

THE FEATURE LOOP SOP

How to build with an AI agent without it spiraling: a 4-line brief, thinnest-slice-first, and 6 guardrails that keep you one step from a working commit.

THE PROBLEM

You tell your agent "build me a thing that captures voice notes and emails them to me." Sounds like one feature. Here is what actually happens.

The agent starts on voice capture. That needs a hotkey, so it wires up a hotkey tool. The hotkey needs an OS permission, so it detours into system settings. Then it decides screen capture would be nice too. Then auto-paste. Then a shortcut. Then it tests against your email and the send fails, so it swaps email providers. Two hours in, you have eight half-built directions, nothing runs end to end, and you cannot tell which change broke what.

Then the real trap springs. One piece breaks, say the microphone permission. The agent tries a fix. Fails. Tries another fix. Fails. Tries another. You are now ten cycles deep in whack-a-mole on a single broken dependency, and the agent keeps going because nothing told it to stop. There is no definition of "done," so there is no reason to stop. There is no last-known-good commit, so there is nowhere to roll back to. The whole session is unrecoverable and you throw it away.

That spiral is not a hard technical problem. It is a missing process. The agent will happily run in any direction you leave open, forever, because "helpful" with no rails means "expansive." This SOP is the rails.

THE PRINCIPLE

Never be more than one small step from a working commit.

That is the whole thing. Everything else here serves it.

Most people build features as one big push and only find out at the end whether it works. The agent amplifies this because it can generate a hundred lines a second and will gladly build the entire tree before running any of it. When it finally breaks, the failure could be anywhere in that hundred lines, and there is no green state to return to.

Invert it. A feature is not one build. It is a chain of tiny proven slices, each one committed the moment it works. You are always sitting on a version that runs. The next change is small enough that if it breaks, the fix is obvious or the rollback is one command. You trade the illusion of speed (lots of code fast) for actual speed (never stuck, never redoing a lost session).

The non-obvious part: this is a constraint you impose on the agent, not a hope you have about it. The agent does not naturally stop, scope down, or commit. You encode those as hard rules and the rules do the work.

HOW IT WORKS

The mechanism is a six-step loop wrapped in six guardrails. Run the loop for every feature. The guardrails are the tripwires that fire when the loop starts to spiral.

The loop, one pass per slice:

1. **Brief.** Before any code, fill four lines: Outcome (when this works, I can ____), Done test (the one thing I will observe to know it works), Out of scope (two or three things I am deliberately not doing this build), Thinnest slice (the smallest piece that proves the core idea). No code until these are filled. The brief is a decision, not paperwork: if you cannot name the one thing you will see when it works, you do not understand the feature well enough to build it yet.
2. **Build.** Build only the thinnest slice. One change at a time. Kill every "while I am here" impulse.
3. **Prove.** Actually run it and watch the real result. Never accept "that should work." Observed, not assumed.
4. **Checkpoint.** If it works, commit that slice with a clear message. That commit is your rollback point. Optionally log one line to a running story file, which becomes free build-in-public content.
5. **Next or done.** Does it hit the Done test from step 1? Ship it, close the build. Not yet? Back to step 2 with the next thinnest slice.

6. **Retro.** Once, when the feature closes or gets abandoned, write one line: what did the process teach me this time? This is the step that lets the loop improve itself instead of just producing features.

The engine of the whole thing is steps 2 through 4 tightened as far as they go: smallest possible change, prove it real, commit. That is the "one small step from green" principle turned into a repeatable move. The brief exists so the loop has a stop condition (the Done test) and a scope boundary (out of scope) before the agent gets momentum. The retro exists so the process gets sharper each time instead of repeating the same mistakes.

THE GOTCHAS

These are the traps that produce the spiral. Each one is a guardrail. They are non-negotiable because the moment you make one optional, the agent will route around it.

1. **Two-strike rule.** A step fails twice, stop digging. Step back, reassess, or roll back to the last green commit. Do not attempt a third fix on the same broken thing. This one rule alone kills the whack-a-mole spiral, because whack-a-mole is just "attempt number N+1 with no plan." Two strikes forces a change of altitude.
2. **Scope freeze.** New ideas that surface mid-build go on a "later" list, never into the current build. Finish the brief, then consider the new idea as its own brief with its own four lines. This is what stops one feature from metastasizing into eight.
3. **Control-test first.** Anything you depend on but do not control (an OS permission, an external API, a third-party app, a login) gets a five-minute isolated test before you build on top of it. Prove the dependency works alone first. Most catastrophic spirals are built on a dependency that was broken from the start, and you do not find out until you have stacked a feature on top of it. Test the foundation before you pour concrete.
4. **Always green.** Never more than one small step from a working commit. If the working tree is not in a known-good state, getting it back to green is the only thing you are allowed to work on. Nothing new until you are green.
5. **Package or it did not ship.** A reusable automation is not done until it is wrapped so it can be invoked by name, not left as a loose script you will forget exists. A script nobody can find or call is not a capability, it is a file. If you or your agent cannot invoke it by name later, it is

half-shipped. (Hard-won: an audit once scored our reusable capabilities near zero despite real, working automation, purely because it was all unpackaged loose scripts the system could not see.)

