GUÍA PASO A PASO

Entender un repositorio heredado con Claude Code

Los cinco prompts listos para copiar que arman el mapa en cuatro capas en plan mode, la plantilla de ARCHITECTURE.md para guardarlo y un script en Node que comprueba que el mapa no cita archivos inventados. Con la checklist del primer cambio.

PARA	Quien hereda un repositorio y trabaja con Claude Code
PROBADO CON	Node 22.22.3 (check-map.mjs) · Git Bash (git log) · documentación de Claude Code leída el 19 de noviembre de 2026
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/permission-modes
ARTÍCULO	ecomlabs.dev/g/legado
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Arranca sin poder cambiar nada
Paso 01	Capas 1 y 2: carpetas, entradas y datos
Paso 02	Capas 3 y 4: dependencias y zonas de riesgo
Paso 03	El plan es el mapa
Paso 04	Plantilla de ARCHITECTURE.md
Paso 05	Comprueba que las rutas existen
Al terminar	Checklist

ANTES DE EMPEZAR

Arranca sin poder cambiar nada

La primera sesión solo lee. Desde la raíz del repositorio, con el árbol de git limpio y una rama para el mapa:

```
git status
git switch -c docs/mapa-arquitectura
claude --permission-mode plan
```

En plan mode Claude lee archivos, corre comandos para explorar y escribe un plan, pero no edita tu código hasta que apruebas el plan. Si ya estás en una sesión, `Shift+Tab` recorre los modos hasta que la barra de estado diga `⏸ plan mode on`. Tres reglas para toda la sesión:

- Una capa por mensaje. Lee la respuesta y corrige antes de pasar a la siguiente.
- Cada dato con su archivo (`archivo:línea`), o marcado "sin confirmar".
- Variables de entorno por nombre. Nunca pegues ni pidas valores.

PASO 01

Capas 1 y 2: carpetas, entradas y datos

Copia cada prompt en su propio mensaje.

```
Estoy heredando este repositorio y todavía no voy a cambiar nada. Dame la capa 1
del mapa: qué hay en cada carpeta de primer nivel, en una línea; cuáles son código
propio, generado, copiado de terceros o configuración; y los comandos para
instalar, arrancar, probar y construir, sacados de los archivos. Cita el archivo
de cada dato. Si no lo puedes confirmar leyendo, escribe "sin confirmar".
```

```
Capa 2: lista las entradas del sistema (rutas HTTP, webhooks, cron, colas, CLI)
con archivo:línea. Sigue el flujo más importante desde la entrada hasta donde se
guarda o se envía, paso a paso. Dime dónde están el esquema y las migraciones.
```

Qué revisar en la respuesta: que los comandos vengan de un archivo citado (manifiesto, Makefile, CI) y que el flujo tenga una cita en cada paso.

PASO 02

Capas 3 y 4: dependencias y zonas de riesgo

```
Capa 3: del manifiesto y el lockfile, dame framework y runtime con versión, las
librerías que tocan dinero, autenticación o datos, y las que parecen sin uso
(compruébalo con grep antes de decirlo). Lista servicios externos y variables de
entorno que lee el código: solo nombres, nunca valores.
```

```
Capa 4: ejecuta
git log --since="12 months ago" --format= --name-only | sort | uniq -c | sort -rn | head -20
y cruza esos archivos con los que no tienen tests, los más largos, los que tocan
pagos, auth o datos, y los que tienen TODO o FIXME. Dame una tabla: archivo,
señal, evidencia, riesgo (alto, medio, bajo). Termina con las preguntas que solo
alguien del equipo puede responder.
```

`git log` corre sin pedir permiso; `sort` y `uniq` pueden pedirlo según tu modo, y solo leen. En PowerShell no existe `uniq`: corre ese paso en Git Bash o WSL. Ante cada señal: test antes de tocar, y nada de refactorizar en el primer cambio.

PASO 03

El plan es el mapa

```
Escribe como plan el contenido de docs/ARCHITECTURE.md con las cuatro capas, cada
dato con su archivo, y una sección de preguntas abiertas. No propongas cambios
de código.
```

Cuando el plan te convenza, elige «Yes, manually approve edits». Claude sale de plan mode y te pide confirmar cada edición: la única debe ser `docs/ARCHITECTURE.md`. Rechaza cualquier otra.

