:

- *«Lee chapters/01-introduction.typ y resume lo que cubre.»*
- *«¿Cuál es el argumento principal de chapters/01--introduction.typ? ¿Para quién cree el autor que es este libro?»*

Esto es útil cuando quieres una orientación rápida antes de leer el capítulo completo tú mismo.

Redacta tu reseña juntos
Una vez que hayas leído el capítulo (tú mismo o con ayuda del agente), úsalo como compañero de reflexión:

- *«Lee chapters/01-introduction.typ y dame tu opinión honesta: ¿qué funciona bien y qué le falta a una introducción de libro?»*
- *«Creo que la introducción dedica demasiado tiempo a la historia personal del autor. ¿Estás de acuerdo? ¿Qué cortarías?»*
- *«Ayúdame a rellenar esta plantilla de reseña para el capítulo 1.»* (pega la plantilla en el prompt)

El agente te dará un borrador al que reaccionar. Acuerda, discrepa, edita: la reseña debe acabar con tu voz, no con la del agente.

Deja que el agente haga el trabajo de Git
Aquí es donde el enfoque basado en prompts realmente brilla. Después de escribir tu reseña, puedes delegar los pasos de Git al agente:

- *«Crea una rama llamada review/chapter-1-TU-NOMBRE-DE-USUARIO-GITHUB, añade mi archivo de reseña, haz commit*

con un mensaje sensato, envíalo a mi fork y abre un pull request.»

El agente ejecutará cada comando, te mostrará lo que está haciendo y señalará cualquier problema. Tú te mantienes al tanto sin tener que recordar la sintaxis exacta.

O ve aún más lejos: describe toda la tarea desde el principio antes de empezar:

- *«Quiero enviar una reseña de chapters/01-introduction.typ como pull request a este repositorio. Guíame paso a paso, o hazlo tú si digo adelante.»*

El agente esbozará el plan, esperará tu aprobación y lo ejecutará.

Cuándo usar comandos vs. prompts

Usa comandos cuando...	**Usa prompts cuando...**
Sabes exactamente qué hacer	No estás seguro de cuál es el siguiente paso
La velocidad importa	Quieres entender qué está pasando
Estás escribiendo scripts o automatizando	Estás explorando o experimentando
El comando es corto y memorable	La tarea implica múltiples pasos

Una nota sobre la honestidad. Ya sea que escribas tu reseña manualmente o la redactes con ayuda de un asistente IA, las opiniones deben ser tuyas. Usa el agente para afinar tu pensamiento y manejar los pasos mecánicos, no para sustituir tu criterio. Las mejores reseñas tienen una voz humana. Un

agente puede ayudarte a encontrar las palabras para lo que ya sientes, pero no puede sentirlo por ti.

Stage y Commit

Una vez que tu reseña esté escrita y guardada, dile a Git que la rastree:

```
git add reviews/
```

Comprueba qué está en stage:

```
git status
```

Deberías ver tu archivo de reseña listado bajo «Changes to be committed».

Ahora haz commit: esto crea un snapshot con tus cambios y un mensaje que explica lo que hiciste:

```
git commit -m "Añadir reseña del Capítulo 1 por TU-NOMBRE-
DE-USUARIO-GITHUB"
```

Push a tu Fork

Envía tu rama a GitHub:

```
git push origin review/chapter-1-TU-NOMBRE-DE-USUARIO-
GITHUB
```

Tus cambios ahora existen tanto en tu máquina como en GitHub.

Crea el Pull Request

Este es el momento. Un pull request dice: «Hice algunos cambios en mi fork — ¿te gustaría integrarlos en el proyecto original?»

```
gh pr create --title "Reseña Capítulo 1: TU-NOMBRE-DE-
USUARIO-GITHUB" --body "Reseña honesta del Capítulo 1
(Introducción) de La Tripulación Agéntica. Incluye
primeras impresiones, puntos fuertes, áreas de mejora y
ajuste al público objetivo."
```

El CLI mostrará una URL. Ábrela en tu navegador: ese es tu pull request, en vivo en GitHub.

