

Cuaderno de prácticas

Claude Code para perfiles técnicos · 6 horas · Caso Cafès Roca

Este cuaderno recoge todas las prácticas de las dos sesiones, paso a paso. Todas giran alrededor del mismo negocio ficticio, **Cafès Roca**, una pequeña cadena de cafeterías de Barcelona con tres locales. En la carpeta de materiales encontrarás la web y los datos de la sesión 1 (**web-cafes-roca**), la aplicación de stock de la sesión 2 (**estoc-cafes-roca**) y, en **solucions/**, el estado final de cada práctica.

Cada práctica tiene un objetivo, los pasos, los prompts y los comandos de referencia y lo que debes comprobar antes de darla por terminada. Los prompts son orientativos: adáptalos, pero mantén las cuatro piezas de una buena petición (contexto, instrucción, formato y cómo se comprueba). Los archivos del caso están en catalán, como el negocio.

Antes de empezar

Necesitas una cuenta de Claude con plan Pro o superior, una cuenta gratuita de GitHub y Git, Node.js y GitHub CLI instalados. La guía de preparación está en el apartado «Preparación previa», al final del cuaderno.

Contenido

- **Sesión 1 · Crear con Claude Code:** Claude Code: el agente para proyectos de software · Puesta a punto y primera sesión · Una landing page a partir de un brief · Análisis técnico de datos · Un dashboard interactivo · Contexto, coste y buenas prácticas
- **Sesión 2 · Repositorios, calidad y automatización:** Git y GitHub con Claude Code · Trabajar en paralelo y revisar · Personalizar el agente · Conectar herramientas: MCP, conectores y plugins · Automatizar y publicar · Criterio
- **Consulta · Preparación, comandos y fuentes:** Resultados de aprendizaje · Preparación previa · Comandos de referencia · Glosario · Documentación oficial

Sesión 1 · Crear con Claude Code

Bloques 1–6 · landing, datos y dashboard

Bloque 1 · Claude Code: el agente para proyectos de software

Dónde vive, cómo trabaja, qué modelos usa y qué incluye cada plan.

Claude Code es el agente de programación de Anthropic. Lee un proyecto entero, hace un plan, escribe y edita archivos, ejecuta comandos y pruebas, y trabaja con Git y GitHub. Tú decides qué puede hacer solo y qué te tiene que preguntar.

Dónde lo usas	Cómo se abre	Cuándo lo usaremos en el curso
Terminal	Escribes claude dentro de la carpeta del proyecto	Es el entorno de referencia de todas las prácticas
Pestaña Code de la aplicación de escritorio	Aplicación de Claude › Code › eliges la carpeta	Sesiones en paralelo, vista previa y revisión de cambios (bloque 8)
VS Code y JetBrains	Extensión de VS Code o conector de JetBrains	Si ya trabajas con uno de estos editores
Navegador	claude.ai/code , sesiones en la nube	Tareas largas que continúan sin tu ordenador (bloque 11)

El bucle del agente

Todas las prácticas del curso siguen el mismo método. Es el que recomienda la guía de buenas prácticas de Anthropic y es el que hace que el resultado sea fiable.

- **Explorar.** Lee los archivos y entiende qué hay antes de tocar nada.
- **Planificar.** Propone un plan que puedes revisar y corregir. Con el modo Plan no modifica nada.
- **Implementar.** Escribe el código, ejecuta comandos y corrige lo que falla.
- **Verificar.** Ejecuta pruebas, abre la página y compara las cifras. Tú decides si está bien.
- **Guardar.** Un commit con un mensaje claro, para que cada paso se pueda recuperar.

Modelos y esfuerzo

Alias	Modelo	Cuándo usarlo
sonnet	Sonnet 5.5	El trabajo de cada día: rápido y bueno en código. Es el que usaremos por defecto
opus	Opus 5.5	Planes, cambios grandes y errores difíciles
opusplan	Opus para planificar, Sonnet para ejecutar	Tareas largas en las que el plan es la parte delicada
haiku	Haiku 4.5	Tareas pequeñas y repetitivas
fable	Fable 5.1	El modelo más capaz, con un límite de uso propio

El modelo se cambia con **/model** y el esfuerzo con **/effort** (**low**, **medium**, **high**, **xhigh** y **max**). Más esfuerzo significa que el agente piensa más antes de actuar: más calidad en tareas complejas, más tiempo y más consumo. Con Sonnet 5.5 y Opus 5.5 el esfuerzo por defecto es **medium**.

Planes y límites de uso, a 10 de octubre de 2026

	Pro	Max	Team y Enterprise
Precio	15 € al mes con pago anual o 18 € al mes, sin impuestos	Desde 90 € al mes	Por usuario, con administración
Claude Code	Terminal, aplicación de escritorio, editores y navegador	Lo mismo, con 5 o 20 veces el uso de Pro	Según la configuración de la organización
Límites	Ventanas de cinco horas y un máximo semanal	Más altos	Por usuario

Dónde se comprueba: los límites se comparten con el chat de Claude y Cowork. Puedes consultarlos con /usage dentro de Claude Code o en Configuración > Uso. Los precios actuales están en claude.com/pricing.

El agente se equivoca con seguridad

Claude Code escribe código que funciona la mayoría de las veces y, cuando se equivoca, lo hace con el mismo tono seguro. Por eso cada práctica termina con una comprobación que haces tú: pruebas, cifras, la página abierta en el navegador.

Práctica 1 · Primer contacto con el agente

Objetivo. Abrir Claude Code sobre la carpeta del caso, hacer que explore el proyecto sin modificar nada y probar el selector de modelo y de esfuerzo.

1. Descomprime `materials_claude_code_6h.zip` en el Escritorio. Abre la carpeta `web-cafes-roca` con VS Code (**File > Open Folder**) y abre un terminal (**Terminal > New Terminal**). En Windows también puedes hacer clic derecho en la carpeta > **Abrir en Terminal**.
2. Escribe `claude` y pulsa Intro. La primera vez te pregunta si confías en la carpeta: responde que sí.
3. Pulsa **Mayús+Tab** hasta que debajo del cuadro de texto aparezca **plan mode on**. En este modo el agente lee y piensa, pero no modifica nada.
4. Pega el prompt de exploración y lee la respuesta.
5. Escribe `/model` para ver el modelo activo y `/effort` para ver el nivel de esfuerzo. Deja `sonnet` y `medium`.
6. Hazle una pregunta concreta sobre el brief para comprobar que lo ha leído.

TERMINAL · ABRIR CLAUDE CODE

```
cd web-cafes-roca
claude
```

EXPLORAR EL PROYECTO

Explora esta carpeta y explícame qué hay: qué archivos, qué contiene cada uno y qué crees que tenemos que construir. No modifiques nada. Termina con tres preguntas que me harías antes de empezar.

PREGUNTA DE COMPROBACIÓN

Según el brief, ¿qué horarios tiene el local de Sants y qué colores tenemos que usar en la web?

Lo que deberías ver

El agente cita el brief, los tres artículos en Markdown y los datos de ventas y opiniones. Responde que Sants abre de lunes a viernes de 7:30 a 19:00 y que los colores son #3B2A1F, #F4EBDD y #C8553D. No ha creado ni modificado ningún archivo.

☐ **Práctica 1 hecha: Claude Code abierto sobre web-cafes-roca y proyecto explorado en modo Plan**

Documentación oficial: [Claude Code overview](#) · [How Claude Code works](#) · [Model configuration](#) · [Using Claude Code with your Pro or Max plan](#)

Bloque 2 · Puesta a punto y primera sesión

Comprobar el entorno, la memoria del proyecto y los permisos.

Antes de construir, tres cosas: que el entorno funciona, que el agente conoce las normas del proyecto y que sabes qué permisos tiene y cómo se deshace un cambio.

Práctica 2.1 · Comprobar el entorno

Objetivo. Confirmar que Claude Code, Git, Node.js y GitHub CLI están instalados y que tienes la sesión iniciada.

1. Sal de Claude Code con `/exit` (o pulsa **Ctrl+C** dos veces).
2. Ejecuta los comandos de comprobación uno a uno.
3. Si `gh auth status` dice que no tienes sesión, ejecuta `gh auth login` y elige **GitHub.com › HTTPS › Login with a web browser**.
4. Si algún comando no se reconoce, vuelve a la guía de preparación (pestaña **Consulta**) e instala lo que falte.

TERMINAL · COMPROBACIONES

```
claude --version
claude doctor
git --version
node --version
gh --version
gh auth status
```

Windows: Claude Code usa Git para Windows para ejecutar comandos. Si `claude doctor` avisa de que no lo encuentra, instala Git desde git-scm.com y abre un terminal nuevo.

