/sp-writing-plans

Source: `~/.claude/skills/sp-writing-plans/SKILL.md`

---

name: writing-plans description: Use when you have a spec or requirements for a multi-step task, before touching code

Writing Plans

Overview

Write comprehensive implementation plans assuming the engineer has zero context for our codebase and questionable taste. Document everything they need to know: which files to touch for each task, code, testing, docs they might need to check, how to test it. Give them the whole plan as bite-sized tasks. DRY. YAGNI. TDD. Frequent commits.

Assume they are a skilled developer, but know almost nothing about our toolset or problem domain. Assume they don't know good test design very well.

Announce at start: "I'm using the writing-plans skill to create the implementation plan."

Context: This should be run in a dedicated worktree (created by brainstorming skill).

Save plans to: `docs/plans/YYYY-MM-DD-<feature-name>.md`

Bite-Sized Task Granularity

Each step is one action (2-5 minutes):

- "Write the failing test" - step
- "Run it to make sure it fails" - step
- "Implement the minimal code to make the test pass" - step
- "Run the tests and make sure they pass" - step
- "Commit" - step

Plan Document Header

Every plan MUST start with this header:

```
# [Feature Name] Implementation Plan

> **For Claude:** REQUIRED SUB-SKILL: Use superpowers:executing-plans to implement this plan task-by-task.

**Goal:** [One sentence describing what this builds]

**Architecture:** [2-3 sentences about approach]

**Tech Stack:** [Key technologies/libraries]

---
```

Task Structure

```
### Task N: [Component Name]

**Files:**
- Create: `exact/path/to/file.py`
- Modify: `exact/path/to/existing.py:123-145`
- Test: `tests/exact/path/to/test.py`

**Step 1: Write the failing test**
```

```python
def test_specific_behavior():
    result = function(input)
    assert result == expected
```

Step 2: Run test to verify it fails

Run: `pytest tests/path/test.py::test_name -v` Expected: FAIL with "function not defined"

Step 3: Write minimal implementation

```python
def function(input):
    return expected
```

Step 4: Run test to verify it passes

Run: `pytest tests/path/test.py::test_name -v` Expected: PASS

Step 5: Commit

```
git add tests/path/test.py src/path/file.py
git commit -m "feat: add specific feature"
```

```
## Remember
- Exact file paths always
- Complete code in plan (not "add validation")
- Exact commands with expected output
- Reference relevant skills with @ syntax
- DRY, YAGNI, TDD, frequent commits

## Execution Handoff

After saving the plan, offer execution choice:

**"Plan complete and saved to `docs/plans/<filename>.md`. Two execution options:**

**1. Subagent-Driven (this session)** - I dispatch fresh subagent per task, review between tasks, fast iteration

**2. Parallel Session (separate)** - Open new session with executing-plans, batch execution with checkpoints
```

```
**Which approach?"**

**If Subagent-Driven chosen:**
- **REQUIRED SUB-SKILL:** Use superpowers:subagent-driven-development
- Stay in this session
- Fresh subagent per task + code review

**If Parallel Session chosen:**
- Guide them to open new session in worktree
- **REQUIRED SUB-SKILL:** New session uses superpowers:executing-plans
```

Revision #5
Created 2026-02-18 08:39:57 UTC by John
Updated 2026-06-21 20:01:03 UTC by John