

Plan mode before you write code

How to use Claude Code plan mode so Claude shows you what it will change before it touches a line: enter it, ask for a complete plan, correct it and approve it. With a demo store to practise on safely and a CLAUDE.md that sets the plan's format.

FOR	Claude Code on macOS, Linux or Windows
TESTED WITH	Claude Code 2.1.238 · Sonnet · Node 22 · Windows 11
UPDATED	26 October 2026
DOCUMENTATION	code.claude.com/docs/en/permission-modes
ARTICLE	ecomlabs.dev/blog/claude-code-plan-mode
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will practise
---------------------	------------------------

Step 01	Set up the demo store
---------	-----------------------

Step 02	Tell Claude how you want plans
---------	--------------------------------

Step 03	Enter plan mode
---------	-----------------

Step 04	Ask for the change
---------	--------------------

Step 05	Review the plan
---------	-----------------

Step 06	Correct and approve
---------	---------------------

Step 07	Check the result
---------	------------------

Going further	Plan mode day to day
------------------	----------------------

If something fails	Troubleshooting
--------------------------	-----------------

When you finish	Checklist
--------------------	-----------

What you will practise

Plan mode is a Claude Code permission mode. While it is on, Claude reads your code, runs commands to explore and writes a plan, but does not edit your files. You read the plan, correct it if needed and, when you approve it, Claude starts writing code.

This guide practises it on a demo store: a 15-line checkout with four tests. The task is to add free shipping from €50 without breaking anything. It looks trivial, which is why it makes a good exercise: it hides a business decision that a good plan brings to light.

What is in the ZIP

FILE	WHAT IT DOES
CLAUDE.md	Tells Claude which sections every plan must have.
.claude/settings.json	Optional: the project always starts in plan mode.
src/checkout.js	The store's checkout. Amounts in cents.
test/checkout.test.js	Four tests using Node's runner.
package.json	The <code>npm test</code> script. No dependencies.

Requirements

- Claude Code installed.
- Node 18 or newer, for `node --test`. We tested with Node 22.
- Git, to see what changed at the end.

Set up the demo store

Unzip into a new folder and, from inside it:

```
git init
git add -A
git commit -m "demo store"
npm test
```

All four tests should pass. This is the checkout you are going to change:

`src/checkout.js`

```
// Checkout summary for the demo store. Amounts are in cents so floating-point
// errors do not creep in.
```

```

export const SHIPPING = 490;

export function subtotal(lines) {
  return lines.reduce((sum, l) => sum + l.price * l.quantity, 0);
}

export function summary(lines, coupon = null) {
  const sub = subtotal(lines);
  const discount = coupon ? Math.min(coupon.amount, sub) : 0;
  const shipping = sub === 0 ? 0 : SHIPPING;
  return { subtotal: sub, discount, shipping, total: sub - discount + shipping };
}

```

And these are its tests. At the end of the exercise all four must still pass:

test/checkout.test.js

```

import { test } from "node:test";
import assert from "node:assert/strict";
import { SHIPPING, summary } from "../src/checkout.js";

const tshirt = { sku: "TSH-01", price: 1999, quantity: 2 };

test("no coupon: subtotal plus shipping", () => {
  assert.deepEqual(summary([tshirt]), { subtotal: 3998, discount: 0, shipping: SHIPPING, total: 4488 });
});

test("the coupon comes off the subtotal", () => {
  assert.equal(summary([tshirt], { amount: 500 }).total, 3988);
});

test("a coupon larger than the subtotal leaves only shipping", () => {
  assert.equal(summary([tshirt], { amount: 10000 }).total, SHIPPING);
});

test("empty cart: no shipping", () => {
  assert.equal(summary([]).total, 0);
});

```

STEP 02

Tell Claude how you want plans

With no instructions, a plan can be a list of steps with no risks or questions. The store's `CLAUDE.md` sets six sections. Claude reads it at the start of every session, so you do not need to ask in every message:

CLAUDE.md

```

# Demo store

Minimal project for trying plan mode. Amounts are in cents. Tests: 'npm test'.

```

```
## How I want plans
```

When you are in plan mode, the plan must have these sections, in this order:

1. **Files**: every file you will create or change, and why.
2. **Steps**: in order, one per line, each one checkable.
3. **Risks**: what could break (checkout, prices, taxes, analytics) and how you avoid it.
4. **Questions**: business decisions you cannot make alone. Ask them; do not assume.
5. **Tests**: which tests you add or change and the command that runs them.
6. **How to undo it**: what to revert if something goes wrong.

Do not change `src/checkout.js` without a test that covers the new case.

For your own projects, copy the "How I want plans" section into your `CLAUDE.md`. The last line is specific to the store; replace it with the rule that matters most in your code, for example "do not change anything in `src/payments/` without a test".

Why the questions section

It is the one that adds the most value in a store. Without it, Claude tends to settle whatever is unclear in the request on its own and write it straight into the code. With it, it asks. In our test, that is exactly what happened.