También puedes verlo en cualquier momento:

```
gh pr view --web
```

Qué Ocurre Después

Una vez que envíes tu PR:

1. **El autor lo lee** — el feedback honesto sobre un libro en progreso es genuinamente valioso
2. **Puede que comenten** — haciendo preguntas de seguimiento o agradeciéndote una observación específica
3. **Se integra** — tu reseña se convierte en parte permanente del historial del proyecto

Ese es el flujo de trabajo del código abierto: fork, rama, cambio, push, PR. Cada contribución a cada proyecto importante del mundo sigue este patrón. Tú acabas de hacerlo de verdad.

Solución de Problemas

`git push` pide contraseña: Ejecuta `gh auth setup-git` para configurar Git para usar las credenciales de tu CLI de GitHub. Funciona igual en todas las plataformas.

`open -e` no abre un editor de texto: TextEdit es el predeterminado. Para texto plano, prueba `open -a TextEdit chapters/01-introduction.typ` o simplemente usa `code` si tienes VS Code.

Cometiste un error tipográfico en el nombre de tu rama: Crea una nueva rama desde main: `git checkout main && git checkout -b review/chapter-1-nombre-corregido`, luego copia tu archivo de reseña.

El PR apunta a la rama incorrecta: Puedes especificar la base: `gh pr create --base main --title "..."`.

Referencia Rápida

Tarea	**Comando**
Ver tus ramas	`git branch`
Ver qué cambió	`git status`
Ver el diff	`git diff`
Obtener lo último del upstream	`git fetch upstream && git merge upstream/main`
Ver tu PR	`gh pr view --web`

Tu Primer Proyecto con IA

Hasta ahora has configurado tus herramientas y enviado tu primer pull request. Ahora vas a hacer algo genuinamente impresionante: tomar un proyecto real de código abierto, describir lo que quieres en español natural y dejar que un agente IA lo transforme, generando ilustraciones personalizadas, audio de narración y fondos animados con modelos de IA de Hugging Face.

No se requiere experiencia previa en programación. El agente escribe el código. Tú describes la visión.

One prompt. Three AI-generated assets. A complete presentation.

Lo que Vas a Construir

Web Presenter es un framework de presentaciones de código abierto: HTML, CSS y JavaScript puros, sin paso de compilación. Admite narración diapositiva por diapositiva (archivos WAV), fondos animados con Three.js y transiciones CSS suaves. Puedes ejecutarlo en cualquier navegador con un solo comando.

Al final de este capítulo, habrás:

1. Hecho un fork y clonado el proyecto
2. Elegido un tema para tu propia presentación
3. Pedido a un agente IA que genere ilustraciones personalizadas para cada diapositiva
4. Pedido a un agente IA que genere narración hablada para cada diapositiva
5. Personalizado el fondo animado para que coincida con tu tema
6. Visto el resultado final en tu navegador

Cada pieza de contenido generado por IA: las imágenes, el audio y los colores de la animación, provendrá de la API de inferencia gratuita de Hugging Face. Hugging Face aloja cientos de modelos de IA abiertos que cualquiera puede usar sin necesidad de tarjeta de crédito. Te comunicarás con ella a través de tu agente de programación IA.

Lo que Necesitarás

De los capítulos anteriores:

- Git instalado y configurado
- CLI de GitHub autenticado
- Claude Code o Gemini CLI listo en tu terminal

Nuevo para este capítulo:

- Una cuenta de Hugging Face (gratuita)
- Un token de API de Hugging Face
- Python 3 (para ejecutar un servidor local y llamar a la API de Hugging Face)
- La biblioteca Python `huggingface_hub` (un solo comando `pip install`, que se cubre a continuación)

Obtén una Cuenta y Token de Hugging Face

Si no tienes una, regístrate en huggingface.co/join. Es gratuita: no se requiere tarjeta de crédito.

