

Plan mode antes de escribir código

Cómo usar plan mode en Claude Code para que Claude te enseñe qué va a cambiar antes de tocar una línea: entrar, pedir un plan completo, corregirlo y aprobarlo. Con una tienda de prueba para practicar sin riesgo y un CLAUDE.md que fija el formato del plan.

PARA	Claude Code en macOS, Linux o Windows
PROBADO CON	Claude Code 2.1.238 · Sonnet · Node 22 · Windows 11
ACTUALIZADA	26 de octubre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/permission-modes
ARTÍCULO	ecomlabs.dev/g/plan
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a practicar
---------------------	---------------------

Paso 01	Prepara la tienda de prueba
---------	-----------------------------

Paso 02	Dile a Claude cómo quieres los planes
---------	---------------------------------------

Paso 03	Entra en plan mode
---------	--------------------

Paso 04	Pide el cambio
---------	----------------

Paso 05	Revisa el plan
---------	----------------

Paso 06	Corrige y aprueba
---------	-------------------

Paso 07	Comprueba el resultado
---------	------------------------

Para ampliarlo	Plan mode en tu día a día
-------------------	---------------------------

Si algo falla	Solución de problemas
------------------	-----------------------

Al terminar	Checklist
----------------	-----------

Qué vas a practicar

Plan mode es un modo de permisos de Claude Code. Mientras está activo, Claude lee tu código, corre comandos para explorar y escribe un plan, pero no edita tus archivos. Tú lees el plan, lo corriges si hace falta y, cuando lo apruebas, Claude empieza a escribir código.

Esta guía lo practica con una tienda de prueba: un checkout de 15 líneas con cuatro tests. La tarea es añadir envío gratis a partir de 50 € sin romper nada. Parece trivial, y por eso es un buen ejercicio: tiene una decisión de negocio escondida que un buen plan saca a la luz.

Lo que trae el ZIP

ARCHIVO	PARA QUÉ SIRVE
<code>CLAUDE.md</code>	Le dice a Claude qué secciones debe tener cada plan.
<code>.claude/settings.json</code>	Opcional: el proyecto arranca siempre en plan mode.
<code>src/checkout.js</code>	El checkout de la tienda. Importes en céntimos.
<code>test/checkout.test.js</code>	Cuatro tests con el runner de Node.
<code>package.json</code>	El script <code>npm test</code> . Sin dependencias.

Requisitos

- Claude Code instalado.
- Node 18 o más reciente, para `node --test`. Lo probamos con Node 22.
- Git, para ver al final qué cambió.

Prepara la tienda de prueba

Descomprime el ZIP en una carpeta nueva y, desde ella:

```
git init
git add -A
git commit -m "tienda de prueba"
npm test
```

Deben pasar los cuatro tests. Este es el checkout que vas a cambiar:

`src/checkout.js`

```
// Resumen del checkout de la tienda de prueba. Los importes van en céntimos para
// no arrastrar errores de coma flotante.
```

```
export const ENVIO = 490;

export function subtotal(lineas) {
  return lineas.reduce((suma, l) => suma + l.precio * l.cantidad, 0);
}

export function resumen(lineas, cupon = null) {
  const sub = subtotal(lineas);
  const descuento = cupon ? Math.min(cupon.importe, sub) : 0;
  const envio = sub === 0 ? 0 : ENVIO;
  return { subtotal: sub, descuento, envio, total: sub - descuento + envio };
}
```

Y estos son sus tests. Al final del ejercicio tienen que seguir pasando los cuatro:

test/checkout.test.js

```
import { test } from "node:test";
import assert from "node:assert/strict";
import { ENVIO, resumen } from "../src/checkout.js";

const camiseta = { sku: "CAM-01", precio: 1999, cantidad: 2 };

test("sin cupón: subtotal más envío", () => {
  assert.deepEqual(resumen([camiseta]), { subtotal: 3998, descuento: 0, envio: ENVIO, total: 4488 });
});

test("el cupón se resta del subtotal", () => {
  assert.equal(resumen([camiseta], { importe: 500 }).total, 3988);
});

test("un cupón mayor que el subtotal deja solo el envío", () => {
  assert.equal(resumen([camiseta], { importe: 10000 }).total, ENVIO);
});

test("carrito vacío: sin envío", () => {
  assert.equal(resumen([]).total, 0);
});
```