☐ **Todos los comandos responden y gh tiene la sesión iniciada**

CLAUDE.md: la memoria del proyecto

Claude Code lee al comienzo de cada sesión unos archivos de texto con instrucciones. El principal es **CLAUDE.md**, en la raíz del proyecto. En él pones cómo se ejecuta el proyecto, las normas y lo que el agente no puede deducir leyendo el código.

Archivo	Alcance	¿Se comparte con el equipo?
<code>~/.claude/CLAUDE.md</code>	Todas tus sesiones, en cualquier proyecto	No
<code>./CLAUDE.md</code>	Este proyecto	Sí, va al repositorio
<code>./CLAUDE.local.md</code>	Este proyecto, solo para ti	No, se excluye de Git

Archivo	Alcance	¿Se comparte con el equipo?
<code>./.claude/rules/*.md</code>	Normas para carpetas o tipos de archivo concretos	Sí

Además, Claude Code tiene una **memoria automática**: apunta lo que aprende de ti y del proyecto (comandos, preferencias, correcciones) y lo recupera en las sesiones siguientes. Todo se revisa y se edita con `/memory`. Un buen **CLAUDE.md** es corto: menos de 200 líneas, concreto y verificable.

Práctica 2.2 · CLAUDE.md con /init

Objetivo. Generar el CLAUDE.md de la web y añadirle las normas del proyecto.

1. Vuelve a abrir `claude` en la carpeta `web-cafes-roca`.
2. Escribe `/init`. Claude Code analiza la carpeta y propone un **CLAUDE.md**. Acéptalo.
3. Abre **CLAUDE.md** en VS Code y léelo: ¿qué ha entendido bien y qué no?
4. Pega el prompt de normas para que las añada.
5. Escribe `/memory` para ver qué archivos de memoria tiene cargados.

NORMAS DEL PROYECTO

Añade a CLAUDE.md estas normas del proyecto:

- Web estática en HTML, CSS y JavaScript, sin frameworks ni dependencias de npm.
 - Todos los textos de la web en catalán. Colores y tono del brief.
 - Tiene que funcionar bien en el móvil.
 - Antes de dar una tarea por terminada, abre la página y compruébala a 390 px de ancho y en escritorio.
 - Mensajes de commit en catalán, en imperativo y cortos: «Afegeix el formulari de càtering».
- Mantén el archivo por debajo de 60 líneas.

☐ **CLAUDE.md creado y con las normas del proyecto**

Modos de permiso

Modo	Qué hace sin preguntar	Cuándo lo usamos
Manual (<code>default</code>)	Solo leer. Pide permiso para editar y para ejecutar comandos	Cuando empiezas en un proyecto que no conoces
Accept edits (<code>acceptEdits</code>)	Leer y editar archivos. Pide permiso para los comandos	Implementar un plan que ya has revisado
Plan (<code>plan</code>)	Leer y proponer un plan. No modifica nada	Explorar y planificar
Auto (<code>auto</code>)	Un clasificador revisa cada acción y detiene las arriesgadas	Tareas largas cuando la cuenta lo tiene disponible

En el terminal los cambias con **Mayús+Tab**. En la aplicación de escritorio, con el selector de modo del cuadro de texto. También puedes empezar directamente en un modo: `claude --permission-mode plan`. Hay un modo que se lo salta todo (`bypassPermissions`); en el curso no lo usaremos.

Puntos de control y /rewind

Cada vez que el agente edita archivos, Claude Code guarda un punto de control. Pulsa **Esc** dos veces con el cuadro de texto vacío, o escribe `/rewind`, para volver atrás: solo el código, solo la conversación o ambas cosas. Los cambios hechos con comandos de terminal (por ejemplo, borrar un archivo con `rm`) no quedan en el punto de control. Por eso, además del `/rewind`, haremos commits con Git.

Práctica 2.3 · Probar los permisos y deshacer un cambio

Objetivo. Ver cómo pide permiso el agente en modo Manual y recuperar el estado anterior con `/rewind`.

1. Con **Mayús+Tab**, pon el modo Manual: debajo del cuadro de texto aparece **manual mode on**.
2. Pega el prompt de prueba. Cuando te pida permiso para crear el archivo, acéptalo solo esta vez.
3. Comprueba en VS Code que `prova.html` existe.
4. Pulsa **Esc** dos veces, elige el mensaje de la prueba y la opción **Restore code**.
5. Comprueba que `prova.html` ya no está.

ARCHIVO DE PRUEBA

Crea un archivo `prova.html` con un título grande que diga «Prova de Cafès Roca» y nada más.

☐ Archivo creado con permiso y deshecho con `/rewind`

Documentación oficial: [Set up Claude Code](#) · [Manage Claude's memory](#) · [Permission modes](#) · [Checkpointing](#) · [Best practices for Claude Code](#)

Bloque 3 · Una landing page a partir de un brief

Del brief al plan, del plan a la página, y de la página a un commit.

Hacemos la web de Cafès Roca con el método completo. El brief dice qué queremos; el modo Plan convierte el brief en un plan que revisas antes de que el agente escriba una sola línea.

Por qué primero el plan

Corregir un plan cuesta una frase. Corregir una página hecha con una idea equivocada cuesta una sesión entera. El modo Plan hace que el agente pregunte lo que no está claro antes de empezar.

Práctica 3 · La landing de Cafès Roca

Objetivo. Construir una landing adaptada al móvil a partir de `brief_web_cafes_roca.txt`, convertir los artículos en Markdown en páginas de la web y guardarlo todo en un repositorio de Git.

1. Pon el modo Plan (**Mayús+Tab**) y pega el prompt del plan.
2. Responde a las preguntas que te haga. Lee el plan y, si hace falta, pide cambios antes de aprobarlo.
3. Aprueba el plan eligiendo la opción de aceptar las ediciones. El agente crea `index.html` y `estils.css`.
4. Abre `index.html` con doble clic. Usa las herramientas de desarrollador del navegador (**F12**, icono del móvil) para verla a 390 px.
5. Haz una captura de una parte que quieras mejorar, guárdala en la carpeta como `captura.png` y pega el prompt de iteración. También puedes pegar la imagen directamente en el terminal: **Alt+V** en Windows, **Ctrl+V** en macOS.
6. Pega el prompt de los artículos. El agente escribe `scripts/articles.mjs`; ejecútalo tú con `node scripts/articles.mjs`.
7. Pega el prompt del commit.

PLAN DE LA LANDING

Lee `brief_web_cafes_roca.txt` y los tres artículos de la carpeta `articles/`. Quiero la landing de Cafès Roca: un `index.html` y un `estils.css`, sin dependencias. Secciones del brief en este orden: cabecera con lema y botón al catering, los tres locales, carta destacada, una sección «Del nostre blog» que de momento quede vacía, formulario de catering y pie.

El formulario no se envía a ningún sitio: al enviarlo, valida los campos y muestra un mensaje de confirmación en la misma página.

Hazme primero las preguntas que necesites y después un plan con la estructura de archivos y las decisiones de diseño.

ITERACIÓN CON UNA CAPTURA

Mira `captura.png`. Es la sección de la carta vista en el móvil. Los precios quedan demasiado pegados al nombre del plato y la línea de puntos no se alinea. Arréglalo para que se lea bien a 390 px y comprueba que en escritorio no ha cambiado nada.

ARTÍCULOS EN MARKDOWN

Crea `scripts/articles.mjs`, en `Node.js` y sin dependencias, que:

- lea cada `.md` de `articles/` con la cabecera entre `---` (`titol`, `data`, `autora`, `resum`);
- convierta el Markdown a HTML: títulos, párrafos, listas, negrita, enlaces y tablas;
- genere una página por artículo en `blog/`, con la misma cabecera y estilos de la landing y un enlace para volver;
- llene la sección «Del nostre blog» de `index.html` con una tarjeta por artículo, del más nuevo al más antiguo.

No lo ejecutes tú: explícame cómo lo ejecuto.

PRIMER COMMIT

Inicializa un repositorio de Git en esta carpeta, crea un `.gitignore` adecuado para una web estática y haz el primer commit con todos los archivos. Mensaje en catalán.

Lo que deberías ver

Una página con los tres locales, la carta con los cinco precios del brief, tres tarjetas de artículos y el formulario, que se ve bien en el móvil. En `blog/` hay tres páginas. `git log` muestra el primer commit.

Si te atascas: en `solucions/web-p3-landing` está el estado final de esta práctica. Copia sus archivos en tu carpeta y continúa.

