NEWS · APRIL 6, 2026

"Fixed Stuff" — A Love Letter to the Worst Commit Messages in History

By Dr. Holger Flick

> Developers have been writing terrible Git commit messages forever. AI just made that inexcusable.

Let's be honest with each other for a moment.

You've done it. I've done it. Every developer reading this has, at some point, looked at a staged set of changes, thought very hard about what to type, and landed on:

"fix"

Or perhaps the slightly more ambitious:

"wip"

Or the unhinged optimism of:

"final" — followed an hour later by *"final2"* — followed the next morning by *"ok actually final"*.

These are not commit messages. These are confessions. Evidence of a crime against future-you, against your teammates, and against the poor soul who will be staring at your Git history at 11 PM six months from now trying to figure out what broke production.

Why We Write Terrible Commit Messages

The reasons are understandable, even if the results are not.

You're in flow. You've just cracked a tricky bug, the solution is elegant, you're feeling good — and stopping to write a thorough commit message feels like pulling the parking brake on a moving car. So you type something fast and move on.

Or you genuinely don't know what to write. The change touched four different things. Do you summarize all of them? Pick the most important one? What even *is* the most important one?

Or — and this is the most honest answer — you just don't think it matters that much. It's a commit message. Who reads those?

Your teammates read those. Your future self reads those. And occasionally, a forensic investigator of your own codebase reads those at a time when clarity is absolutely critical and "quick fix" tells them precisely nothing.

Git Roasted: The Mirror We Deserved

Our friends at GitKraken recently built a little tool called Git Roasted that holds this mirror up with some well-earned snark.

You paste in your commit messages. It roasts them.

I fed it *"quick fix"* — a personal favorite from the hall of shame — and the verdict was swift and merciless. Commit Chaos Score: **90/100**. Archetype: **The Midnight Committer**. Future debug pain: *"Hot fix at 3 AM guaranteed."*

The translator section was the real highlight:

- **What you wrote:** quick fix
- **What your PM hears:** We're behind schedule and nothing was tested
- **What future you hears:** I regret everything about this Tuesday
- **What production feels:** The load balancer just flinched

The Reality Check flagged it as *unclear* ("no scope or context — this could literally be anything") and *missing* ("no indication of whether this was tested"). Suggested rewrite: `test: quick fix with improved clarity and context`.

It's funny. It's also completely, uncomfortably accurate. Go try it with your own greatest hits — I promise it will feel personal.

Funny Diagnoses Don't Write Better Commits. AI Does.

The roasting tool is a great laugh. But the real question is: why have developers been writing terrible commit messages for so long, and what's actually going to change that?

The answer isn't guilt. It isn't better documentation about conventional commits. It isn't a team policy written by someone who has clearly never been in a deadline crunch.

The answer is removing the friction entirely — and that's exactly what GitKraken Desktop now does with its AI-powered commit tooling.

Here's the scenario it solves: you've been working for two hours. You fixed a bug, refactored a helper function, updated an environment config, and added a test. Four logical changes, all sitting in one messy working directory. The old you would have typed `git add .`, written "updates", and called it done.

GitKraken Desktop looks at everything you've changed and does two things that used to require discipline and time:

First, it helps you split your changes into separate, logically grouped commits. Not one giant "misc stuff" commit, but clean, atomic commits — one per concern. The bug fix lives alone. The refactor lives alone. Your history becomes something a human can actually read and navigate.

Second, it writes the commit messages for you — real ones, with context, scope, and meaning. Not "fix", but something that tells the story of what actually happened and why.

This is the part where AI stops being a novelty and starts being genuinely useful to working developers.

Pull Requests That Actually Explain Themselves

The AI assistance doesn't stop at commits.

When you open a pull request in GitKraken, you get a complete, changelog-style description generated automatically — the kind of thorough write-up that explains what changed, what it affects, and why it was done. The kind of PR description that most developers intend to write and never quite get around to.

What you end up with is something that functions as a proper work report. At the end of a sprint, you can point directly to your PRs and have a detailed, readable account of exactly what you shipped and why it mattered. No

reconstruction from memory. No vague bullet points. Actual documentation, generated as a natural byproduct of doing the work.

The Real Cost of "wip"

We've normalized vague commit messages to the point where good ones feel like extra work. They're not — they're just work we keep deferring to a worse moment.

The cost shows up during code reviews, during debugging sessions, during onboarding, during incident response. Every "asdf" commit is a small tax you're levying on your future self and everyone else who touches that code.

The good news is that this is now a solved problem. The tools exist. The AI is right there in your workflow. Writing a proper commit message no longer requires stopping, thinking, and composing — it requires reviewing what the AI drafted and pressing confirm.

There's really no excuse left.

Not even "quick fix."

Holger Flick is a GitKraken Ambassador, independent software consultant, and Delphi-to-web migration specialist. He helps teams modernize legacy desktop applications and occasionally reminds developers that "wip" is not a commit message. Find him at holgerscode.com.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —