

Everyone Needs *a* Samwise

001

Claude Code *Mods*

starter guide

Three mods to install, one to build in three files, and a short safety checklist.

EVERYONENEEDSASAMWISE.COM/GUIDES/CLAUDE-CODE-MODS

CHECKED AGAINST CLAUDE CODE 2.1.295 DOCS ON 2026-10-08 (MODS SHIPPED IN V2.1.287); THE MODS SURFACE IS EARLY ACCESS AND MAY CHANGE.

01 WHAT A MOD IS

A plugin that runs inside Claude Code.

A mod is a plugin that changes how Claude Code looks and behaves. Skills, MCP servers, settings hooks and status lines work from outside Claude Code. A mod's handlers are functions that run inside it. Claude Code calls one when an event happens, such as a tool call, a submitted prompt, or a part of the interface being drawn, and the handler can watch the event, change it, or take it over.

That lets a mod draw a pane beside the transcript or a band above the prompt, with tabs, buttons and text fields. It can draw a line beside the spinner, and add a slash command that runs your function at once, with no Claude turn. Mods are on by default in Claude Code v2.1.287 or later in the terminal. They shipped in v2.1.287 on October 1.

THING	WHAT IT IS	PICK IT WHEN
<i>Mod</i>	A plugin whose handlers run inside Claude Code	You want a pane, a band above the prompt, a custom command, or to rewrite an event
<i>Skill</i>	A <code>SKILL.md</code> file of instructions Claude reads	You keep pasting the same instructions into chat
<i>MCP server</i>	An external process or service that gives Claude tools	Claude needs to reach an external system
<i>Settings hook</i>	A shell command, HTTP request, or prompt that Claude Code runs on a lifecycle event	You want to block, allow, or log an event with a script you already have

02 HOW A MOD WORKS

Event, hook chain, next.

You send a prompt and Claude takes a turn. Before Claude Code runs a tool, it fires an event called `tool.call`. Hooks on the same event form one middleware chain: each mod's `next` calls the following mod's hook. Every hook receives three arguments: the mods API as `$`, the event as `e`, and `next`. Calling `next(e)` passes the event down the chain and, at the end, to Claude Code's own behavior.



What a hook does with `next` decides which of three moves it makes:

- **Observe.** Do your work, then return `next(e)` unchanged.
- **Rewrite.** Call `next` with a modified copy of the event.
- **Answer.** Return a result without calling `next`. That short-circuits the chain, which is how a mod refuses a command. This hook, from the events page, refuses every Bash command:

ANSWER: FROM THE EVENTS DOCS

```
on('tool.call', { tool: 'Bash' }, async () => {
  // No call to next, so the command never runs
  return { deny: 'Bash is turned off in this project. Use the file tools.' }
})
```

Anything outside a hook's own code, like drawing, reading a file or running a process, goes through the mods API. A hook has no other way to do those things, which is why Claude Code can list what a mod does before you install it. Most of what Claude Code draws is an event too, so a mod can hook the spinner or the band above the prompt and add to it. The hooks inside one mod share the variables in its file, so one hook can count while another shows the count.

03 THREE MODS WORTH INSTALLING

Token Weather, Blast Radius, Replay Theater.

All three are sample mods from the `claude-code-playground` repo. They are Anthropic samples, not built into Claude Code, and you don't build them: you clone the repo and load them. The repository shares them as they are, without support, and the README says they are not an official Anthropic product.

Before you start

You need a Claude Code with mods. Mods shipped in v2.1.287; everything in this guide was captured on v2.1.295. Check yours:

TERMINAL

```
claude --version
```

Blast Radius opens its side pane only in a terminal about 144 columns wide or more. Narrower terminals get the same warning in the band above the prompt instead.

Try one for a single session

TERMINAL

```
git clone https://github.com/anthropics/claude-code-playground.git
cd claude-code-playground/claude-code/mods
claude --plugin-dir ./token-weather
```

`--plugin-dir` loads the mod's directory for one session without installing anything. Run it from that same `claude-code/mods` folder, and swap in `./blast-radius` or `./replay-theater` for the other two.

Validate before you install

Before you install anyone's mod, list what it hooks and which calls it makes, without running anything:

TERMINAL, FROM CLAUDE-CODE/MODS

```
claude plugin validate ./token-weather
claude plugin validate ./blast-radius
claude plugin validate ./replay-theater
```

Blast Radius and Replay Theater also print a `gating hook without .catch` warning. That is expected: if a hook fails before calling `next`, Claude Code skips it rather than blocking, so treat Blast Radius as a speed bump, not a lock.