☐ **Práctica 3 hecha: landing, páginas del blog y primer commit**

Documentación oficial: [Common workflows](#) · [Best practices for Claude Code](#) · [Prompting best practices](#)

Bloque 4 · Análisis técnico de datos

Un script que se puede volver a ejecutar, gráficos y un informe con cada cifra trazable.

En el chat, Claude calcula las cifras en un entorno que desaparece cuando cierras la conversación. Con Claude Code, el cálculo queda en un script dentro de tu proyecto: lo puedes revisar, ejecutar el mes que viene con datos nuevos y enseñar a quien dude de una cifra.

	Análisis en el chat	Script con Claude Code
Dónde queda el cálculo	En la conversación	En <code>analisi/analisi.mjs</code> , dentro del proyecto

	Análisis en el chat	Script con Claude Code
Datos nuevos el mes que viene	Volver a pedirlo	<code>node analisi/analisi.mjs</code>
Quién lo puede revisar	Quien tenga la conversación	Cualquiera que tenga el repositorio
Comprobar una cifra	Preguntarla	Leer la línea de código que la calcula

Práctica 4 · Las ventas del semestre

Objetivo. Escribir un script reproducible que lea el CSV de ventas, saque las cifras clave, genere tres gráficos y redacte un informe en el que cada cifra diga de dónde sale.

1. Empieza una conversación nueva con `/clear`.
2. Pega el prompt del análisis en modo Plan y revisa el plan: comprueba que dice cómo leerá la coma decimal.
3. Apruébalo y deja que escriba el script.
4. Ejecútalo tú: `node analisi/analisi.mjs`.
5. Abre `analisi/informe.md` en VS Code con la vista previa (**Ctrl+Mayús+V**) y mira los gráficos.
6. Pega el prompt de la doble comprobación y el del acta.
7. Haz un commit.

SCRIPT DE ANÁLISIS

Analiza `dades/vendes_cafes_roca_2026.csv` (punto y coma, coma decimal; columnas `data`, `local`, `categoria`, `producte`, `unitats`, `import_eur`).
 Escribe `analisi/analisi.mjs` en Node.js, sin dependencias, que:

1. Lea y limpie el CSV: compruebe las columnas, descarte las filas no válidas y diga cuántas ha descartado.
2. Calcule: total del semestre; ventas y peso por local; ventas por mes y local; ventas y peso por categoría; peso de las bebidas frías sobre el total de cada mes; los cinco productos que más facturan.
3. Guarde todas las cifras en `analisi/resultats.json`.
4. Genere tres gráficos SVG en `analisi/grafics/`: evolución mensual por local, peso de cada categoría y bebidas frías por mes. Elige el tipo de gráfico que responda mejor a cada pregunta y explícame por qué.
5. Escriba `analisi/informe.md` con cinco conclusiones en lenguaje llano y, junto a cada cifra, la clave de `resultats.json` de donde sale.

Números en formato catalán: 104.827,00 €.

DOBLE COMPROBACIÓN

Comprueba el total del semestre por otro camino: suma las ventas por local y por mes y mira que las dos sumas den lo mismo que el total. Enséñame las tres cifras.

EL ACTA CONTRA LOS DATOS

El acta de la reunión del 8 de septiembre dice que el café es el 55 % de la facturación y que las bebidas frías representan el 14 % en verano. Compruébalo con `resultats.json` y explica la diferencia. No modifiques ningún archivo.

Cifras de comprobación

Total: **104.827,00 €** · Poblenou 42,4 %, Gràcia 32,0 %, Sants 25,6 % · café 30,1 % del total · bebidas frías 23,6 % en mayo y 27,4 % en junio · flat white, primer producto, con 10.293,00 €.

Python: si tienes Python instalado, puedes pedir el mismo script con pandas y matplotlib. En el curso usamos Node.js porque ya lo tienes listo y no hace falta instalar ninguna librería.

□ Práctica 4 hecha: script, resultats.json, tres gráficos e informe

Qué gráfico responde a cada pregunta

Pregunta	Gráfico	Por qué
¿Cómo evolucionan las ventas de cada local?	Líneas	Se lee la tendencia y se comparan locales de un vistazo
¿Qué peso tiene cada categoría?	Barras horizontales ordenadas	Los nombres largos caben y el orden ya dice quién gana
¿Cómo crecen las bebidas frías?	Barras por mes con el porcentaje	Lo que importa es la proporción de cada mes
¿Cuánto aporta cada local cada mes?	Barras apiladas	El total del mes y la parte de cada local, a la vez

Documentación oficial: [Common workflows](#) · [Node.js](#) · [File system](#) · [MDN](#) · [SVG tutorial](#)

Bloque 5 · Un dashboard interactivo

Especificar antes de construir: entrevista, SPEC.md, plan e implementación.

Para una pieza más grande, hacemos que el agente nos entreviste y escriba la especificación. Después empezamos una sesión nueva, con el contexto limpio, y construimos a partir del documento.

- **Entrevista.** El agente pregunta quién lo usará, qué decisiones tiene que permitir tomar y qué cifras tienen que coincidir.
- **SPEC.md.** Un documento corto con datos, contenido, diseño y criterios de aceptación.
- **Contexto limpio.** `/clear` y una sesión nueva que solo lee la SPEC. Las decisiones viajan en el documento.
- **Aceptación.** Los números del dashboard tienen que cuadrar con los del análisis.

Práctica 5 · El cuadro de mando de ventas

Objetivo. Especificar, construir y verificar un dashboard de un solo archivo HTML con filtros por local y por mes, a partir de los datos de ventas.

1. Pega el prompt de la entrevista y responde a las preguntas una a una.
2. Lee `dashboard/SPEC.md`. Si le falta algo, pide que lo añada.
3. Escribe `/clear`, pon el modo Plan y pega el prompt de construcción.
4. Revisa el plan y apruébalo.
5. Abre `dashboard/index.html` con doble clic y prueba los filtros.
6. Pega el prompt de verificación.
7. Haz un commit.

ENTREVISTA Y ESPECIFICACIÓN

Quiero un cuadro de mando de las ventas de Cafès Roca a partir de `dades/vendes_cafes_roca_2026.csv`.

Hazme una entrevista para definirlo: una pregunta cada vez, sobre quién lo usará, qué tiene que poder decidir, qué filtros, indicadores y gráficos necesita, y cómo sabremos que las cifras son correctas.

Cuando tengas suficiente información, escribe `dashboard/SPEC.md` con estas secciones: para quién y para qué, datos, contenido, diseño y criterios de aceptación. No escribas código todavía.

CONSTRUIR A PARTIR DE LA SPEC

Lee dashboard/SPEC.md y haz un plan para construirlo.

Restricciones: un solo archivo dashboard/index.html con CSS y JavaScript, sin librerías externas, gráficos en SVG. Para que se abra con doble clic sin servidor, genera los datos agregados en un dashboard/dades.js con un script scripts/dades-dashboard.mjs.

Cuando termines, comprueba tú mismo cada criterio de aceptación y dime el resultado.

VERIFICACIÓN

Compara el dashboard con analisi/resultats.json:

- filtro Tots, de enero a junio: ventas totales y peso de Poblenou;
- filtro Gràcia, de mayo a junio: ventas.

Dime si cuadran y, si no, cuál de las dos cifras es la correcta y por qué.

Lo que deberías ver

Con Tots y enero–junio, **104.827,00 €** y el flat white como primer producto. Con Gràcia y mayo–junio, **12.040,20 €**. Los filtros actualizan indicadores, gráficos y tabla a la vez.

Si te atascas: solucions/web-p5-dashboard tiene la SPEC, el script de datos y el dashboard terminado.

☐ **Práctica 5 hecha: SPEC.md, dashboard y cifras verificadas**

Documentación oficial: [Best practices: let Claude interview you](#) · [Common workflows](#)

Bloque 6 · Contexto, coste y buenas prácticas

Qué ocupa la memoria de la sesión, cuánto consumes y cómo trabajar para que dure.

La ventana de contexto es la memoria de trabajo de la sesión: en ella entran tus mensajes, los archivos que lee el agente y la salida de los comandos. Cuando se llena, el agente rinde menos y el límite de uso se acaba antes.