Una vez que hayas iniciado sesión:

1. Haz clic en tu foto de perfil (arriba a la derecha) → **Settings**
2. Haz clic en **Access Tokens** en la barra lateral izquierda
3. Haz clic en **New token**
4. Ponle un nombre como `agentic-crew` y selecciona acceso de **Read**
5. Haz clic en **Generate a token** y cópialo

Guarda este token a mano: lo pegarás en tu terminal en el siguiente paso.

Comprueba Python

Ejecuta:

```
python3 --version
```

Deberías ver Python 3.8 o superior. Si Python no está instalado:

```
brew install python
```

Cierra y vuelve a abrir tu terminal tras la instalación, luego ejecuta `python3 --version` de nuevo para confirmar.

Instala la Biblioteca de Hugging Face
Los scripts que escribirá el agente usarán la biblioteca oficial de Python de Hugging Face. Instálala ahora:

```
pip install huggingface_hub
```

En algunos sistemas puede que necesites `pip3` en lugar de `pip`. Solo tienes que hacer esto una vez.

Configura tu Token de Hugging Face

Tu agente IA usará este token para llamar a la API de Hugging Face. Configúralo como variable de entorno para que el agente pueda acceder a él sin codificarlo en ningún archivo.

```
export HF_TOKEN="tu-token-aquí"
```

Reemplaza `tu-token-aquí` con el token que copiaste. Ahora verifica que se configuró:

```
echo $HF_TOKEN
```

Deberías ver tu token impreso. Si no aparece nada, vuelve a ejecutar el comando `export`: la variable no se guardó.

> **Mantén los tokens fuera de los archivos.** Nunca pegues tu token de API directamente en el código ni lo incluyas en un commit de Git. Las variables de entorno lo mantienen solo en memoria: sin riesgo de enviarlo accidentalmente a un repositorio público.

Esta configuración dura tu sesión de terminal actual. Para hacerla permanente, añade la línea `export` a tu archivo de configuración del shell (`~/.zshrc`).

Fork y Clone del Proyecto

Este único comando crea tu propia copia del proyecto en GitHub y la descarga a tu máquina:

```
gh repo fork schlunsen/web-presenter --clone --remote
```

Entra en el directorio del proyecto:

```
cd web-presenter
```

Verifica tus conexiones con GitHub:

```
git remote -v
```

Deberías ver dos entradas:

- `origin` — tu fork (donde envías tus cambios)
- `upstream` — el proyecto original (de donde puedes obtener actualizaciones)

Cada remoto aparece listado dos veces: una para fetch y otra para push. Cuatro líneas en total es correcto. Si solo ves `origin`, algo salió mal: abre tu asistente IA y describe lo que ves; te ayudará.

Explora el Proyecto

Pide al agente que te explique con qué estás trabajando. Abre tu asistente IA desde dentro del directorio del proyecto.

Si instalaste Claude Code, ejecuta:

```
claude
```

Si instalaste Gemini CLI, ejecuta:

```
gemini
```

Cualquiera funciona para todo en este capítulo. Una vez abierto, pregunta:

- *«Lee index.html, presentation-script.js y presentation--styles.css. Explica cómo funciona este framework de presentaciones: cómo se estructuran las diapositivas, cómo se carga la narración de audio y cómo funciona el fondo animado.»*

El agente leerá los archivos y te dará un resumen en lenguaje sencillo. Entenderás el proyecto en dos minutos en lugar de treinta. Solo está leyendo en este punto: nada se está cambiando todavía.

Una vez que tengas una idea general, pregunta sobre los assets existentes:

- *«Lista todos los archivos en presentation-audio/ y presentation-images/. ¿Qué hay ahí ya?»*

Verás los clips de narración e imágenes existentes: marcadores de posición que estás a punto de reemplazar con los tuyos.

Elige tu Tema

Elige algo que conozcas o que te importe. La presentación tiene diez diapositivas, así que quieres un tema con suficiente contenido para diez puntos cortos. Algunas ideas:

- Una tecnología que usas en el trabajo
- Un hobby o habilidad que quieres explicar a un amigo
- Un proyecto que estás construyendo
- Un argumento que quieres defender sobre algo

Para el resto de este capítulo, usaremos el marcador de posición [**TU TEMA**]: reemplázalo con lo que hayas elegido.

