EcomLabs

STEP-BY-STEP GUIDE

A short CLAUDE.md for your agent

Two short CLAUDE.md templates, one for Next.js and one for a Shopify theme, with what goes in and what stays out. Then how to confirm Claude Code loaded them, and when to compact or clear the session so the agent works with just the right context.

FOR	Claude Code on Next.js projects or Shopify themes
TESTED WITH	shopify theme check (Shopify CLI 3.88.1) · Node 22 · Windows 11
UPDATED	October 30, 2026
DOCS	code.claude.com/docs/en/memory
ARTICLE	ecomlabs.dev/g/contexto-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What Claude Code loads at startup
Step 01	Start with /init, then cut
Step 02	Template for Next.js
Step 03	Template for a Shopify theme
Step 04	Move the specific parts into a rule
Step 05	Confirm it loaded
Step 06	During the session
Final	Checklist

BEFORE YOU START

What Claude Code loads at startup

Every Claude Code session starts with an empty context. Before your first message it loads your `CLAUDE.md` files, auto memory, MCP tool names and skill descriptions. Whatever you put in a `CLAUDE.md` takes up context on every task, which is why it should stay short.

SCOPE	FILE	WHAT FOR
User	`~/.claude/CLAUDE.md`	Your preferences across all projects
Project	`./CLAUDE.md` or `./.claude/CLAUDE.md`	The team's commands and rules, in git
Local	`./CLAUDE.local.md`	Your notes for this project, in `.gitignore`
Subdirectory	`folder/CLAUDE.md`	Loads when Claude works in that folder
Rule	`.claude/rules/*.md` with `paths`	Loads with the files that match

The files do not override each other: they are concatenated, from the most general to the one closest to where you launch. The docs recommend under 200 lines per file.

STEP 01

Start with /init, then cut

In a session inside your project, run `/init`. Claude analyzes the code and writes a first `CLAUDE.md` with commands and conventions; if you already have one, it suggests improvements instead of overwriting it. From there the work is removing lines. For each one, ask whether Claude could find it by reading the code, and whether it would actually go and look. If both answers are yes, delete it.

STAYS	GOES
Exact lint, typecheck and test commands	Folder tree
Misleading code: an old pattern not to copy	Dependency list
Decisions that are not in the repo	"Write clean code" and other generic rules
Anything you have corrected twice	Long procedures (they belong in a skill)

STEP 02

Template for Next.js

Save it as `CLAUDE.md` at the project root. The Conventions and Careful lines are examples: replace them with your codebase's.

- `@AGENTS.md` imports the file that `create-next-app` includes in its default setup. If a `CLAUDE.md` exists, Claude Code reads that one and not `AGENTS.md`, unless you import it. If you have no `AGENTS.md`, delete the line.

- **The commands** come from the scripts the Next.js docs put in `package.json`. Before saving, open your `package.json` and confirm they exist under those names.
- **An import does not save context:** the imported file loads in full alongside the `CLAUDE.md`.

```
@AGENTS.md

# Commands
- Dev server: npm run dev
- Lint: npm run lint
- Types: npx tsc --noEmit
- Production build: npm run build, only when I ask for it.

# Conventions
- Prices arrive in cents; format them with lib/price.ts.
- components/legacy/ is still live, but do not copy its patterns.

# Careful
- app/checkout/ and app/api/payments/: plan first, never a direct change.
- Do not read or print .env.local.
```

> **CAUTION**
> "Do not read or print .env.local" is an instruction, not a lock. Claude Code treats `CLAUDE.md` as context, not enforced configuration. If it has to be impossible, block it with a hook or with permission rules.

STEP 03

Template for a Shopify theme

Save it as `CLAUDE.md` at the theme root, next to `layout/` and `sections/`. Replace `your-store.myshopify.com` with your store.

```
# Commands
- Preview: shopify theme dev --store your-store.myshopify.com
- Check the theme: shopify theme check

# Conventions
- Visible text lives in locales/ and is read with the t filter, never
  hard-coded in the Liquid.
- Do not rename or delete the id of an existing setting: the store has
  values saved under that id.

# Careful
- Never run shopify theme push or shopify theme publish: they upload to
  or publish on the store. If it is needed, ask me.
- layout/theme.liquid loads the analytics: tell me before touching it.
```

- `shopify theme check` checks the theme on your machine. We tried it on a minimal theme: with no errors it exits with code 0; with a translation key missing from `locales/` it reports `TranslationKeyExists` and exits

with code 1. That is exactly the mistake the first convention prevents.

- `shopify theme dev` uploads the theme as a development theme to the connected store and gives you a local preview at `http://127.0.0.1:9292`. It does not publish anything.
- **Setting ids** are the key under which the store saves what was already configured in the theme editor, which is why they must not be renamed.

> **CAUTION**
>
> `shopify theme push` uploads your files to the store's theme and `shopify theme publish` changes the live theme. That is why the template forbids them: list in your `CLAUDE.md` the commands Claude may run, and keep these for yourself.

STEP 04

Move the specific parts into a rule

When `CLAUDE.md` grows, move what only applies to one part of the code into `.claude/rules/`. A rule with `paths` loads when Claude reads or edits a matching file. This one, in `.claude/rules/liquid.md`, only applies to sections and snippets:

```
---
paths:
  - "sections/**/*.liquid"
  - "snippets/**/*.liquid"
---
- Visible text lives in locales/ and is read with the t filter.
- After editing a Liquid file, run shopify theme check.
```

We checked that both patterns match `sections/hero.liquid` and `snippets/envio.liquid` and not `layout/theme.liquid`. One warning from the docs: these rules get summarized with the rest of the conversation on compaction, and load again when Claude next touches a matching file. If a rule must always be there, drop its `paths`.

STEP 05

Confirm it loaded

1. Open Claude Code at the project root.
2. Run `/context`. Your `CLAUDE.md` should appear under **Memory files**.
3. If it does not, run `/memory`: it lists your user and project memory files and lets you open them.
4. A subdirectory `CLAUDE.md` does not appear in that list at startup, because it loads when Claude works in that folder.

STEP 06

During the session

Name the task's files with `@` and the path, say what not to touch and what "done" looks like. When the session runs long, decide yourself what stays:

SITUATION	COMMAND	WHAT HAPPENS
Same task, long session	`/compact` with a focus	Summarizes the conversation; the root `CLAUDE.md` is re-read from disk
New, unrelated task	`/clear`	New conversation with empty context; get the old one back with `/resume`
Same fix failed twice	`/clear`	Start without the failed attempts, with a prompt that folds in what you learned
Only part of it is noise	`/rewind`	Summarizes from or up to the message you pick

A focused `/compact` looks like this:

```
/compact keep the changes in components/cart/ and the test commands; drop the exploration of app/checkout/
```

An instruction you gave only in chat can be lost when the session compacts. If it matters in every session, move it to `CLAUDE.md`.

FINAL

Checklist

- [ ] My `CLAUDE.md` is under 200 lines.
- [ ] Every line is something Claude could not work out from the code.
- [ ] The commands exist in my `package.json` or my CLI, under those names.
- [ ] If the project has an `AGENTS.md`, my `CLAUDE.md` imports it with `@AGENTS.md`.
- [ ] Anything that only applies to one folder lives in a rule with `paths`.
- [ ] Anything that cannot fail lives in a hook or permission rules, not a line.
- [ ] `/context` lists my `CLAUDE.md` under **Memory files**.
- [ ] In every prompt I name the files with `@` and say what not to touch.
- [ ] I run `/clear` between unrelated tasks and a focused `/compact` on the same task.