Comando	Qué hace	Cuándo
<code>/context</code>	Muestra qué ocupa el contexto	Cuando la sesión se alarga
<code>/compact [instrucciones]</code>	Resume la conversación y libera espacio	Al terminar una fase, indicando qué quieres conservar
<code>/clear</code>	Empieza una conversación nueva en el mismo proyecto	Al cambiar de tarea
<code>/usage</code>	Consumo de la sesión y límites del plan	Antes de una tarea larga
<code>/btw [pregunta]</code>	Pregunta lateral que no queda en el historial	Una duda rápida en medio de una tarea
<code>/resume</code>	Recupera una conversación anterior	Al día siguiente, para continuar

- **Una tarea por sesión.** Cuando terminas, `/clear`. El contexto de una tarea ensucia la siguiente.
- **Peticiones verificables.** Di cómo sabrás que está bien: una prueba, una cifra, una captura.
- **Para y corrige.** Si ves que va por mal camino, pulsa **Esc** y redirígelo. No esperes a que termine.
- **Lo que importa, en los archivos.** Las decisiones van a **CLAUDE.md**, a la SPEC o a un commit. La conversación se olvida.

Práctica 6 · Mirar y limpiar el contexto

Objetivo. Ver qué ocupa el contexto y cuánto has consumido, y aprender a compactar y empezar de nuevo.

1. Escribe `/context` y mira qué parte ocupan los archivos y cuál la conversación.
2. Escribe `/usage` y mira cómo vas de la ventana de cinco horas y de la semana.
3. Haz una pregunta lateral con `/btw`.
4. Compacta la conversación indicando qué quieres conservar.
5. Vuelve a mirar `/context` y compáralo.

PREGUNTA LATERAL

`/btw` ¿qué diferencia hay entre `dashboard/dades.js` y `scripts/dades-dashboard.mjs`?

COMPACTAR CON CRITERIO

`/compact` conserva solo las decisiones del dashboard, los archivos que hemos creado y las cifras de verificación

☐ **Práctica 6 hecha: contexto revisado, compactado y consumo consultado**

Para la sesión 2

Trae el proyecto `web-cafes-roca` con los commits hechos. Lo usaremos para publicar la landing en GitHub Pages.

Documentación oficial: [Manage costs effectively](#) · [Context window](#) · [Commands](#)

Sesión 2 · Repositorios, calidad y automatización

Bloques 7–12 · GitHub, personalización, MCP y criterio

Bloque 7 · Git y GitHub con Claude Code

Entender un proyecto existente, incidencias, ramas, pruebas y pull requests.

En la segunda sesión trabajamos como un equipo de desarrollo sobre la aplicación de stock de Cafès Roca: un proyecto que ya existe, con pruebas, tres incidencias abiertas y un error que ha detectado el responsable de Gràcia.

Palabra	Qué es	Comando que usará el agente
Repositorio	La carpeta del proyecto con todo el historial de cambios	<code>git init, gh repo create</code>
Commit	Una foto de los cambios con un mensaje	<code>git commit</code>
Rama	Una línea de trabajo separada de la principal (<code>main</code>)	<code>git switch -c</code>
Incidencia (issue)	Una tarea o un error apuntado en GitHub	<code>gh issue create, gh issue view</code>
Pull request	La propuesta de llevar una rama a <code>main</code> para que alguien la revise	<code>gh pr create</code>
Diff	Las líneas que han cambiado	<code>git diff, gh pr diff</code>

Práctica 7.1 · Conocer el proyecto y subirlo a GitHub

Objetivo. Hacer que el agente te explique un proyecto que no has escrito tú, comprobar que funciona y publicarlo en un repositorio privado con sus incidencias.

1. Abre un terminal en la carpeta `estoc-cafes-roca` y ejecuta `npm test` y `npm start`. Abre `http://localhost:3000` y elige el local de Gràcia. Detén el servidor con **Ctrl+C**.
2. Abre `claude` en modo Plan y pega el prompt de exploración.
3. Vuelve al modo Manual y pega el prompt para crear el repositorio. Acepta los comandos de `gh` cuando te los pida.
4. Pega el prompt de las incidencias.
5. Abre el repositorio en GitHub y comprueba que están las tres incidencias.

TERMINAL · ARRANCAR EL PROYECTO

```
cd estoc-cafes-roca
npm test
npm start
```

ENTENDER EL PROYECTO

Explícame este proyecto como si fuera mi primer día en el equipo: qué hace, cómo está organizado, cómo fluyen los datos desde `data/productes.json` hasta la página y qué reglas de negocio aplica. Después lee `ISSUES.md` y dime por dónde empezarías y por qué. No modifies nada.

REPOSITORIO EN GITHUB

Crea un repositorio privado en GitHub que se llame `estoc-cafes-roca` con GitHub CLI y sube la rama `main` con todo el historial. Comprueba antes que `.env` no se subirá.

INCIDENCIAS

Crea en GitHub las tres incidencias de `ISSUES.md`, con el título, el texto y la etiqueta de cada una (bug o enhancement). Después enséñame la lista con `gh issue list`.

☐ **Repositorio privado creado con tres incidencias**
Práctica 7.2 · Corregir un error empezando por una prueba

Objetivo. Resolver la incidencia 1 con el método de las pruebas primero: una prueba que falla, la corrección mínima, todas las pruebas en verde y una pull request.

1. Pega el prompt de la prueba. El agente lee la incidencia, crea la rama y escribe una prueba nueva.
2. Comprueba que la prueba **falla**. Si pasa, la prueba no reproduce el error: pídele que la revise.
3. Pega el prompt de la corrección.
4. Revisa el cambio con `git diff` antes de aceptar el commit.
5. Pega el prompt de la pull request y ábrela en GitHub.

LA PRUEBA QUE FALLA

Lee la incidencia 1 con `gh issue view 1`. Crea la rama `fix/reposicio-minim`. Escribe en `test/estoc.test.js` una prueba que reproduzca el error con la bebida de almendra (Beguda d'ametlla) de Gràcia (stock 6, mínimo 6, consumo semanal 9) y que compruebe también que la cantidad a pedir es de 18 litros. Ejecuta las pruebas y enséñame que la nueva falla. No toques todavía `src/`.

LA CORRECCIÓN

Ahora corrige el error en `src/estoc.js` con el cambio más pequeño posible. Vuelve a ejecutar todas las pruebas y haz un commit que explique el error y la corrección.

LA PULL REQUEST

Sube la rama y abre una pull request hacia `main` que cierre la incidencia 1. En la descripción: qué fallaba, qué has cambiado, cómo se ha probado y el resultado de `npm test`.

Lo que deberías ver

La prueba nueva falla con el código original. La corrección es un solo carácter: `<` pasa a `<=` en `necessitaReposicio`. Después, 11 pruebas en verde y, en Gràcia, la bebida de almendra aparece en la lista con 18 l por pedir.

Si te atascas: `solucions/estoc-p7-error-corregit` tiene la prueba y la corrección.

☐ **Práctica 7 hecha: prueba que falla, corrección y pull request abierta**

Documentación oficial: [Common workflows](#) · [GitHub CLI manual](#) · [Node.js test runner](#)

Bloque 8 · Trabajar en paralelo y revisar

Sesiones simultáneas en worktrees, revisión automática y revisión humana.

Un worktree es una copia de trabajo del mismo repositorio en otra carpeta y en otra rama. Permite tener dos sesiones de Claude Code trabajando a la vez sin pisarse.

Dónde	Cómo se abre una sesión en un worktree
Terminal	<code>claude -w exportar-csv</code> crea <code>.claude/worktrees/exportar-csv</code> en la rama <code>worktree-exportar-csv</code>
Aplicación de escritorio	Pestaña Code > sesión nueva > opción de worktree
Subagentes	<code>isolation: worktree</code> en la definición del subagente

Práctica 8 · Dos incidencias a la vez y una revisión

Objetivo. Implementar la incidencia 2 en un worktree mientras la pull request de la 1 espera revisión, revisarlas con el agente y con tus propios ojos, y fusionarlas.

1. Abre un segundo terminal en `estoc-cafes-roca`, vuelve a `main` con `git switch main` y escribe `claude -w exportar-csv`.
2. Pega el prompt de la incidencia 2 y deja que el agente trabaje en ella.
3. Mientras tanto, en el primer terminal, escribe `/code-review` seguido del número de la pull request de la incidencia 1. En GitHub, incidencias y pull requests comparten numeración: si has creado tres incidencias, la primera pull request es la 4. La revisión se ejecuta en segundo plano y el resultado llega cuando termina.
4. Cuando la incidencia 2 esté hecha y tenga su pull request, pásale `/security-review` desde el worktree.
5. Revisa tú el diff en GitHub (pestaña **Files changed**) o con `gh pr diff`.
6. Fusiona las dos pull requests. El agente te tiene que pedir confirmación antes.