Anota diez puntos cortos: los temas de tus diapositivas. Una oración cada uno. Los pegarás en los prompts del agente que siguen.

Haz que tus puntos sean visuales. Cada uno se convertirá en una imagen generada por IA, así que lo específico y concreto funciona mejor que lo abstracto. «Un almacén lleno de filas de servidores» generará una mejor ilustración que «infraestructura tecnológica». «Un niño leyendo bajo un árbol al atardecer» es mejor que «educación».

Genera las Ilustraciones

En este paso pedirás al agente que escriba un script de Python. No lo escribirás tú: el agente lo hará. Tu trabajo es proporcionar tus diez puntos y luego pedir al agente que ejecute el script. Todo lo demás es automático.

Abre tu asistente IA (si no está ya abierto) y dale este prompt. **No lo copies palabra por palabra**: reemplaza `[TU TEMA]` con tu tema real y `[pega tus puntos]` con tus diez oraciones:

- *«Escribe un script de Python usando la biblioteca huggingface_hub. Usa InferenceClient con el token de la variable de entorno HF_TOKEN. Llama a client.text_to_image() con el modelo black-forest-labs/FLUX.1-schnell para generar una imagen por diapositiva y guarda cada resultado en presentation-images/ como slide-01.jpg, slide-02.jpg, etc. Aquí están los diez temas de las diapositivas: [pega tus puntos]. Haz que los prompts sean vívidos y consistentes en estilo.»*

El agente escribirá el script. Revísalo: no necesitas entender cada línea, pero comprueba que lea el token del entorno (`os.environ` o similar) en lugar de tenerlo codificado en el código.

Cuando estés conforme, pide al agente que lo ejecute:

- *«Ejecuta el script.»*

La generación tarda aproximadamente 10-20 segundos por imagen. El agente informará del progreso a medida que se complete cada una. Cuando termine, la carpeta `presentation-images/` contendrá diez nuevos archivos.

> **¿Y si una imagen no queda bien?** Pide al agente que regenere solo esa: *«La ilustración de la diapositiva 3 no queda bien: debería mostrar [descripción]. Regénera solo esa con un prompt mejor.»* La generación de imágenes es iterativa. Un segundo o tercer intento suele acercarse más a lo que tenías en mente.

Genera la Narración

A continuación: audio hablado para cada diapositiva. Hugging Face aloja modelos de texto a voz que convierten texto en un archivo de audio. El audio sonará claro e inteligible: no del todo humano, pero suficientemente natural para una presentación.

Dale al agente este prompt, reemplazando los marcadores con tu texto de narración real:

- *«Escribe un script de Python usando la biblioteca huggingface_hub. Usa InferenceClient con el token de la variable de entorno HF_TOKEN. Llama a client.text_to_speech() con el modelo facebook/mms-tts-eng para generar narración para cada diapositiva. Guarda cada resultado en presentation-audio/ como slide-01.wav hasta slide-10.wav. Aquí están los textos de narración para cada diapositiva: [pega tus descripciones de una oración].»*

El agente escribirá el script. Ejecútalo del mismo modo:

- *«Ejecuta el script de narración.»*

La generación de audio es más rápida que la de imágenes: espera unos pocos segundos por clip. Cuando termine, `presentation-audio/` tendrá diez archivos `.wav` listos para reproducir.

> **¿Quieres una voz más natural?** Pide al agente que pruebe `suno/bark-small` en su lugar: misma biblioteca, modelo diferente, más lento pero más expresivo. Solo pregunta: *«Reescribe el script de narración para usar el modelo suno/bark-small y ejecútalo de nuevo.»*

Actualiza la Presentación

`index.html` es el archivo principal del que lee tu presentación: contiene todo el contenido de las diapositivas, las rutas de imágenes y los nombres de archivos de audio. Ahora el agente lo actualizará con tu nuevo contenido.