PASO 02

Dile a Claude cómo quieres los planes

Sin instrucciones, un plan puede ser una lista de pasos sin riesgos ni preguntas. El `CLAUDE.md` de la tienda fija seis secciones. Claude lo lee al empezar cada sesión, así que no tienes que pedirlo en cada mensaje:

CLAUDE.md

```
# Tienda de prueba

Proyecto mínimo para probar plan mode. Los importes van en céntimos. Tests: `npm test`.
```

```
## Cómo quiero los planes
```

Cuando estés en plan mode, el plan debe tener estas secciones, en este orden:

1. **Archivos**: cada archivo que vas a crear o cambiar, y por qué.
2. **Pasos**: en orden, uno por línea, cada uno comprobable.
3. **Riesgos**: qué puede romperse (checkout, precios, impuestos, analítica) y cómo lo evitas.
4. **Dudas**: decisiones de negocio que no puedes tomar solo. Pregúntalas; no las supongas.
5. **Tests**: qué tests añades o cambias y con qué comando se corren.
6. **Cómo deshacerlo**: qué revertir si algo sale mal.

No cambies `src/checkout.js` sin un test que cubra el caso nuevo.

Para tus proyectos, copia la sección «Cómo quiero los planes» a tu propio `CLAUDE.md`. La última línea es específica de la tienda; cámbiala por la regla que más te importe en tu código, por ejemplo «no cambies nada en `src/pagos/` sin un test».

Por qué la sección de dudas

Es la que más valor da en una tienda. Sin ella, Claude tiende a decidir por su cuenta lo que no está claro en el pedido y a escribirlo directamente en el código. Con ella, lo pregunta. En nuestra prueba fue justo lo que pasó.

PASO 03

Entra en plan mode

El ZIP trae un `.claude/settings.json` que hace que la tienda arranque siempre en plan mode:

```
.claude/settings.json
```

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

Abre Claude Code en la carpeta con `claude`. La barra de estado debe mostrar `plan mode on`. En tus proyectos, sin ese archivo, tienes estas formas:

FORMA	QUÉ HACE
<code>Shift+Tab</code>	Cambia de modo: <code>default</code> , <code>acceptEdits</code> , <code>plan</code> . Desde <code>auto</code> , la primera pulsación va a <code>default</code> .
<code>/plan</code> al principio del mensaje	Planifica solo esa tarea.
<code>claude --permission-mode plan</code>	Arranca la sesión en plan mode. También con <code>-p</code> .
<code>"defaultMode": "plan"</code>	El proyecto arranca siempre en plan mode.

CUIDADO

Las conversaciones que abre la extensión de VS Code no leen el modo inicial del proyecto. Ahí se configura con `claudeCode.initialPermissionMode` en los ajustes de usuario de VS Code.

PASO 04

Pide el cambio

Escribe la tarea como se la dirías a alguien del equipo, sin resolverla tú:

Añade envío gratis para pedidos desde 50 € sin romper el checkout

Claude va a leer el checkout, los tests y `package.json`. Puede que corra algún comando para explorar o lance un subagente que lo haga; según tu configuración, esos comandos los aprueba un clasificador o te los pregunta. No va a editar nada.

El plan queda guardado como Markdown en `~/.claude/plans/`, con un nombre generado a partir del pedido. En nuestra prueba, la planificación con Sonnet tardó 62 segundos y costó 0,72 USD.

PASO 05

Revisa el plan

Lee el plan con esta tabla al lado. Cada fila es una pregunta que el plan tiene que dejarte contestar:

SECCIÓN	QUÉ COMPROBAR	EN LA TIENDA DE PRUEBA
Archivos	¿Son los que esperabas? ¿Falta o sobra alguno?	<code>src/checkout.js</code> y <code>test/checkout.test.js</code> , nada más
Pasos	¿Cada uno se puede comprobar?	Añadir la constante del umbral, cambiar el cálculo del envío, añadir tests
Riesgos	¿Nombra lo que puede romperse?	El total con cupón, los tests existentes
Dudas	¿Pregunta lo que no está claro en el pedido?	¿Los 50 € van antes o después del cupón? ¿50,00 € exactos cuentan?
Tests	¿Cubren el borde del umbral?	49,99 €, 50,00 € y más de 50 €
Deshacer	¿Sabes qué revertir?	Los dos archivos, con <code>git checkout</code>

En nuestra prueba, el plan proponía medir los 50 € antes del cupón. Con esa regla, un carrito de 60 € con un cupón de 15 € paga 45 € y no paga envío. Ni bien ni mal: es una decisión de negocio que te toca a ti.