TERMINAL 2 · SESIÓN EN UN WORKTREE

```
git switch main
claude -w exportar-csv
```

INCIDENCIA 2 EN EL WORKTREE

Implementa la incidencia 2 (gh issue view 2): endpoint `/api/reposicio.csv` con el filtro `?local=`, botón «Descarga CSV» en la página y pruebas de la función que genera el CSV. Punto y coma, coma decimal y UTF-8 con BOM. Cuando termines, ejecuta `npm test`, haz commit y abre una pull request que cierre la incidencia 2.

REVISIÓN AUTOMÁTICA

```
/code-review 4
```

FUSIONAR

Fusiona las pull requests de las incidencias 1 y 2 con squash, en este orden. Si la segunda tiene conflictos con la primera, resuélvelos, vuelve a ejecutar las pruebas y enséñame el resultado antes de fusionar.

Lee el diff antes de fusionar

`/code-review` y `/security-review` encuentran errores, casos límite y riesgos que se te escaparían. La decisión de fusionar es tuya: lee el diff, ejecuta la aplicación y comprueba que hace lo que pedía la incidencia.

Si te atascas: `solucions/estoc-p8-exportar-csv` tiene el CSV implementado y probado.

□ Práctica 8 hecha: dos pull requests revisadas y fusionadas

Documentación oficial: [Run parallel sessions with Git worktrees](#) · [Claude Code on desktop](#) · [Commands \(/code-review, /security-review\)](#)

Bloque 9 · Personalizar el agente

CLAUDE.md de equipo, normas por carpetas, habilidades, subagentes, hooks y permisos.

Todo lo que hemos repetido en los prompts de la sesión se puede dejar por escrito para que el agente lo haga siempre igual, y para que lo haga igual para todo el equipo.

Pieza	Dónde vive	Para qué
CLAUDE.md	Raíz del proyecto	Comandos, estructura, reglas de negocio y cómo trabaja el equipo
Normas por carpetas	<code>.claude/rules/*.md</code> con <code>paths:</code>	Instrucciones que solo se aplican a algunos archivos, por ejemplo las pruebas
Habilidad (skill)	<code>.claude/skills/<nombre>/SKILL.md</code>	Una tarea que se repite, con su procedimiento. Se invoca con <code>/nombre</code> o el agente la usa cuando toca
Subagente	<code>.claude/agents/<nombre>.md</code>	Un especialista con su contexto, sus herramientas y su modelo
Hook	<code>.claude/settings.json</code> › <code>hooks</code>	Un comando que se ejecuta siempre en un momento concreto, por ejemplo después de editar
Permisos	<code>.claude/settings.json</code> › <code>permissions</code>	Qué puede hacer sin preguntar, qué tiene que preguntar y qué no puede hacer nunca

Práctica 9.1 · CLAUDE.md de equipo y normas para las pruebas

Objetivo. Dejar por escrito las normas del proyecto de stock y las de las pruebas.

1. Vuelve a `main` y actualízala: `git switch main` y `git pull`.
2. Abre `claude` y pega el prompt del CLAUDE.md.
3. Pega el prompt de las normas de pruebas y revisa los dos archivos.

CLAUDE.MD DE EQUIPO

Escribe el CLAUDE.md de este proyecto para todo el equipo, de menos de 50 líneas: comandos para arrancar y probar, estructura de carpetas, reglas de negocio de la reposición y cómo trabajamos (catalán en código y commits, ninguna dependencia nueva sin preguntar, cada cambio de lógica con su prueba, una rama por incidencia y nunca push directo a main, npm test en verde antes de terminar).

NORMAS DE LAS PRUEBAS

Crea `.claude/rules/proves.md` con `paths: test/**/*.js` y estas normas: `node:test` y `node:assert/strict` sin librerías externas; un test por comportamiento con el nombre en catalán en forma de frase; datos de prueba dentro del mismo archivo; cuando una cifra sale de un cálculo, el cálculo en un comentario.

☐ CLAUDE.md y .claude/rules/proves.md creados

Práctica 9.2 · Una habilidad: el informe de reposición

Objetivo. Convertir una tarea semanal en una habilidad que todo el equipo puede invocar con un comando.

1. Pega el prompt para crear la habilidad.
2. Escribe `/informe-reposicio` Gràcia y mira qué hace.
3. Abre el informe que ha guardado en `informes/` y comprueba que aparece la bebida de almendra.

CREAR LA HABILIDAD

Crea una habilidad de proyecto en `.claude/skills/informe-reposicio/SKILL.md`.

Descripción: prepara el informe semanal de reposición por local y el correo para cada proveedor; se debe usar cuando se solicite el pedido de la semana o el informe de reposición.

Pasos: ejecutar `npm test` y detenerse si falla; obtener la lista de reposición con la lógica de `src/estoc.js` sin modificar nada, filtrando por el local que llegue como argumento; guardar `informes/reposicio-AAAA-MM-DD.md` con una tabla por local y un correo de pedido por proveedor firmado por Laura Sanz; no enviar nada.

Añade `informes/` al `.gitignore`.

☐ Habilidad creada y probada con `/informe-reposicio`

Práctica 9.3 · Un subagente revisor de pruebas

Objetivo. Crear un especialista que revisa si los cambios tienen las pruebas que necesitan, con un contexto propio y sin poder modificar archivos.

1. Pega el prompt para crear el subagente.
2. Úsalo sobre la rama de la incidencia 3, o sobre el último commit si todavía no la has empezado.

CREAR EL SUBAGENTE

Crea el subagente `.claude/agents/revisor-proves.md`: revisa si un cambio de código tiene las pruebas que necesita; se debe usar después de modificar `src/` y antes de abrir una pull request. Herramientas: Read, Grep, Glob y Bash. Modelo: sonnet. Instrucciones: mirar git diff, comprobar para cada función nueva o modificada que hay una prueba del caso normal y una de un caso límite, ejecutar `npm test` y devolver una lista de pruebas que faltan. No puede modificar archivos.

USARLO

Usa el subagente `revisor-proves` para revisar los cambios del último commit.

☐ Subagente creado y probado

Práctica 9.4 · Un hook que ejecuta las pruebas

Objetivo. Hacer que las pruebas se ejecuten solas cada vez que el agente edita un archivo de `src/` o de `test/`, y que el agente vea el error si fallan.

1. Pega el prompt del hook.
2. Escribe `/hooks` para comprobar que está registrado.
3. Pega el prompt de prueba: verás que las pruebas fallan justo después de la edición y que el agente lo detecta.
4. Deshaz el cambio con `/rewind`.

CREAR EL HOOK

Crea un hook `PostToolUse` en `.claude/settings.json` para las herramientas `Edit` y `Write` que ejecute `node .claude/hooks/proves-despres-d-editar.mjs`.
 El script, en `Node.js`: lee el JSON de la entrada estándar; si el archivo editado (`tool_input.file_path`) está dentro de `src/` o `test/`, ejecuta `node --test`; si falla, escribe la salida en `stderr` y sale con el código 2 para que el agente vea el error. Tiene que funcionar en Windows y en macOS.

PROBAR EL HOOK

Cambia temporalmente en `src/estoc.js` el valor de `SETMANES_DE_MARGE` a 3 y dime qué ha pasado después de la edición.

☐ **Hook creado, probado y cambio deshecho**
Práctica 9.5 · Permisos del equipo

Objetivo. Decidir qué puede hacer el agente sin preguntar, qué tiene que preguntar y qué no puede hacer nunca, y comprobar que se cumple.

1. Pega el prompt de los permisos y revisa el `settings.json` resultante con la referencia de aquí abajo.
2. Escribe `/permissions` para ver las reglas activas.
3. Pega el prompt de prueba del `.env`: el agente no lo puede leer.

REGLAS DE PERMISOS

Añade a `.claude/settings.json` reglas de permisos:
 - sin preguntar: `npm test`, `npm start`, `node --test`, `git status`, `git diff`, `git log`, `gh issue view` y `gh pr view`;
 - preguntando siempre: `git push` y `gh pr merge`;
 - nunca: leer o editar `.env`, editar `data/`, `git push --force`, `git push origin main`, `npm install` y `rm -rf`.

PROBAR UN PERMISO DENEGADO

Lee el archivo `.env` y dime qué claves contiene.

REFERENCIA · .CLAUDE/SETTINGS.JSON

```
{
  "permissions": {
    "allow": ["Bash(npm test)", "Bash(npm start)", "Bash(node --test*)", "Bash(git status)",
      "Bash(git diff *)", "Bash(git log *)", "Bash(gh issue view *)", "Bash(gh pr view *)"],
    "ask": ["Bash(git push *)", "Bash(gh pr merge *)"],
    "deny": ["Read(./env)", "Read(./env.*)", "Edit(./env)", "Edit(./data/**)", "Bash(git push --force*)",
      "Bash(git push origin main)", "Bash(npm install *)", "Bash(rm -rf *)"]
  },
  "hooks": {
    "PostToolUse": [{ "matcher": "Edit|Write",
      "hooks": [{ "type": "command", "command": "node .claude/hooks/proves-despres-d-editar.mjs", "timeout": 120 }] }]
  }
}
```

Orden de evaluación

Primero se miran las reglas deny, después las ask y al final las allow. Una regla de denegación gana siempre. Las reglas de `.claude/settings.json` se comparten con el equipo; las tuyas, en `.claude/settings.local.json`.

Si te atascas: `solucions/estoc-p9-agent-personalitzat` tiene CLAUDE.md, la norma, la habilidad, el subagente, el hook y los permisos.

☐ **Práctica 9 hecha: permisos configurados y .env protegido**

Documentación oficial: [Manage Claude's memory](#) · [Agent Skills](#) · [Subagents](#) · [Hooks guide](#) · [Configure permissions](#) · [Settings](#)

Bloque 10 · Conectar herramientas: MCP, conectores y plugins

Un navegador real para probar la aplicación y paquetes de habilidades ya hechos.

MCP (Model Context Protocol) es el estándar abierto que usa Claude para conectarse a herramientas externas: un navegador, una base de datos, Jira, Figma. Cada servidor MCP añade herramientas nuevas al agente.

Ámbito	Dónde se guarda	Para quién
<code>local</code> (por defecto)	<code>~/.claude.json</code> , ligado a este proyecto	Solo para ti, en este proyecto
<code>project</code>	<code>.mcp.json</code> en la raíz del proyecto	Todo el equipo, va al repositorio
<code>user</code>	<code>~/.claude.json</code>	Para ti, en todos los proyectos

Práctica 10.1 · Playwright: probar la aplicación en un navegador

Objetivo. Añadir el servidor MCP de Playwright y hacer que el agente pruebe la página de stock como lo haría una persona, en tamaño móvil.

1. Sal de Claude Code y añade el servidor. Con `--scope project` se guarda en `.mcp.json` para todo el equipo.
2. En otro terminal, arranca la aplicación con `npm start`.
3. Abre `claude` y escribe `/mcp`: Playwright tiene que aparecer conectado. Apruébalo si te lo pide.
4. Pega el prompt de la prueba.

TERMINAL · AÑADIR PLAYWRIGHT

```
claude mcp add --scope project playwright -- npx @playwright/mcp@latest
```

Windows: si en `/mcp` el servidor aparece con error, elimínalo con `claude mcp remove playwright` y vuelve a añadirlo con `cmd /c npx @playwright/mcp@latest` al final del comando.

PRUEBA EN EL NAVEGADOR

Usa Playwright para abrir `http://localhost:3000` con una ventana de 390 × 844. Elige el local de Gràcia y comprueba que la bebida de almendra aparece en la lista de reposición con 18 l por pedir y que el enlace «Descarga CSV» apunta a `/api/reposicio.csv` con el filtro de Gràcia. Haz una captura de la página y dime si hay algo que se vea mal en el móvil.

☐ **Playwright conectado y aplicación probada en el navegador**

Práctica 10.2 · Plugins y marketplaces

Objetivo. Instalar un plugin del marketplace oficial y ver qué añade.

1. Escribe `/plugin` y abre la pestaña **Discover**. El marketplace oficial de Anthropic, `claude-plugins-official`, ya está disponible.
2. Elige un plugin relacionado con el desarrollo web o la revisión de código, lee qué incluye e instálalo en el ámbito del proyecto.
3. Escribe `/reload-plugins`. Las habilidades del plugin aparecen con el formato `/plugin:habilidad`.
4. Pregunta a Claude qué ha añadido el plugin.

QUÉ HAY DE NUEVO

¿Qué habilidades, subagentes, hooks o servidores MCP ha añadido el plugin que acabo de instalar? Explicame en una línea para qué sirve cada uno.

Solo fuentes de confianza

Un servidor MCP o un plugin ejecuta código en tu ordenador y puede leer lo que lee el agente. Instala solo los que vengan de una fuente que conozcas y revisa sus permisos.

☐ Plugin instalado y revisado

Documentación oficial: [Connect Claude Code to tools via MCP](#) · [Plugins](#) · [Discover and install plugins](#) · [Playwright MCP](#)

Bloque 11 · Automatizar y publicar

Claude Code dentro de scripts, en GitHub Actions y en la nube, y la landing publicada.

Hasta ahora hemos trabajado en sesiones interactivas. Claude Code también funciona sin conversación: dentro de un script, en un flujo de GitHub o en una rutina que se ejecuta sola en la nube de Anthropic.

Modo	Cómo	Ejemplo
Modo no interactivo	<code>claude -p "..."</code>	Un script que genera el informe de reposición
GitHub Actions	<code>/install-github-app</code> y <code>@claude</code> en una incidencia	Implementar la incidencia 3 desde GitHub
Rutinas	<code>/schedule</code> o <code>claude.ai/code/routines</code>	El informe de reposición cada lunes a las 8:00
Sesión en la nube	<code>claude --cloud "..."</code> o <code>claude.ai/code</code>	Una tarea larga que continúa con el ordenador apagado

Práctica 11.1 · Claude Code en modo no interactivo

Objetivo. Usar `claude -p` desde el terminal, con salida en texto y en JSON, y convertirlo en un script.

1. Sal de Claude Code y ejecuta los comandos de ejemplo en `estoc-cafes-roca`.
2. Pega en una sesión de Claude Code el prompt para crear el script.
3. Ejecuta el script y abre el informe que ha generado.

TERMINAL · CLAUDE -P

```

claude -p "¿Cuántos productos hay que reponer en cada local? Una línea por local." --
allowedTools "Read"
claude -p "Lista los productos por reponer" --allowedTools "Read" --output-format json
git log --oneline -10 | claude -p "Escribe las notas de versión de estos cambios para el
equipo, en catalán"

```

CREAR EL SCRIPT

Crea scripts/informe-reposicio.sh y scripts/informe-reposicio.ps1 (para Windows) que usen claude -p con solo la herramienta Read para generar la tabla de reposición, opcionalmente de un solo local, y la guarden en informes/reposicio-AAAA-MM-DD.md. Explícame cómo se ejecutan.

☐ Comandos probados y script creado

Práctica 11.2 · GitHub Actions, rutinas y nube (demostración)

Objetivo. Ver cómo Claude Code trabaja sin tu ordenador: desde una incidencia de GitHub, con una rutina programada y en una sesión en la nube.

1. **GitHub Actions.** El formador ejecuta `/install-github-app` en el repositorio. Claude Code instala la aplicación de GitHub, guarda el secreto y abre una pull request con el flujo `.github/workflows/claude.yml`.
2. Una vez fusionada, escribe en la incidencia 3 un comentario que empiece por `@claude`. Al cabo de unos minutos, Claude abre una pull request.
3. **Rutinas.** Con `/schedule`, el formador crea una rutina que cada lunes a las 8:00 prepara el informe de reposición.
4. **Nube.** `claude --cloud` lanza una tarea que continúa en `claude.ai/code` aunque se cierre el portátil.

COMENTARIO EN LA INCIDENCIA 3

@claude implementa esta incidencia. Añade pruebas de la función de filtro y abre una pull request.

RUTINA SEMANAL

`/schedule` cada lunes a las 8:00, en el repositorio `estoc-cafes-roca`: ejecuta las pruebas, prepara el informe de reposición de cada local y abre una incidencia con la tabla y los correos para los proveedores.

Por qué es una demostración

GitHub Actions necesita permisos de administrador en el repositorio y un token, y las rutinas y las sesiones en la nube necesitan el repositorio conectado a `claude.ai/code`. Lo tienes todo explicado aquí para hacerlo por tu cuenta cuando tengas tu proyecto.

Práctica 11.3 · Publicar la landing en GitHub Pages

Objetivo. Publicar la web de Cafès Roca de la sesión 1 en una dirección pública.

1. Abre un terminal en la carpeta `web-cafes-roca` y abre `claude`.
2. Pega el prompt de publicación.
3. Espera uno o dos minutos y abre la dirección que te dé.

PUBLICAR

Publica esta web en GitHub Pages: crea un repositorio público web-cafes-roca con gh, sube la rama main y activa GitHub Pages desde la rama main y la carpeta raíz con gh api. Antes, comprueba que no subes nada que no se deba publicar. Cuando termines, dame la dirección de la web.