> **CUIDADO**
> No importes el mapa con `@docs/ARCHITECTURE.md` dentro de `CLAUDE.md`: se cargaría en cada sesión, y la documentación desaconseja describir la arquitectura ahí. Cítalo con `@` en el prompt de la tarea que lo necesite. Para nombrarlo en `CLAUDE.md` sin importarlo, escribe la ruta entre comillas invertidas.

PASO 04

Plantilla de ARCHITECTURE.md

Si prefieres llenarlo tú con las respuestas, o comparar el plan con una estructura fija, usa esta plantilla. Borra las filas de ejemplo.

```
# Mapa de arquitectura · <nombre del repositorio>
Actualizado: AAAA-MM-DD · Commit: <hash corto> · Revisado por: <tu nombre>

## 1. Carpetas
| Carpeta | Qué hay | Tipo (propio, generado, terceros, config) |
|---|---|---|
| `src/` | Código de la aplicación | propio |

Comandos (citados de `package.json`, Makefile o CI):
- Instalar:
- Arrancar:
- Probar:
- Construir:
```

```
## 2. Entradas y datos
| Entrada | Tipo (ruta, webhook, cron, cola, CLI) | Archivo |
|---|---|---|

Flujo principal, paso a paso:
1. (`archivo:línea`)

Esquema y migraciones:
Escribe fuera del repositorio (base de datos, APIs, archivos):

## 3. Dependencias
- Runtime y framework, con versión:
- Críticas (dinero, auth, datos):
- Posiblemente sin uso (grep sin resultados):
- Servicios externos:
- Variables de entorno (solo nombres):

## 4. Zonas de riesgo
| Archivo | Señal | Evidencia | Riesgo |
|---|---|---|---|

## Preguntas abiertas
- [ ] Pregunta · quién puede responderla

## Sin confirmar
- Dato · por qué no se pudo confirmar
```

Escribe cada ruta entre comillas invertidas: así el script del paso siguiente la encuentra, y `CLAUDE.md` no la importa si algún día la copias ahí.

PASO 05

Comprueba que las rutas existen

Un mapa escrito por un modelo puede citar un archivo que no existe. Guarda esto como `check-map.mjs` y córrelo desde la raíz con `node check-map.mjs` (o pásale otra ruta de mapa como argumento):

```
// check-map.mjs: every file path the map cites must exist in the repo.
import { readFileSync, existsSync } from "node:fs";

const map = process.argv[2] ?? "docs/ARCHITECTURE.md";
const lines = readFileSync(map, "utf8").split(/\r?\n/);
const EXT = /\.(json|md|ts|tsx|js|jsx|mjs|cjs|py|rb|go|php|sql|ya?ml|toml|prisma|liquid|css|html|sh)$/i;
const isPath = (s) =>
  !/\s|\*|^\/|^\$|^[a-z]+:/i.test(s) && (s.includes("/") || s.startsWith(".") || EXT.test(s));

let checked = 0;
const missing = [];
lines.forEach((line, i) => {
```

```
  for (const [, span] of line.matchAll(/`([^`]+)`/g)) {
    const path = span.replace(/#.*$/, "").replace(/:\d+(-\d+)?$/, "");
    if (!isPath(path)) continue;
    checked++;
    if (!existsSync(path)) missing.push(`line ${i + 1}: ${path}`);
  }
});

for (const m of missing) console.log(`MISSING ${m}`);
console.log(`${missing.length ? "FAIL" : "OK"}: ${checked} paths checked, ${missing.length} missing`);
process.exit(missing.length ? 1 : 0);
```

Cuenta como ruta lo que no tiene espacios ni comodines y lleva una barra, empieza con punto o termina en una extensión de código. Salta rutas HTTP (`/api/orders`), variables (`$DATABASE_URL`), `node:fs`, comandos y globs. Sale con código 1 si falta alguna. Un `.env` ignorado por git aparece como faltante en un clon nuevo: es esperado. El script prueba que los archivos existen, no que la descripción sea correcta.

AL TERMINAR

Checklist

- [ ] Árbol limpio, rama propia y sesión en plan mode.
- [ ] Cuatro capas, una por mensaje, cada dato con su archivo o "sin confirmar".
- [ ] Variables de entorno solo por nombre.
- [ ] El mapa en `docs/ARCHITECTURE.md`, sin importarlo en `CLAUDE.md`.
- [ ] `node check-map.mjs` pasa.
- [ ] `/init` después del mapa; el `CLAUDE.md` que genera, recortado.
- [ ] Primer cambio en una sesión nueva: riesgo bajo, con test, diff corto, se revierte con un commit.
- [ ] Las preguntas abiertas, enviadas a quien puede responderlas.