Spec Driven Development with Fission AI OpenSpec

AI Assist Generated Content

What is OpenSpec?

OpenSpec is an open-source, lightweight framework designed to manage and enhance **spec-driven development (SDD)** specifically for AI coding assistants (such as GitHub Copilot, Claude, or Cursor).

Its primary purpose is to solve the problem of "vibe coding"—where developers give AI assistants vague, iterative instructions in a chat history, leading to unpredictable, un-reviewable, and non-deterministic results.

OpenSpec institutionalizes a simple workflow that forces alignment between the human developer and the AI assistant *before* any code is written. It does this by creating explicit, version-controlled specification files that act as a "source of truth" for any change. This ensures all AI-generated code is based on an agreed-upon plan, making the output predictable, auditable, and easier to review.

Key features and benefits include:

- **Human-AI Alignment:** Guarantees both the developer and the AI agree on *what* to build upfront.
- **Auditability & Review:** Creates a clear, auditable trail of proposals, tasks, and specification changes, separate from the code itself.
- **Deterministic Output:** Moves requirements out of ephemeral chat history and into structured files, leading to more reliable AI code generation.
- **Tool-Agnostic:** Works with most major AI coding assistants through native slash commands or simple context-sharing.
- **"Brownfield" Friendly:** Excels at modifying existing codebases (1 $\to$ n updates), not just new projects (0 $\to$ 1).

Detailed Steps and Workflow for Using OpenSpec

The OpenSpec methodology is built around a simple, four-stage workflow: **Propose → Review & Refine → Apply → Archive**.

Phase 1: Setup (One-Time)

Before you can use the workflow, you must initialize OpenSpec in your project.

1. **Install the CLI:** OpenSpec is a Node.js tool. Install it globally in your terminal:

```
Bash
```

```
npm install -g @fission-ai/openspec@latest
```

2. **Initialize Your Project:** Navigate to your project's root directory and run:

```
Bash
```

```
openspec init
```

3. **Configure:** The init command will:
   - Ask you to select your AI coding assistant (e.g., Cursor, GitHub Copilot, Claude). This automatically configures slash commands (`/openspec:proposal` , etc.) for supported tools.
   - Create a new directory: `openspec/` .
   - Generate a `openspec/project.md` file.
4. **Populate Project Context:** Open `openspec/project.md` and (ideally with your AI's help) fill it with details about your project's tech stack, coding conventions, architectural patterns, and other guidelines. This file acts as a permanent set of instructions for the AI on *how* to build things for *this specific project*.

Phase 2: The Core Workflow

This cycle is repeated for every new feature, bugfix, or change.

Step 1: Draft the Proposal

You start by describing *what* you want to change.

- **Action:** In your AI assistant's chat, use the slash command:

```
/openspec:proposal Add user authentication with 2FA
```

- **What Happens:** The AI, using the context from your `project.md` , creates a new change folder (e.g., `openspec/changes/add-user-auth/`). This folder contains:
  - `proposal.md` : A human-readable description of the feature (the "why" and "what").
  - `tasks.md` : A checklist of implementation steps the AI plans to take.

- `specs/`: A subdirectory containing "spec deltas"—files that explicitly state the *changes* to the system's specifications.

Step 2: Verify & Refine Specs

This is the most critical step for human-AI alignment.

- **Action:** As a developer, you **review** the files the AI generated in the change folder.
- **Checklist:**
  - Does `proposal.md` accurately capture your intent?
  - Do the `tasks.md` steps make sense?
  - Are the `specs/` deltas correct?
- **Iteration:** If anything is wrong or missing, you iterate with your AI in the chat (e.g., "You missed the acceptance criteria for the 2FA screen. Please update the spec delta."). You repeat this until the specification is precise and you (the human) approve it.

Step 3: Implement the Change

Once the spec is locked, you give the AI the "go-ahead" to write code.

- **Action:** Run the apply command (using the ID from the folder name):

```
/openspec:apply add-user-auth
```

- **What Happens:** The AI now executes the plan. It reads the approved `tasks.md` and `specs/` deltas from that folder and writes the actual code in your codebase. Because the plan is explicit, the AI's output is no longer a guess; it's an implementation of a rigid spec.

Step 4: Archive the Change

After you have reviewed the AI-generated code, tested it, and merged it into your main branch, you close the loop.

- **Action:** Run the archive command:

```
/openspec:archive add-user-auth
```

- **What Happens:** OpenSpec "archives" the change. This process typically involves:
  a. Merging the "spec deltas" from the change folder into the main, source-of-truth `openspec/specs/` directory.

b. Moving the `openspec/changes/add-user-auth/` folder to an `openspec/archive/` directory for historical record-keeping.

Your project's main specification is now up-to-date, reflecting the new feature, and the whole process is documented.