STEP 03

Enter plan mode

The ZIP includes a `.claude/settings.json` that makes the store always start in plan mode:

`.claude/settings.json`

```
{
  "permissions": {
    "defaultMode": "plan"
  }
}
```

Open Claude Code in the folder with `claude`. The status bar should show `plan mode on`. In your projects, without that file, you have these options:

WAY	WHAT IT DOES
<code>Shift+Tab</code>	Cycles the mode: <code>default</code> , <code>acceptEdits</code> , <code>plan</code> . From <code>auto</code> , the first press goes to <code>default</code> .
<code>/plan</code> at the start of the message	Plans just that task.
<code>claude --permission-mode plan</code>	Starts the session in plan mode. Also with <code>-p</code> .
<code>"defaultMode": "plan"</code>	The project always starts in plan mode.

CAUTION

Conversations started by the VS Code extension do not read the project's starting mode. There you set `claudeCode.initialPermissionMode` in your VS Code user settings.

STEP 04

Ask for the change

Write the task the way you would to someone on the team, without solving it yourself:

Add free shipping for orders from 50 euros without breaking checkout

Claude will read the checkout, the tests and `package.json`. It may run a command to explore or launch a subagent to do it; depending on your setup, a classifier approves those commands or you are asked. It will not edit anything.

The plan is saved as Markdown in `~/.claude/plans/`, with a name generated from the request. In our test, planning with Sonnet took 62 seconds and cost USD 0.72.

STEP 05

Review the plan

Read the plan with this table beside it. Each row is a question the plan must let you answer:

SECTION	WHAT TO CHECK	IN THE DEMO STORE
Files	Are they the ones you expected? Any missing or extra?	<code>src/checkout.js</code> and <code>test/checkout.test.js</code> , nothing else
Steps	Can each one be checked?	Add the threshold constant, change the shipping calculation, add tests
Risks	Does it name what could break?	The total with a coupon, the existing tests
Questions	Does it ask what the request leaves unclear?	Is the €50 before or after the coupon? Does exactly €50.00 count?
Tests	Do they cover the edge of the threshold?	€49.99, €50.00 and above €50
Undo	Do you know what to revert?	The two files, with <code>git checkout</code>

In our test, the plan proposed measuring the €50 before the coupon. Under that rule, a €60 cart with a €15 coupon pays €45 and no shipping. Neither right nor wrong: it is a business decision, and it is yours.

STEP 06

Correct and approve

When the plan is ready, Claude Code asks how to proceed. Per the current docs, the options are:

- **Yes, and use auto mode** (or **Yes, auto-accept edits** if your session has no `auto` mode): approve and edit without asking about each change.
- **Yes, manually approve edits**: approve and review each edit.
- **No, keep planning**: stay in plan mode and write what to change.

For the coupon question, pick "No, keep planning" and answer, for example:

```
The threshold is measured on the subtotal after the coupon. Exactly 50.00 euros does get free shipping.
```

For wording changes in the plan, `Ctrl+G` opens it in your text editor. When the plan is right, approve it with manual review the first time: that way you see every edit before it is applied.

Approving takes the session out of plan mode. To plan the next task, go back with `Shift+Tab` or start the message with `/plan`.

STEP 07

Check the result

```
npm test
git diff --stat
```

The four original tests and the new ones should pass, and the diff should only touch the two files in the plan. In our test (on the Spanish store), with the step 6 correction, Claude changed the shipping calculation to this:

```
const neto = sub - descuento;
const envio = sub === 0 ? 0 : neto >= UMBRAL_ENVIO_GRATIS ? 0 : ENVIO;
```

It added four tests (just below the threshold, exactly at it, above it and one that pins down "the threshold is evaluated on the subtotal after the coupon") and `npm test` passed 8 of 8. The implementation cost USD 0.71.

If the diff touches a file that was not in the plan, ask why before committing.

GOING FURTHER

Plan mode day to day

When it pays off

- Changes to checkout, payments, prices, taxes or shipping.
- Tasks that touch several files, or a repository you do not know.
- Requests with business rules that can be read two ways.

For a typo, or a change you have made ten times, it adds waiting and gives nothing back.

Keep the plan with the change

The file in `~/ .claude/plans/` explains what was decided and why. Paste it into the PR description or the ticket: whoever reviews it will know the coupon rule was a decision, not an oversight.

Plans that are too long

If the plan does not fit in one reading, neither does the task. Ask it to split the work into parts that can be approved and tested separately, and plan the first one.

— IF SOMETHING FAILS

Troubleshooting

SYMPTOM	LIKELY CAUSE	FIX
No <code> plan mode on</code>	You are in another mode	Press <code>Shift+Tab</code> until it shows
VS Code does not start in plan mode	The extension does not read the project mode	Use <code>claudeCode.initialPermissionMode</code>
The plan has no questions or risks	The section is missing from <code>CLAUDE.md</code>	Copy "How I want plans"
Claude starts editing on the next request	Approving the plan took you out of plan mode	Go back with <code>Shift+Tab</code> or <code>/plan</code>
It asks permission for many commands	Your session does not have the <code>auto</code> mode classifier	Approve the exploration ones or add <code>allow</code> rules
Original tests fail after implementing	The change broke checkout	Do not commit; go back to plan mode with the error

— WHEN YOU FINISH

Checklist

- ☐ You entered plan mode and saw `|| plan mode on`.
- ☐ The plan had the six sections from `CLAUDE.md`.
- ☐ You answered the coupon question before approving.
- ☐ The diff only touches the files in the plan.
- ☐ `npm test` passes with the original and the new tests.
- ☐ You copied "How I want plans" into your project's `CLAUDE.md`.