6. **A process ships with its check.** Any recurring process (a scheduled job, a cron, a poller) is not done until it has a check that reads the outcome in the world, never the process's own log. This is the sharpest one. A process cannot grade its own homework. We had a scheduled job whose log said "sent" every single day while nothing ever reached the phone. The log confirmed the code ran; it could not confirm the thing happened. Your check must read ground truth: a real database row, a file stamped with today's date, an HTTP 200, a live process, an actual received message. If the only evidence a thing worked is the thing telling you it worked, you have no evidence.

IMPLEMENT IT YOURSELF

Stack-agnostic. Works with any agent, any language, any repo.

1. **Write a brief template** and keep it where every build starts. Four lines: Outcome, Done test, Out of scope, Thinnest slice. Make filling it the literal first step. Paste it into your agent chat, or drop it as a `BRIEF.md` in the project folder.
2. **Add a standing instruction to your agent** (in whatever config file it reads at startup) that says: no build starts without the four-line brief; build the thinnest slice first; prove it runs before moving on; commit each proven slice; enforce the six guardrails. The agent needs the rules loaded before it starts, not reminded after it spirals.
3. **Make "prove it" a real step.** Require the agent to run the thing and report the observed result before it commits. Ban "this should work." If it did not run, it is not proven.
4. **Commit per slice.** One proven slice, one commit, clear message. Small commits are your rollback grid. This is the mechanical enforcement of "always green."
5. **Encode the two-strike rule explicitly.** Tell the agent: if a step fails twice, stop and step back or roll back, do not attempt a third fix. Agents will loop forever without this.
6. **Keep a "later" list** in the brief. When the agent proposes a mid-build extra, the answer is always "note it on the later list, continue the current slice."

7. **Control-test dependencies before building on them.** Before any feature touches an API, permission, or third-party tool, run a five-minute isolated test that proves that one dependency works alone.
8. **For anything recurring, write a check that reads the world.** Not the job's log. A separate check that queries the actual outcome. Store these checks somewhere you review them.
9. **Add a one-line retro** when a feature closes. What did the process teach you? Feed the recurring lessons back into the brief template and the agent instructions. The loop should get tighter over time.

THE COPY-PASTE TEMPLATE

Adapt freely. The four lines are the load-bearing part; do not skip them.

```
# BRIEF
```

```
Date: [YYYY-MM-DD]
```

```
Project: [where this lives]
```

```
## The 4 lines (no code until these are filled)
```

- Outcome: When this works, I can _____.
- Done test: I will know it works when I see _____.
- Out of scope (NOT this build):
 1. _____
 2. _____
 3. _____
- Thinnest slice: The smallest piece that proves the core idea is _____.

```
## Dependencies I do not control (control-test each FIRST)
```

```
> OS permissions, external APIs, third-party apps, logins.
```

```
> Prove each works alone before building on it.
```

```
- [ ] _____ -> tested working alone? [ ]
```

```
- [ ] _____ -> tested working alone? [ ]
```

```
## Later list (scope-frozen: each revisited as its own brief)
```

```
> Ideas that surface mid-build go here, never into this build.
```

```
- _____
```

```
- _____
```

```
## Checkpoints (one line per PROVEN slice, commit each)
```

- [] slice 1: _____
- [] slice 2: _____
- [] slice 3: _____

```
## Guardrails (non-negotiable, active for the whole build)
```

1. Two strikes -> stop and step back or roll back. No third fix on the same break.
2. Scope freeze -> new ideas go to the later list, not this build.
3. Control-test first -> prove every uncontrolled dependency alone before building
4. Always green -> never more than one small step from a working commit.
5. Package or it did not ship -> reusable automation gets wrapped so it is invocabl
6. A process ships with its check -> recurring jobs get a check that reads the worl

```
## Retro (one line, when this feature closes or is abandoned)
```

- What did the PROCESS teach me this time? _____

Drop this template where every build begins, load the guardrails into your agent's startup instructions, and hold the line on the four lines. That is the entire primitive. Receipts, not hype: you will feel it the first time an agent tries to spiral and the two-strike rule stops it cold with a green commit still sitting right behind you.