Install for keeps

Run these from the same `claude-code/mods` folder. Add the clone's `claude-code/mods` directory as a local marketplace, then install by name. The marketplace points at your clone, so the mod stops loading if you move or delete the clone.

TERMINAL, FROM CLAUDE-CODE/MODS

```
claude plugin marketplace add ./
claude plugin install token-weather@claude-code-playground-mods --scope user
claude plugin install blast-radius@claude-code-playground-mods --scope user
claude plugin install replay-theater@claude-code-playground-mods --scope user
```

MOD 1 *Token Weather*

What it does. Draws a live forecast of your context window in the band above the prompt. After each turn it shows a weather word, the percent used, tokens used out of the window, a chart of the last 12 turns, and how much the last turn added. Clear under 25 percent, then Cloudy, Showers, Storm, and Compact soon at 90 percent and up. The numbers are real: the mod reads them with `$.session.usage()`, the same figures the status line shows.

Honest limits. The percentage is of the full window. Claude Code's own context notice counts against the auto-compact point, which is lower, so the two can differ (in the README's testing, the band read 81 percent when the notice read 90). The band updates once per turn, not during one. It reads 0 percent before the first response. History resets when the session starts or the plugin reloads. Only one mod can use the band above the prompt at a time.

MY TAKE *Keep this one. When Claude seems to forget something, the answer is usually boring, and this puts the answer on screen.*

MOD 2 *Blast Radius*

What it does. When Claude calls Bash with a risky command (`rm -rf`, `git reset --hard`, `git clean`, a force push, or a database migration), it holds the call, works out what the command would touch, and opens a pane with two buttons. It lists the files

that would be deleted with a count and total size, or the commits a force push would drop. Press **1** to Proceed or **2** to Cancel. Cancel refuses the command, and Claude sees the refusal and the reason. Wide terminals (about 144 columns) get the pane; narrower ones get the band above the prompt.

Honest limits. It reads the command text and does not parse shell fully. These are not caught: `$(...)`, aliases, `eval`, `bash -c "..."`, `xargs rm`, `find -delete`, and scripts that call `rm`. It only watches the Bash tool, holds one command at a time, and after you press Proceed the command runs as written. If a hook fails before calling `next`, Claude Code skips it and the next handler runs, and `claude plugin validate` flags gating hooks that have no `.catch`. The README calls it a safety net, not a permission system. For a hard block, use permission rules.

MY TAKE *Install it anyway, because a speed bump that shows you what is about to happen beats a wall you turn off.*

MOD 3 *Replay Theater*

What it does. After a turn where Claude edits files, a hint appears above the prompt: Replay, with the edit count (press **r**). Type `/replay` and a pane steps through the edits Claude made in the last turn, one diff at a time. Use Prev, Next and Close, or **n**, **p** and **c**. It records the Edit and Write tool calls as they happen and only watches: it never blocks, changes or delays an edit.

Honest limits. The diff shows up to 12 lines per step. For Edit, the diff is between `old_string` and `new_string`, not the whole file. An edit you then deny, or that fails, still shows in the replay. The replay lives in memory for the current session only and is lost on restart or plugin reload. Only one mod can use the band above the prompt at a time.

MY TAKE *Long runs are where people stop reading diffs. This gives you a quick review of exactly what changed, in order.*

A small mod is three files.

You don't need Node.js, a bundler or a build step, because Claude Code loads `.js` and `.ts` files directly. This one, `first-mod`, counts the tool calls Claude makes, shows the count beside the spinner while Claude works, and adds a `/tally` command that prints it. The code below is the real mod.

FILE 1: .CLAUDE-PLUGIN/PLUGIN.JSON

```
{
  "name": "first-mod",
  "version": "0.1.0",
  "description": "Counts Claude's tool calls, shows the
count beside the spinner, and adds a /tally command",
  "author": { "name": "ENAS demo" }
}
```

FILE 2: HOOKS/HOOKS.JSON

```
{
  "description": "The first-mod hooks module",
  "modules": ["/register.js"]
}
```