- *«Actualiza index.html para reemplazar las diapositivas existentes con mis diez diapositivas sobre [TU TEMA]. Cada diapositiva debe usar la ilustración .jpg correspondiente de presentation-images/ y la narración .wav correspondiente de presentation-audio/. Usa los diseños de diapositivas existentes: diseño de título para la diapositiva 1, dos columnas o centrado para el resto. Mantén la estructura HTML exactamente como está; solo actualiza el contenido y las referencias a los assets.»*

El agente hará las ediciones. Cuando termine, pídele que compruebe su propio trabajo:

- *«Lee index.html de nuevo y confirma que las diez diapositivas están presentes, cada una con la imagen y el archivo de audio correctos.»*

Esta autocomprobación detecta referencias que faltan antes de que abras el navegador.

Personaliza el Fondo Animado

El fondo Three.js dibuja una red de nodos y líneas de conexión. Por defecto usa azules y grises fríos. Pide al agente que lo adapte al ambiente de tu tema:

- *«Actualiza presentation-bg.js para cambiar los colores de la animación de la red a [describe tu paleta: por ejemplo, "ámbar cálido y marrón oscuro", "verde profundo y blanco suave" o "azul eléctrico sobre negro"]. También actualiza las variables de color CSS en presentation-styles.css para que coincidan.»*

El agente editará ambos archivos. Si el resultado no se siente bien, describe lo que quieres con más precisión:

- *«El fondo es demasiado brillante. Haz los nodos más pequeños y reduce la opacidad de las líneas al 30%.»*

Itera tantas veces como necesites. Cada cambio tarda unos segundos.

Vista Previa en tu Navegador

Antes de abrir el navegador, confirma tres cosas:

1. La carpeta `presentation-images/` contiene 10 archivos `.jpg` (slide-01.jpg hasta slide-10.jpg)
2. La carpeta `presentation-audio/` contiene 10 archivos `.wav` (slide-01.wav hasta slide-10.wav)
3. La terminal que usaste para ejecutar los scripts de generación sigue activa en el directorio `web-presenter`

Si falta algún archivo, pregunta al agente: *«Comprueba las carpetas presentation-images/ y presentation-audio/. ¿Qué archivos hay y cuáles faltan?»*

Web Presenter necesita un servidor HTTP local: usa `fetch()` para cargar archivos de audio, lo que los navegadores bloquean para rutas `file://` simples. Pide al agente que inicie uno:

- *«Inicia un servidor HTTP local en este directorio en el puerto 8000.»*

Ejecutará:

```
python3 -m http.server 8000
```

Deberías ver una salida como:

```
Serving HTTP on 0.0.0.0 port 8000
(http://0.0.0.0:8000/) ...
```

Eso significa que el servidor está listo. **Deja esta ventana de terminal abierta**: cerrarla detiene el servidor. Ahora abre tu navegador y ve a:

```
http://localhost:8000
```

Deberías ver tu presentación. Presiona la barra espaciadora o la flecha derecha para avanzar a la siguiente diapositiva. Cada diapositiva mostrará su ilustración, reproducirá la narración y avanzará automáticamente cuando el audio termine. También puedes presionar la flecha derecha para saltar manualmente.

Presiona `M` para activar/desactivar la música de fondo, `A` para activar/desactivar la narración y `Esc` para detener la reproducción.

Cuando hayas terminado, vuelve a la terminal y presiona **Ctrl+C** para detener el servidor.

Commit y Push

Una vez que estés satisfecho con tu presentación, guarda tu trabajo en GitHub:

- *«Crea una rama llamada presentation/[TU TEMA], añade todos los archivos modificados, haz commit con un mensaje que describa lo que construí y envíalo a mi fork.»*

El agente se encargará de cada paso de Git. Cuando termine, verifica en GitHub:

1. Ve a `https://github.com/TU_USUARIO/web-presenter`
2. Abre el selector de ramas y busca tu rama (`presentation/[TU TEMA]`)
3. Haz clic en ella: deberías ver tus nuevas imágenes, archivos de audio e `index.html` actualizado

Tu presentación ya está guardada en GitHub y es compartible con cualquiera a través de esa URL.

