

# Improved Session Persistence for a Multi-Agent Pipeline

Prepared by Patrick Skinner

---

## The Problem: Suggestion vs. Enforcement

## The Solution: A 3-Layer Defense

### Layer 1: The `CLAUDE.md` Directive

**What it is:** A behavioral suggestion. The sign on the door.

**Why it is needed:** The project-level `CLAUDE.md` is loaded into every session automatically. Placing a short, sharp, and unavoidable directive at the very top primes the agent's behavior before it receives any user input.

**Action:** Add the following to the top of `<project>/.claude/CLAUDE.md`.

Markdown

```
# CRITICAL WORKFLOW MANDATE
```

```
You are a pipeline orchestrator. You MUST NOT edit, write,  
or create any source code files directly.
```

```
ALL code changes, no matter how small, MUST go through the  
todo-orchestrator agent. To initiate any change, use the  
/todo skill.
```

- To fix a bug or add a feature: `/todo "Fix the login button overlap"`
- Do NOT use: `Edit src/components/Login.jsx` directly

```
Any direct attempt to edit files will be blocked by the  
system. You must follow the pipeline.
```

**Limitation:** This is still a suggestion. A sufficiently direct user prompt can cause the agent to bypass it. This is why Layers 2 and 3 are required.

---

### Layer 2: The `/todo` Skill

**What it is:** A reusable, on-demand skill that acts as the official front door to the pipeline.

**Why it is needed:** Instead of relying on the agent to recall a 10-step orchestration process from memory, a dedicated skill gives both you and the main agent a single, concrete, and easy-to-invoke starting point. Skills are lazy-loaded, meaning they consume no tokens until called, making this a token-efficient solution.

**Action:** Create a new skill file at `~/claude/skills/todo.md` .

Markdown

```
---
name: todo
description: "Initiates the full development pipeline for a
new task, bug fix, or feature. This is the ONLY correct way
to start a code change."
args:
  - name: task_description
    type: string
    description: "A clear, one-sentence description of the task."
---

# Todo Skill Invoked

User has requested a new task: "{{task_description}}"

Your instructions are to initiate the full development
pipeline. Spawn the todo-orchestrator agent to classify,
plan, and delegate this task. The orchestrator will research
the codebase and return a staging payload.

Do not attempt to implement the change yourself.
Launch the todo-orchestrator agent now.
```

**How to use it:** You simply types `/todo "Fix the sticky note color picker overlap"` and the skill handles the rest, launching the orchestrator automatically.

## Layer 3: The `PreToolUse` Hook

**What it is:** Deterministic enforcement. The security guard that cannot be bypassed.

**Why it is needed:** This is the most critical piece of the solution. Unlike `CLAUDE.md` rules, hooks are not suggestions. They execute code on specific events and cannot be overridden by the agent. A `PreToolUse` hook that fires before the `Edit` or `Write` tools will physically block the agent from editing files directly, making the wrong path impossible.

**Action:** Add the following to `.claude/settings.json` .

JSON

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "block",
            "message": "ACTION BLOCKED. You cannot edit files
directly. All code changes must be initiated through the
/todo skill to launch the orchestration pipeline. Please use
/todo to describe the change you want to make."
          }
        ]
      }
    ]
  }
}
```

**What happens:** When the main session agent attempts to call the `Edit` or `Write` tool, the hook fires, stops the action completely, and injects the block message directly into the agent's context. The agent reads the reason for the block and is forced to use the `/todo` skill as instructed. The desired workflow becomes not just the best path, but the only path.

**Advanced version:** A more granular hook can be configured to allow edits only when the agent's working directory is inside a designated story worktree (e.g., `.claude/worktrees/`). This would permit coder agents (quick-fixer, architect) to operate normally while still blocking the main session from making direct edits. This is the recommended next step once the basic block is confirmed to be working.

## How the Three Layers Work Together

The power of this approach is that each layer compensates for the weakness of the one before it.

Layer	Mechanism	Type	Failure Mode
<code>CLAUDE.md</code> Directive	Loaded into context at session start	Behavioral suggestion	Agent can ignore it under pressure
<code>/todo</code> Skill	Invoked explicitly by user or agent	Guided entrypoint	Agent may not invoke it if it forgets

PreToolUse Hook	Fires on every Edit / Write call	Deterministic enforcement	Cannot be bypassed or ignored
-----------------	-------------------------------------	------------------------------	----------------------------------

By combining all three, you'll create a system where the agent does not need to *remember* the pipeline. The structure and enforcement mechanisms guide it to the correct behavior every single time, regardless of how a session starts or how a user phrases their request.

## Summary of Actions

1. Add the **CRITICAL WORKFLOW MANDATE** block to the top of `<project>/.claude/CLAUDE.md` .
2. Create the `/todo` **skill** at `~/.claude/skills/todo.md` .
3. Add the **PreToolUse block hook** to `.claude/settings.json` .
4. *(Optional, next step)* Refine the hook to allow edits only inside worktree paths, enabling coder agents to operate normally.