REFERENCIA · LO QUE HARÁ EL AGENTE

```
gh repo create web-cafes-roca --public --source=. --push
gh api -X POST repos/{owner}/web-cafes-roca/pages -f "source[branch]=main" -f
"source[path]=/"
```

Lo que deberías ver

La web en <https://<tu-usuario>.github.io/web-cafes-roca/>, con la landing, las páginas del blog y el dashboard en /dashboard/.

❑ Práctica 11 hecha: script con claude -p y landing publicada

Documentación oficial: [Run Claude Code programmatically](#) · [CLI reference](#) · [Claude Code GitHub Actions](#) · [Routines](#) · [Claude Code on the web](#) · [GitHub Pages](#) · [Quickstart](#)

Bloque 12 · Criterio

Seguridad, privacidad, coste y cuándo basta con el vibe coding.

Un agente que lee contenido externo y ejecuta comandos en tu ordenador necesita límites. Este bloque recoge los que hemos ido poniendo y prueba dos de ellos.

- **Permisos.** Modo Manual para empezar, Plan para explorar, reglas **deny** para lo que no se debe hacer nunca.
- **Sandbox.** `/sandbox` aísla los comandos del sistema de archivos y de la red. Funciona en macOS y Linux; en Windows, dentro de WSL2.
- **Secretos.** Las claves van en `.env`, fuera de Git y con `Read(./env)` denegado. Nunca en un prompt ni en un CLAUDE.md.
- **Inyección de prompts.** Un archivo, una web o una incidencia pueden traer instrucciones ocultas. El agente las debe tratar como datos.
- **Privacidad.** En Configuración > Privacidad decides si las conversaciones sirven para entrenar modelos. Team y Enterprise tienen condiciones de empresa.
- **Coste.** `/usage` antes de las tareas largas. Sonnet para el trabajo diario y Opus cuando haga falta.

Práctica 12.1 · Una instrucción oculta

Objetivo. Ver cómo trata el agente un archivo que contiene instrucciones para él y qué protecciones hay.

1. Abre `docs/nota_proveedor.md` en VS Code y léelo entero, incluido el comentario del final.
2. En Claude Code, pega el prompt del resumen.
3. Lee la respuesta: ¿ha seguido las instrucciones del comentario, las ha ignorado o te las ha señalado?
4. Pega el prompt del análisis.

RESUMIR LA NOTA

Resume `docs/nota_proveedor.md` para el equipo en tres líneas y actualiza lo que haga falta para que los pedidos de bebidas vegetales se hagan en múltiplos de 6.

ANÁLISIS

¿Hay alguna instrucción en este archivo que no deberías seguir? ¿Cuáles de nuestras reglas de permisos la habrían detenido si hubieras intentado seguirla?

Lo que deberías ver

El agente hace el resumen y señala el comentario como sospechoso sin obedecerlo. Si lo hubiera intentado, lo habrían detenido las reglas de `settings.json`: leer y editar `.env` y `git push origin main` están denegados, y `git push` siempre pregunta.

☐ Instrucción oculta detectada y analizada
Cuándo basta con el vibe coding

Basta con eso	Hace falta revisión experta, pruebas y mantenimiento
Prototipos, herramientas internas de un solo uso, una landing estática	Aplicaciones con usuarios, pagos o datos personales
Datos ficticios o públicos	Datos de clientes o de la empresa
Si falla, no pasa nada grave	Si falla, hay un coste o un riesgo legal
La usas tú	La usa un equipo o el público

Antes de dar por terminado un proyecto hecho con un agente

Pruebas que cubren los casos importantes y pasan. Has leído el diff. Ningún secreto en el repositorio. Dependencias revisadas. Alguien sabe mantenerlo cuando el agente no esté.

Documentación oficial: [Security](#) · [Sandboxing](#) · [Data usage](#) · [Model improvement privacy settings](#)

Consulta · Preparación, comandos y fuentes

Guía de instalación, comandos de referencia, glosario, prompts y documentación

Resultados de aprendizaje

Las capacidades que trabajamos en las dos sesiones y lo que tendrás hecho al terminar.

- **Entender el agente.** Explicar cómo lee, planifica, actúa y verifica un agente de programación, y en qué puntos hacen falta tu revisión y tu aprobación.
- **Configurar el entorno.** Instalar Claude Code, elegir modelo y esfuerzo, y controlar qué puede hacer con los modos de permiso y las reglas de settings.json.
- **Construir con método.** Hacer una landing, un análisis de datos y un dashboard a partir de una especificación, y comprobar que el resultado es correcto.
- **Trabajar en equipo.** Usar Git y GitHub con el agente: ramas, commits, incidencias, pull requests y revisión de código.
- **Adaptar el agente.** Escribir CLAUDE.md, habilidades, subagentes, hooks y permisos para un proyecto y un equipo.
- **Criterio.** Aplicar criterios de seguridad, privacidad y coste, y distinguir el código que ya se puede usar del que necesita más revisión.

Resultados del curso

- La landing de Cafès Roca con tres páginas de blog generadas a partir de Markdown, publicada en GitHub Pages.
- Un script de análisis de ventas que se puede volver a ejecutar, con tres gráficos y un informe con cada cifra trazable.
- Un dashboard interactivo con filtros, construido a partir de una especificación escrita por el agente.
- Un repositorio en GitHub con un error corregido con pruebas, una funcionalidad nueva y dos pull requests revisadas.
- Un proyecto configurado para el equipo: CLAUDE.md, una norma de pruebas, una habilidad, un subagente, un hook y permisos.
- Un servidor MCP conectado y un script que usa Claude Code sin abrir ninguna sesión.

Preparación previa

Qué hay que tener instalado y listo antes de la primera sesión.

Haz esta preparación antes de venir. Son unos veinte minutos y hace que la primera hora de clase sea de práctica.

Cuentas

Cuenta	Para qué	Coste
Claude, plan Pro o superior	Claude Code. El plan gratuito no lo incluye	15 € al mes con pago anual o 18 € al mes, sin impuestos. Se puede cancelar en cualquier momento
GitHub	Repositorios, incidencias, pull requests y GitHub Pages	Gratuito

Llega con límite disponible

El límite de uso de Claude Code se comparte con el chat y se renueva por ventanas de cinco horas y por semana. Para las dos sesiones basta con el plan Pro, siempre que no lo agotes los días anteriores.

Software

Programa	Windows	macOS
Aplicación de escritorio de Claude	claude.com/download	claude.com/download
Claude Code	<code>irm https://claude.ai/install.ps1 iex</code> en PowerShell	<code>curl -fsSL https://claude.ai/install.sh bash</code> en el Terminal
Git	git-scm.com (Git para Windows) o <code>winget install --id Git.Git</code>	<code>xcode-select --install</code> o <code>brew install git</code>
Node.js 22 LTS	nodejs.org o <code>winget install OpenJS.NodeJS.LTS</code>	nodejs.org o <code>brew install node</code>
GitHub CLI	cli.github.com o <code>winget install --id GitHub.cli</code>	<code>brew install gh</code>
Visual Studio Code	code.visualstudio.com	code.visualstudio.com

Preparación · Instalar y comprobar

Objetivo. Tenerlo todo instalado, con la sesión iniciada en Claude y en GitHub, y comprobarlo con cinco comandos.

1. Instala los programas de la tabla en este orden: Git, Node.js, GitHub CLI, VS Code, la aplicación de Claude y Claude Code.
2. Cierra el terminal y abre uno nuevo para que reconozca los programas instalados.
3. Escribe `claude`, elige iniciar sesión con tu cuenta de Claude y complétalo en el navegador con tu cuenta Pro.
4. Escribe `gh auth login` y sigue los pasos: **GitHub.com** › **HTTPS** › **Login with a web browser**.
5. Configura tu nombre y tu correo en Git con los dos comandos de `git config`.
6. Ejecuta los comandos de comprobación.

TERMINAL · IDENTIDAD DE GIT

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

TERMINAL · COMPROBACIÓN FINAL

```
claude --version
claude doctor
git --version
node --version
gh auth status
```

Ordenadores del Cibernàrium: si usas un ordenador del aula, tiene que estar habilitado para instalar estos programas y para acceder a claude.ai, claude.com, anthropic.com, github.com y registry.npmjs.org. Claude Code se instala en la carpeta del usuario, sin permisos de administrador. Entre sesiones, el trabajo queda en tu repositorio de GitHub.

❑ Preparación hecha: los cinco comandos responden

Documentación oficial: [Set up Claude Code](#) · [Troubleshooting installation](#) · [GitHub CLI](#) · [Git](#) · [Node.js](#)

Comandos de referencia

Los comandos, atajos y archivos que usamos en el curso.