Lo que Acaba de Pasar

Generaste diez ilustraciones de IA, las narraste, personalizaste gráficos 3D animados y actualizaste un proyecto web, todo describiendo lo que querías en español natural. El agente se encargó de la mecánica. Así es como se ve la programación agéntica.

Tú aportaste	**El agente aportó**
Un tema que te importa	Llamadas a la API de Hugging Face
Diez oraciones de contenido	Un script de Python funcional
Una descripción de paleta de colores	JavaScript y CSS actualizados

El criterio sobre qué queda bien	La mecánica para hacerlo realidad

La habilidad no es programar. Es saber qué pedir, cómo comprobar el resultado y cómo iterar cuando no es del todo correcto. Eso es lo que desarrollarán los próximos capítulos.

Solución de Problemas

La generación de imágenes devuelve un error sobre el modelo demasiado ocupado: La API de inferencia gratuita tiene límites de velocidad. Espera 30 segundos e inténtalo de nuevo, o pide al agente que añada un retraso: *«Añade una pausa de 10 segundos entre cada llamada de generación de imágenes.»*

Los archivos de audio están en silencio o no se reproducen: El modelo TTS devuelve archivos WAV, no MP3. Si index.html referencia extensiones `.mp3`, pide al agente: *«Comprueba si los atributos src de audio en index.html usan extensiones .wav. Actualiza los que no coincidan con los archivos en presentation-audio/.»* También confirma el tamaño de los archivos: cualquier archivo `.wav` de menos de 5 KB probablemente es una respuesta de error que necesita regenerarse.

Las diapositivas muestran iconos de imagen rota: La ruta de la imagen en el HTML no coincide con el nombre de archivo real. Pregunta: *«Comprueba todos los atributos src de imágenes en index.html contra los archivos reales en presentation-images/. Corrige cualquier discrepancia.»*

Mi HF_TOKEN no lo encuentra el script: La variable se configuró en una sesión de terminal diferente. Configúrala de

nuevo en la sesión actual (`export HF_TOKEN="..."`), y luego vuelve a ejecutar el script.

No carga nada o la página está en blanco: Asegúrate de que el servidor se está ejecutando en el directorio `web-presenter`, no en una carpeta superior. Abre las herramientas de desarrollo de tu navegador (F12) y revisa la pestaña Consola: los archivos que faltan aparecen listados ahí. Dile al agente lo que ves y lo arreglará.

La animación Three.js ha dejado de funcionar después de editar: Pregunta al agente: *«La animación del fondo ha dejado de funcionar. Lee presentation-bg.js y comprueba si hay errores de sintaxis o llamadas a funciones que faltan.»*

La rama incorrecta aparece en GitHub después del push: Asegúrate de que el nombre de la rama no tenga espacios: usa guiones. Pregunta al agente: *«¿Qué rama enviamos? Muéstrame la salida de git branch -a.»*

Si encuentras un error que no aparece aquí, descríbeselo a tu agente IA: ha visto la mayoría de los errores antes y generalmente sabrá qué hacer.

Referencia Rápida

Tarea	**Prompt o comando**
Verificar que el token HF está configurado	`echo $HF_TOKEN`
Explorar el proyecto	*«Lee index.html y explica cómo funcionan las diapositivas»*

Instalar la biblioteca HF	`pip install huggingface_hub`
Generar imágenes	*«Usa InferenceClient.text_to_image() con FLUX.1-schnell…»*
Generar narración	*«Usa InferenceClient.text_to_speech() con facebook/mms-tts-eng…»*
Actualizar contenido de diapositivas	*«Actualiza index.html con mis diez diapositivas…»*
Cambiar fondo	*«Actualiza los colores de la animación a…»*
Comprobar archivos generados	*«Lista los archivos en presentation-images/ y presentation-audio/»*
Iniciar servidor local	`python3 -m http.server 8000`
Ver presentación	`http://localhost:8000`
Detener el servidor	`Ctrl+C` en la terminal del servidor

La mejor manera de aprender
es publicar algo real.
Este libro es tu primer commit.