

CLAUDE CODE · CODE QUALITY · PONYTAIL

Find the code Claude didn't need to write, with Ponytail

Ponytail is an open-source plugin that makes your AI agent ask "does this need to exist?" before it writes code. Installing it in Claude Code, auditing an existing project, sorting the findings, deleting safely, and keeping it on for new code.

TIME	LEVEL	STEPS	PROMPTS
20 min	Intermediate	5	1

Agents like Claude Code write code fast. Sometimes they write more than you need: a component nothing renders, a package installed for something the browser already does, an abstraction with a single caller. This guide shows you how to find that extra code with Ponytail and delete it safely. I tried it on my own site, the dec5 site you're reading right now. Hit **Copy** in the top right of any command or prompt, paste it, keep going.

What we're making

We'll find the unneeded code in your project, delete it, and then stop it from piling up again in new code. First, why.

Unneeded code costs you three things:

1. **It's harder to read.** Everyone who opens the project (you, and Claude) first tries to work out what that code is for. If it's for nothing, that time is wasted.
2. **Maintenance.** Unused code still comes back to you as updates, type errors and build time.
3. **Packages mean attack surface and size.** Every installed package is a dependency that may ship a vulnerability one day. Some of them also grow the JavaScript your site loads.

What does Ponytail do? [Ponytail](#) is a single open-source (MIT) prompt. Before writing code, the agent climbs a ladder and stops at the first rung that holds:

1. Does this need to exist at all?
2. Is it already in this codebase?
3. Does the standard library do it?
4. Is there a native platform feature?
5. Does an installed dependency do it?
6. Can it be one line?
7. Only then: the minimum code that works.

The ladder doesn't replace understanding the problem. The agent reads the code and traces the flow first, then picks a rung. Input validation, error handling that prevents data loss, security and accessibility are never cut, at any level.

Example: you ask for a date picker. Without Ponytail, the agent may install a package and write a wrapper component around it. With Ponytail, it uses the browser's own field:

HTML

```
<input type="date">
```

About the numbers, honestly. Ponytail's page says "~54% less code, ~20% cheaper, ~27% faster". That was measured on one real FastAPI + React repo, 12 tasks, with Haiku 4.5, each task run 4 times, comparing the same agent with and without Ponytail. The effect is large on tasks with a real over-build trap (date picker: 404 lines down to 23) and close to zero on code that's already minimal. The author has acknowledged that an earlier single-shot "80-94%" figure was partly misleading. He also notes that on long-thinking models like GPT-5.5, cost can go the other way. These are the repo owner's own measurements on a small sample. Measure it yourself on your project with `/ponytail-audit`.

You'll end up with three things:

1. A ranked list of the unneeded code in your project
2. A separate branch (PR) with the parts you could safely delete
3. Ponytail switched on while you write new code

INGREDIENTS

- [Claude Code](#)
- A project tracked with Git. We'll do the deleting on a separate branch; without Git you have no way back.
- Your project's build (and, if you have them, test) command. We'll use it to confirm everything still works after deleting.

TOOLS

Claude Code

Ponytail

Git

01 Install Ponytail

Send these two commands in Claude Code **as two separate messages**. If you put both in one message, the install doesn't work.

TEXT

```
/plugin marketplace add DietrichGebert/ponytail
```

TEXT

```
/plugin install ponytail@ponytail
```

Once installed, Ponytail is on in every session at the default **full** level.

Only install from the official source. The author names two:

`DietrichGebert/ponytail` on GitHub and `@dietrichgebert/ponytail` on npm. Ponytail never ships `.exe` or `.dll` files; a copy that does is fake.

If you use Codex, Cursor, Copilot or another agent, the steps are in Ponytail's [INSTALL.md](#).

02 Audit your existing project

Start from a clean state. Commit any unsaved work, then open a new branch for the cleanup:

TERMINAL

```
git checkout -b cleanup/ponytail
```

Then in Claude Code:

TEXT

```
/ponytail-audit
```

`/ponytail-audit` scans the whole project and returns a numbered list, biggest cut first. Each line starts with a tag:

- `delete:` dead code, unused flexibility, speculative features
- `native:` code or a package doing what the platform already does
- `stdlib:` a hand-rolled version of something the standard library ships
- `reuse:` a copy of a helper that already exists in the project
- `yagni:` an abstraction with one implementation, a setting nobody sets
- `shrink:` the same logic in fewer lines

The list ends with `net: -N lines, -M deps`. **The audit changes nothing**; it only reports.

What did it find for me? On dec5 the audit took about 3.5 minutes. It found roughly 1,400 lines of dead code and 7 unused packages. The three biggest findings:

1. Homepage sections that were no longer rendered (Contact, Services, Work, Process, CreatorTopics and others), about 781 lines.
2. Packages that were never imported: `gsap`, `@react-three/fiber`, `@react-three/drei`, `react-hook-form`, `rehype-raw` and others.