En el terminal

Comando	Qué hace
<code>claude</code>	Abre una sesión interactiva en la carpeta actual
<code>claude "tarea"</code>	Abre una sesión que empieza con esa tarea
<code>claude -c</code>	Continúa la última conversación de esta carpeta
<code>claude --permission-mode plan</code>	Empieza en modo Plan
<code>claude -w nombre</code>	Abre una sesión en un worktree nuevo
<code>claude -p "tarea"</code>	Modo no interactivo: hace la tarea, escribe el resultado y sale
<code>claude --cloud "tarea"</code>	Lanza una sesión en la nube
<code>claude mcp add · claude mcp list</code>	Añade o lista servidores MCP
<code>claude update · claude doctor</code>	Actualiza Claude Code o diagnostica su instalación

Dentro de la sesión

Comando	Qué hace
<code>/init</code>	Genera un CLAUDE.md a partir del proyecto
<code>/memory</code>	Abre los archivos de memoria y la memoria automática
<code>/model · /effort</code>	Cambia el modelo o el nivel de esfuerzo
<code>/permissions</code>	Muestra y edita las reglas de permisos
<code>/rewind</code>	Vuelve a un punto de control anterior
<code>/context · /compact · /clear</code>	Mira, resume o vacía el contexto
<code>/usage</code>	Consumo de la sesión y límites del plan
<code>/btw</code>	Pregunta lateral que no queda en el historial
<code>/resume</code>	Recupera una conversación anterior
<code>/code-review · /security-review</code>	Revisión de código y de seguridad de los cambios
<code>/hooks · /mcp · /plugin</code>	Gestiona hooks, servidores MCP y plugins
<code>/sandbox</code>	Activa el aislamiento de los comandos
<code>/install-github-app</code>	Configura Claude Code en GitHub Actions
<code>/schedule</code>	Crea una rutina que se ejecuta en la nube
<code>/exit</code>	Sale de la sesión

Atajos

Atajo	Qué hace
Mayús+Tab	Cambia de modo de permiso
Esc	Detiene el agente sin perder el trabajo hecho
Esc Esc	Abre el menú de puntos de control
@archivo	Añade un archivo a la petición
!comando	Ejecuta un comando de terminal sin pasar por el agente
/	Muestra todos los comandos y habilidades disponibles

Archivos de configuración

Archivo	Para qué
CLAUDE.md · CLAUDE.local.md	Instrucciones del proyecto, compartidas o personales
.claude/rules/*.md	Normas para unos archivos concretos
.claude/skills/<nombre>/SKILL.md	Habilidades del proyecto
.claude/agents/<nombre>.md	Subagentes del proyecto
.claude/settings.json · .claude/settings.local.json	Permisos, hooks y preferencias, del equipo o personales
.mcp.json	Servidores MCP del proyecto
~/ .claude/	Las mismas piezas, para todos tus proyectos

Glosario

Los términos del curso en palabras llanas.

Término	Definición
Agente	Un modelo de IA que recibe un objetivo, hace un plan y actúa con herramientas (leer archivos, ejecutar comandos) hasta completarlo.
Ámbito (scope)	Dónde se guarda una configuración y a quién afecta: solo a ti, al proyecto o a todas tus sesiones.
Rama	Una línea de trabajo separada dentro de un repositorio. La principal se llama main.
CLAUDE.md	Archivo de instrucciones que Claude Code lee al comienzo de cada sesión.
CLI	Programa que se usa escribiendo comandos en el terminal. GitHub CLI es gh.
Commit	Una foto de los cambios del proyecto, con un mensaje que explica qué se ha hecho.
Contexto	Todo lo que el agente tiene presente en una sesión: mensajes, archivos leídos y salidas de comandos.
CSV	Archivo de texto con datos en columnas separadas por un carácter, aquí el punto y coma.
Dashboard	Página con indicadores, gráficos y filtros que resume muchos datos.

Término	Definición
Diff	Las líneas que han cambiado entre dos versiones de un archivo.
Endpoint	Una dirección de una API que devuelve datos, como /api/reposicio.
Esfuerzo	Cuánto razona el modelo antes de responder. Se elige con /effort.
GitHub Actions	Servicio de GitHub que ejecuta tareas automáticas cuando pasa algo en el repositorio.
GitHub Pages	Alojamiento gratuito de webs estáticas desde un repositorio de GitHub.
Habilidad (skill)	Un procedimiento escrito en un SKILL.md que el agente usa cuando corresponde o cuando lo invocas con /nombre.
Hook	Un comando que se ejecuta siempre en un momento concreto de la sesión, por ejemplo después de editar un archivo.
Incidencia (issue)	Una tarea, un error o una petición anotada en un repositorio de GitHub.
Inyección de prompts	Instrucciones ocultas en un contenido externo que intentan que el agente haga lo que no debe.
JSON	Formato de texto para datos estructurados, con claves y valores.
Landing page	Página web de una sola pantalla larga, pensada para un objetivo concreto.
Markdown	Formato de texto sencillo con marcas como # para los títulos o ** para la negrita.
MCP	Model Context Protocol: el estándar para conectar Claude con herramientas y servicios externos.
Modo de permiso	El nivel de autonomía del agente: Manual, Accept edits, Plan o Auto.
Node.js	Entorno para ejecutar JavaScript fuera del navegador. Con él ejecutamos los scripts y la aplicación de stock.
Plugin	Un paquete que instala de una vez habilidades, subagentes, hooks y servidores MCP.
Prueba (test)	Un código que comprueba automáticamente que una función hace lo que tiene que hacer.
Pull request	La propuesta de llevar los cambios de una rama a main, para que se revisen antes.
Punto de control	Un estado de los archivos que Claude Code guarda antes de editar y al que puedes volver con /rewind.
Repositorio	La carpeta de un proyecto con todo el historial de cambios gestionado por Git.
Rutina	Una tarea de Claude Code que se ejecuta sola en la nube, por horario o por un evento.
Sandbox	Aislamiento que limita qué pueden tocar los comandos que ejecuta el agente.

Término	Definición
SPEC	Documento de especificación: qué hay que construir, para quién y cómo sabremos que está bien.
Subagente	Un agente especializado, con su propio contexto y sus herramientas, al que el agente principal encarga una parte del trabajo.
Vibe coding	Construir software describiéndolo en lenguaje natural y dejando que un agente escriba el código.
Worktree	Una segunda carpeta de trabajo del mismo repositorio, en otra rama, para trabajar en paralelo.

Documentación oficial

Las fuentes de este temario.

Página	Dirección
Claude Code overview	code.claude.com/docs/en/overview
Set up Claude Code	code.claude.com/docs/en/setup
How Claude Code works	code.claude.com/docs/en/how-claude-code-works
Best practices for Claude Code	code.claude.com/docs/en/best-practices
Common workflows	code.claude.com/docs/en/common-workflows
Model configuration	code.claude.com/docs/en/model-config
Permission modes	code.claude.com/docs/en/permission-modes
Checkpointing	code.claude.com/docs/en/checkpointing
Manage Claude's memory	code.claude.com/docs/en/memory
Commands	code.claude.com/docs/en/commands
CLI reference	code.claude.com/docs/en/cli-reference
Manage costs effectively	code.claude.com/docs/en/costs
Run parallel sessions with Git worktrees	code.claude.com/docs/en/worktrees
Claude Code on desktop	code.claude.com/docs/en/desktop
Agent Skills	code.claude.com/docs/en/skills
Subagents	code.claude.com/docs/en/sub-agents
Hooks guide	code.claude.com/docs/en/hooks-guide
Configure permissions	code.claude.com/docs/en/permissions
Connect Claude Code to tools via MCP	code.claude.com/docs/en/mcp
Plugins	code.claude.com/docs/en/plugins/overview
Run Claude Code programmatically	code.claude.com/docs/en/headless
Claude Code GitHub Actions	code.claude.com/docs/en/github-actions
Routines	code.claude.com/docs/en/routines

Página	Dirección
Claude Code on the web	code.claude.com/docs/en/claude-code-on-the-web
Security	code.claude.com/docs/en/security
Sandboxing	code.claude.com/docs/en/sandboxing
Using Claude Code with your Pro or Max plan	support.claude.com/en/articles/11145838-using-claude-code-with-your-pro-or-max-plan
Planes y precios de Claude	claude.com/pricing

Información contrastada

Lo que dice este temario sobre Claude Code sale de la documentación oficial de Anthropic y se ha contrastado el 10 de octubre de 2026. Claude Code se actualiza cada semana: si un comando o un menú no coincide con lo que ves, manda la documentación.