FILE 3: HOOKS/REGISTER.JS

```
// The count, shared by the hooks below
let calls = 0

// Claude Code calls this once when the mod loads
export function register(on) {
  // Runs when the session starts, before your first prompt
  on('session.start', async ($, e, next) => {
    await $.command.register({
      name: 'tally',
      description: 'Show how many tool calls Claude has made',
    })
    return next(e)
  })

  // Runs each time Claude is about to use a tool
  on('tool.call', async ($, e, next) => {
    calls += 1
    $.ui.invalidate('ui.render')
    return next(e)
  })

  // Runs when you type /tally, and only then
  on('command.run', { command: 'tally' }, async () => {
    return { text: 'Claude has made ' + calls + ' tool calls since this mod loaded' }
  })

  // Runs each time Claude Code draws the spinner
  on('ui.render', { component: 'Spinner' }, async ($, e, next) => {
    return next({ ...e, props: { ...e.props, suffix: ' · tool calls: ' + calls + '...' } })
  })
}
```

Run it with `claude --plugin-dir ./first-mod`. While Claude works, the spinner gets a suffix such as `• tool calls: 1...`. Afterwards, `/tally` answers with a line like `first-mod: Claude has made 2 tool calls since this mod loaded`; the count depends on what Claude did.

Check it without running it

TERMINAL

```
claude plugin validate ./first-mod
```

This checks the manifest and runs the same static analysis Claude Code runs when it loads a mod, without running your code or starting a session. For `first-mod` the output includes these lines:

OUTPUT, FROM THE CREATE DOCS

```
 › ./register.js hooks: session.start, tool.call, command.run{command=tally}, ui.render{component=Spinner}  
 › ./register.js calls: $.command.register, $.ui.invalidate
```

```
✓ Validation passed
```

If an event you meant to handle is missing from the `hooks:` line, Claude Code won't call that hook either.

Hot reload

Claude Code watches a directory loaded with `--plugin-dir` and hot-reloads the hooks module when a file in it changes. Each reload runs `register` again, so `calls` resets to 0.

Or ask Claude to write it

Describe the mod you want in an interactive session and Claude writes it, working from a built-in skill named `plugin-authoring`. You can load the skill yourself with `/plugin-authoring`. The docs' example prompt:

PROMPT, FROM THE CREATE DOCS

```
make a mod that shows the current git branch above the prompt
```

Claude writes the mod in `~/.claude/dev-mods/` followed by the session's ID, and asks whether to enable hot reloading for the session. Run `/plugin` and open the Installed tab to confirm it loaded. A mod Claude wrote loads only in the session that made it; copy its directory somewhere of your own to keep it.

05

SAFETY CHECKLIST

A mod is code with your permissions.

- ☐ **It runs with your permissions.** A mod can read and write files anywhere your user account can, start programs, and make network requests.
- ☐ **It can see a lot.** It can read your environment variables and settings files, see every prompt you send and every tool call Claude makes, and approve a tool call before you are asked.
- ☐ **It is not sandboxed.** If you turn on sandboxing, it isolates the Bash commands Claude runs, but a process a mod starts runs outside it.

- ☐ **It can't change the permission prompt.** A mod can restyle much of Claude Code's interface, but not the permission prompt.
- ☐ **Only install from authors you trust.** Install mods only from authors and marketplaces you trust.
- ☐ **Validate first.** Run `claude plugin validate` on the folder to list the events the mod hooks and the calls it makes, without running anything.
- ☐ **Samples come as they are.** The playground mods are shared without support.

Off switches

TO STOP

DO THIS

One mod

Run `/plugin`, open the Installed tab, and disable or uninstall its plugin

A session without your installed mods

Start it with `claude --safe-mode`. Built-in mods still load.

Every installed mod, every session

Set `"disableAllHooks": true` in `~/.claude/settings.json`. Your settings hooks and custom status line stop too.

06

SOURCES

- code.claude.com/docs/en/plugins/mods/overview
- code.claude.com/docs/en/plugins/mods/create
- code.claude.com/docs/en/plugins/mods/reference
- code.claude.com/docs/en/plugins/mods/events
- code.claude.com/docs/en/plugins/mods/admin
- github.com/anthropics/claude-code-playground, [claude-code/mods](https://github.com/anthropics/claude-code-mods)
- GitHub release v2.1.287 of anthropics/claude-code

CHECKED AGAINST CLAUDE CODE 2.1.295 DOCS ON 2026-10-08; THE MODS SURFACE IS EARLY ACCESS AND MAY CHANGE.

Everyone Needs a Samwise

WWW.EVERYONENEEDSASAMWISE.COM/GUIDES/CLAUDE-CODE-MODS