EcomLabs

STEP-BY-STEP GUIDE

Understand a legacy codebase with Claude Code

The five copy-ready prompts that build the four-layer map in plan mode, the ARCHITECTURE.md template to save it in, and a Node script that checks the map doesn't cite files that don't exist. With the first-change checklist.

FOR	Anyone who inherits a codebase and works with Claude Code
TESTED WITH	Node 22.22.3 (check-map.mjs) · Git Bash (git log) · Claude Code docs read on November 19, 2026
UPDATED	November 19, 2026
DOCS	code.claude.com/docs/en/permission-modes
ARTICLE	ecomlabs.dev/g/legado-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	Start where nothing can change
Step 01	Layers 1 and 2: folders, entry points and data
Step 02	Layers 3 and 4: dependencies and risk zones
Step 03	The plan is the map
Step 04	ARCHITECTURE.md template
Step 05	Check that the paths exist
When you finish	Checklist

BEFORE YOU START

Start where nothing can change

The first session only reads. From the repository root, with a clean git tree and a branch for the map:

```
git status
git switch -c docs/architecture-map
claude --permission-mode plan
```

In plan mode Claude reads files, runs shell commands to explore and writes a plan, but doesn't edit your source until you approve the plan. If you're already in a session, `Shift+Tab` cycles modes until the status bar reads `⏸ plan mode on`. Three rules for the whole session:

- One layer per message. Read the answer and correct it before the next one.
- Every fact with its file (`file:line`), or marked "unconfirmed".
- Environment variables by name. Never paste or ask for values.

STEP 01

Layers 1 and 2: folders, entry points and data

Paste each prompt as its own message.

```
I'm inheriting this repository and I won't change anything yet. Give me layer 1
of the map: what each top-level folder holds, in one line; which parts are
hand-written, generated, vendored or config; and the commands to install, run,
test and build, taken from the files. Cite the file for every fact. If you
can't confirm something by reading, write "unconfirmed".
```

```
Layer 2: list the system's entry points (HTTP routes, webhooks, cron, queues,
CLI) with file:line. Trace the most important flow from the entry point to
where it is stored or sent, step by step. Tell me where the schema and the
migrations live.
```

What to check in the answer: the commands come from a cited file (manifest, Makefile, CI) and every step of the flow has a citation.

STEP 02

Layers 3 and 4: dependencies and risk zones

```
Layer 3: from the manifest and the lockfile, give me the framework and runtime
with versions, the libraries that touch money, auth or data, and the ones that
look unused (check with grep before you say so). List external services and
the environment variables the code reads: names only, never values.
```

```
Layer 4: run
git log --since="12 months ago" --format= --name-only | sort | uniq -c | sort -rn | head -20
and cross those files with the ones without tests, the longest ones, the ones
that touch payments, auth or data, and the ones with TODO or FIXME. Give me a
table: file, signal, evidence, risk (high, medium, low). End with the
questions only someone on the team can answer.
```

`git log` runs without a prompt; `sort` and `uniq` may ask depending on your mode, and they only read. PowerShell has no `uniq`: run that step in Git Bash or WSL. For each signal: a test before touching, and no refactoring in the first change.

STEP 03

The plan is the map

```
Write as your plan the content of docs/ARCHITECTURE.md with the four layers,
every fact with its file, and an open questions section. Don't propose code
changes.
```

When the plan holds up, choose "Yes, manually approve edits". Claude leaves plan mode and asks you to confirm each edit: the only one should be `docs/ARCHITECTURE.md`. Reject anything else.

> **CAUTION**
> Don't import the map with `@docs/ARCHITECTURE.md` inside `CLAUDE.md`: it would load in every session, and the docs advise against describing architecture there. Reference it with `@` in the prompt of the task that needs it. To name it in `CLAUDE.md` without importing it, put the path in backticks.

STEP 04

ARCHITECTURE.md template

If you'd rather fill it in yourself from the answers, or compare the plan against a fixed structure, use this template. Delete the example rows.

```
# Architecture map · <repository name>
Updated: YYYY-MM-DD · Commit: <short hash> · Reviewed by: <your name>
```

```
## 1. Folders
| Folder | What it holds | Kind (hand-written, generated, vendored, config) |
|---|---|---|
| `src/` | Application code | hand-written |

Commands (cited from `package.json`, Makefile or CI):
- Install:
- Run:
- Test:
- Build:

## 2. Entry points and data
| Entry point | Kind (route, webhook, cron, queue, CLI) | File |
|---|---|---|

Main flow, step by step:
1. (`file:line`)

Schema and migrations:
Writes outside the repo (database, APIs, files):

## 3. Dependencies
- Runtime and framework, with versions:
- Critical (money, auth, data):
- Possibly unused (grep found nothing):
- External services:
- Environment variables (names only):

## 4. Risk zones
| File | Signal | Evidence | Risk |
|---|---|---|---|

## Open questions
- [ ] Question · who can answer it

## Unconfirmed
- Fact · why it couldn't be confirmed
```

Put every path in backticks: that way the script in the next step finds it, and `CLAUDE.md` won't import it if you ever copy it there.

STEP 05

Check that the paths exist

A map written by a model can cite a file that doesn't exist. Save this as `check-map.mjs` and run it from the root with `node check-map.mjs` (or pass another map path as an argument):

```
// 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);
```

It counts as a path anything with no spaces or wildcards that has a slash, starts with a dot or ends in a code file extension. It skips HTTP routes (`/api/orders`), variables (`$DATABASE_URL`), `node:fs`, commands and globs. It exits with code 1 if anything is missing. A git-ignored `.env` shows up as missing in a fresh clone: that's expected. The script proves the files exist, not that the description is right.

WHEN YOU FINISH

Checklist

- [ ] Clean tree, its own branch and a plan mode session.
- [ ] Four layers, one per message, every fact with its file or "unconfirmed".
- [ ] Environment variables by name only.
- [ ] The map in `docs/ARCHITECTURE.md`, not imported into `CLAUDE.md`.
- [ ] `node check-map.mjs` passes.
- [ ] `/init` after the map; the `CLAUDE.md` it generates, trimmed.
- [ ] First change in a fresh session: low risk, tested, short diff, reverted by one commit.
- [ ] Open questions sent to whoever can answer them.