3. Scroll-restore code that never ran (ScrollRestore and smooth-scroll), about 35 lines.

When I asked Claude "what percentage is that?", it said about 15%. On my own more complex projects I've seen it go up to around 35%. That's my own observation, not a measured statistic; your project may come out differently.

03 Read the findings and sort them

Don't apply the list as is. Split it into two groups first:

- **Safe to delete:** files nothing imports, packages listed in `package.json` but never used in code, functions nobody calls.
- **Ask first:** a script that's triggered by hand, a module loaded by file name or a string, shared code another project uses. You know about these; the agent may not.

Also ask Claude for its list of things it won't touch. On dec5, Claude kept lenis, three, next-intl and react-markdown aside because they're actively used. It asked me before deleting a module it wasn't sure about. Check that list; if something belongs there and isn't, say so.

You can have Claude Code do this sorting too:

PROMPT

We're going to apply the /ponytail-audit results together. Don't delete anything yet.

1. Split the findings into two groups: "safe to delete" and "ask me first". If a file might be used through a dynamic import, a string path, a script or a CI config, put it in the second group.
2. For each "safe to delete" item, search the whole project (including tests, config files and scripts) and show me it isn't used anywhere.
3. List the packages and files you won't touch, with the reason for each.
4. Show me the plan. Don't delete anything until I approve.

04 Delete and verify

Have the approved items deleted. Keep the deletion on its own branch and its own PR, with nothing else mixed in. If something goes wrong, you can undo it in one move.

After deleting, in order:

1. If you removed packages, reinstall dependencies (e.g. `npm install`). The lock file should update too.
2. Run the build. If it fails, the cause is most likely a file that looked dead but was actually used.
3. Run your tests and lint, if you have them.
4. Open the site or app and click through it by hand. Some things slip past the build: a page loads but a section comes up empty.

TERMINAL

```
npm install  
npm run build  
npm run lint
```

Command names vary by project; check your own `package.json`.

On dec5 the cleanup went into a separate PR this way: 1,060 lines and 7 packages removed. Not all of the 1,400 lines the audit found were deleted; for example, I deliberately left a script that can be triggered by hand. As I write this guide, that PR is still open and hasn't been shipped yet.

05 Keep it on for new code

The cleanup is a one-off. The real win is the extra code not piling up again. Once installed, Ponytail is on in every session; you can change its level:

TEXT

```
/ponytail lite  
/ponytail full  
/ponytail ultra  
/ponytail off
```

- **lite:** Builds what you asked for, and names the simpler option in one line if there is one. You pick.
- **full:** The default. The ladder is enforced: standard library and native features first, shortest change.

- **ultra:** The most aggressive level. Prefers deleting over adding, and questions the request itself.
- **off:** Turns it off.

`/ponytail` with no argument turns it on if it's off, and otherwise reports the current level.

Before each PR, you can also check the change:

TEXT

```
/ponytail-review
```

`/ponytail-review` looks only at the current change (the diff) and gives you a "these can go" list. Like the audit, it changes nothing.

Other commands:

- `/ponytail-debt`: Collects the deliberate shortcuts you've left as `ponytail:` comments in your code into one list, so "later" doesn't become "never".
- `/ponytail-gain`: Shows Ponytail's own benchmark averages. It's not a number calculated for your project; for that, use `/ponytail-audit`.
- `/ponytail-help`: A quick reference for the commands.

TIPS

A file that's in use looks "dead". Files used through a dynamic import, a file path passed as a string, a script or a CI config can look dead because nothing

imports them. Run the build after every deletion. If you're not sure, move that item to the "ask first" group.

Commands fail along the way. The terminal commands the agent runs can hit quoting or escaping errors (in zsh, for example). That's not a Ponytail finding; it's a broken command. Paste the error back to the agent so it can fix the command and retry. Don't read a search that died halfway because of a bad command as "nothing found".

ultra feels too aggressive. `ultra` prefers deleting over adding and questions the request itself. Stay on `full` while cleaning up. If you use `ultra`, read each suggestion one by one.

Applying a finding broke something. Since you're working on a separate branch, you can drop that branch and start over. That's why the clean branch in step 2 matters.

DONE?

- ☐ I installed Ponytail with two separate commands, from the official source only
- ☐ I committed my work and opened a separate branch for the cleanup
- ☐ I ran `/ponytail-audit` and read the list
- ☐ I sorted the findings into "safe to delete" and "ask first"
- ☐ I checked Claude's list of things it won't touch
- ☐ Build, tests and lint pass after deleting
- ☐ I opened the site or app and clicked through it by hand
- ☐ I opened the cleanup as its own PR

- ☐ Ponytail is on for new code, and I run `/ponytail-review` before each PR

Got stuck somewhere? Tell me on Instagram and I'll add it to the guide.