PASO 06

Corrige y aprueba

Cuando el plan está listo, Claude Code te pregunta cómo seguir. Según la documentación actual, las opciones son:

- **Yes, and use auto mode** (o **Yes, auto-accept edits** si tu sesión no tiene modo `auto`): aprueba y edita sin preguntarte cada cambio.
- **Yes, manually approve edits**: aprueba y revisas cada edición.
- **No, keep planning**: sigues en plan mode y escribes qué cambiar.

Para la duda del cupón, elige «No, keep planning» y responde, por ejemplo:

```
El umbral se mide sobre el subtotal después del cupón. 50,00 € exactos sí tienen envío gratis.
```

Para cambios de redacción en el plan, `Ctrl+G` lo abre en tu editor de texto. Cuando el plan esté bien, apruébalo con revisión manual la primera vez: así ves cada edición antes de que se aplique.

Al aprobar, la sesión sale de plan mode. Para planificar la siguiente tarea, vuelve con `Shift+Tab` o empieza el mensaje con `/plan`.

PASO 07

Comprueba el resultado

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

Deben pasar los cuatro tests originales y los nuevos, y el diff solo debe tocar los dos archivos del plan. En nuestra prueba, con la corrección del paso 6, Claude cambió el cálculo del envío a esto:

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

Añadió cuatro tests (justo por debajo del umbral, justo en él, por encima y uno que fija «el umbral se evalúa sobre el subtotal después del cupón») y `npm test` pasó 8 de 8. La implementación costó 0,71 USD.

Si el diff toca un archivo que no estaba en el plan, pregunta por qué antes de hacer commit.

PARA AMPLIARLO

Plan mode en tu día a día

Cuándo vale la pena

- Cambios en checkout, pagos, precios, impuestos o envíos.
- Tareas que tocan varios archivos o un repositorio que no conoces.
- Pedidos con reglas de negocio que admiten dos lecturas.

Para una errata o un cambio que ya hiciste diez veces, añade espera sin aportar nada.

Guarda el plan con el cambio

El archivo de `~/ .claude/plans/` explica qué se decidió y por qué. Pégallo en la descripción del PR o en el ticket: quien revise sabrá que la regla del cupón fue una decisión, no un descuido.

Planes demasiado largos

Si el plan no cabe en una lectura, la tarea tampoco. Pide que la divida en partes que se puedan aprobar y probar por separado, y planifica la primera.

— SI ALGO FALLA

Solución de problemas

SÍNTOMA	CAUSA PROBABLE	ARREGLO
No aparece <code> plan mode on</code>	Estás en otro modo	Pulsa <code>Shift+Tab</code> hasta verlo
VS Code no arranca en plan mode	La extensión no lee el modo del proyecto	Usa <code>claudeCode.initialPermissionMode</code>
El plan no trae dudas ni riesgos	Falta la sección en <code>CLAUDE.md</code>	Copia «Cómo quiero los planes»
Claude empieza a editar en el siguiente pedido	Aprobar el plan te sacó de plan mode	Vuelve con <code>Shift+Tab</code> o <code>/plan</code>
Te pide permiso para muchos comandos	Tu sesión no tiene el clasificador del modo <code>auto</code>	Aprueba los de exploración o añade reglas <code>allow</code>
Fallan tests originales tras implementar	El cambio rompió el checkout	No hagas commit; vuelve a plan mode con el error

— AL TERMINAR

Checklist

- ☐ Entraste en plan mode y viste `|| plan mode on`.
- ☐ El plan trajo las seis secciones del `CLAUDE.md`.
- ☐ Respondiste la duda del cupón antes de aprobar.
- ☐ El diff solo toca los archivos del plan.
- ☐ `npm test` pasa con los tests originales y los nuevos.
- ☐ Copiaste «Cómo quiero los planes» al `CLAUDE.md` de tu proyecto.