MY WEIRD PROMPTS

Podcast Transcript

EPISODE #262

# Beyond Git: Version Control for the Solo Creator

Published January 20, 2026 • Runtime: 21:14

https://myweirdprompts.com/episode/git-alternatives-solo-creators/

## EPISODE SYNOPSIS

Git was born out of a 2005 software crisis, designed to manage the massive Linux kernel—but is it the right tool for a solo blogger or developer? In this episode, Herman and Corn discuss why Git's architectural complexity can stifle creativity and introduce powerful, low-friction alternatives like Fossil, Jujutsu, and Radicle. Learn how to manage your project's evolution without the "merge conflict" headaches and find the workflow that actually fits your creative process.

## DANIEL'S PROMPT

**Daniel**

We've discussed collaborative development workflows and Git as a key tool for version control. It's impressive that Linus Torvalds created both Linux and Git. I've used GitHub repositories for various projects, including blog posts, to maintain cloud backup and lifecycle control. However, Git can be complex, and its features are often overkill for solo projects, especially when dealing with conflicts. For those interested in version control and tech documentation, what alternatives provide comprehensive cloud-based tracking without the complexity of Git? Additionally, let's discuss the history of Git.

# TRANSCRIPT

**Corn**

Hey everyone, welcome back to My Weird Prompts. I am Corn, and as always, I am joined by my brother and our resident deep-diver into all things technical and historical.

**Herman**

Herman Poppleberry at your service. Today we are tackling a prompt that actually hits quite close to home for us. Our housemate Daniel sent us a recording asking about the history of Git and whether there are better ways for solo creators to manage their projects without getting bogged down in the complexity of traditional version control.

**Corn**

It is a great question because I think we have all been there. You start a simple project, maybe a series of blog posts or a personal knowledge base, and you think, I should probably use version control for this. Next thing you know, you are staring at a merge conflict in a terminal window at two in the morning, wondering why you did this to yourself.

**Herman**

Exactly. It is like using a heavy-duty industrial crane to pick up a single grape. Git is incredibly powerful, but it was born out of an absolute crisis in the software world. It was never really designed to be user-friendly for the casual solo developer. It was designed for war.

**Corn**

Well, before we get into the alternatives for Daniel and everyone else listening who wants a smoother workflow, let us talk about that war. Daniel mentioned being impressed that Linus Torvalds created both Linux and Git. How did that actually happen? Was it just a side project he whipped up on a weekend?

**Herman**

Not quite a weekend, but it was shockingly fast. It is one of the most legendary stories in computing history. To understand Git, you have to understand the BitKeeper controversy of two thousand five. Before that, the Linux kernel was being managed using a proprietary tool called BitKeeper. The company that owned it, BitMover, allowed the kernel developers to use it for free, but there were strings attached.

**Corn**

Proprietary software for an open-source project as big as Linux sounds like a ticking time bomb.

**Herman**

It really was. The tension came to a head when a developer named Andrew Tridgell—the same guy who created Samba—decided to reverse-engineer the BitKeeper protocol. He wanted to create an open-source client so people could interact with the repositories without needing the proprietary software. Larry McVoy, the head of BitMover, was furious. He felt it violated the license agreement and decided to revoke the free status for all Linux developers.

**Corn**

So suddenly the most important software project on the planet had no way to manage its source code?

**Herman**

Precisely. It was April two thousand five, and the whole development process was basically at a standstill. Linus Torvalds looked at the existing options like Subversion or C-V-S, and he absolutely hated them. He famously said that C-V-S was a system designed to ensure that you lose your data, and he found Subversion just as flawed because it was centralized. He wanted something distributed, something that could handle the massive scale of the Linux kernel, and most importantly, something that was fast.

**Corn**

And that is when he went into his cave and came out with Git?

**Herman**

Pretty much. He started writing the code on April third, two thousand five. By April seventh, Git was self-hosting, meaning he was using Git to track the development of Git itself. It only took him about four days to build the core engine. By mid-April, he was using it to manage the Linux kernel release candidates, and by June, the first full kernel version managed entirely by Git was released. It was a sprint that changed software development forever.

**Corn**

Less than a week to build the foundation of modern coding. That is incredible. But the name itself, Git, that has a bit of a story too, right?

**Herman**

Oh, definitely. In British slang, a git is an unpleasant or contemptible person. Linus, in his typical self-deprecating but sharp humor, said he named it after himself. He said, I am an egotistical bastard, and I name all my projects after myself. First Linux, now Git.

**Corn**

That sounds like Linus. But here is the thing, Herman. While it was a miracle for the Linux kernel, it introduced a mental model that is fundamentally difficult for most people to grasp. Daniel mentioned using it for blog posts and repositories, but then running into conflicts that feel like overkill. Why is Git so notoriously difficult to learn?

**Herman**

It comes down to the architecture. Most earlier systems tracked changes to individual files. They would say, okay, line forty-five of this file changed. Git does not do that. Git thinks in snapshots of the entire project. It uses something called a Merkle tree, which is a mathematical structure where every file and every directory is hashed into a unique identifier.

**Corn**

So every time you commit, you are essentially taking a photo of the entire house, rather than just noting that you painted the kitchen door?

**Herman**

Exactly. And because it is distributed, every single person has the entire history of that house on their own computer. The complexity comes when you try to merge two different histories. If I painted the door blue and you painted it red, Git looks at the history and says, I do not know which one of you is right. Resolve this. For a solo developer, that feels like a lot of overhead because you are often just working on one branch, but you still have to deal with the staging area, the local repo, and the remote repo.

**Corn**

Right, the staging area! That is the one that always trips people up. Why do I have to add the file before I commit it? Why can I not just save it?

**Herman**

It is that extra step of intentionality that makes sense for a team of thousands but feels like a chore for one person. You are essentially saying, okay, out of all the mess I made in my workshop today, only these three specific tools are ready to be put back on the shelf. If you are just writing a blog post, you probably just want a save button that happens to remember what the document looked like ten minutes ago.

**Corn**

This leads perfectly into Daniel's question about alternatives. If Git is the heavy-duty crane, what are the more agile tools for someone who wants version control without the headache? You have mentioned Fossil to me before. Is that still a viable contender in twenty twenty-six?

**Herman**

Fossil is actually my favorite recommendation for solo projects. It was created by Richard Hipp, the same genius who gave us S-Q-L-ite. In many ways, Fossil is the anti-Git. While Git is a collection of small, complex tools that you have to string together, Fossil is a single executable file. It is tiny—about eight megabytes—and it contains everything you need. It is very much alive and actively maintained.

**Corn**

When you say everything, what do you mean?

**Herman**

I mean it includes the version control, a built-in wiki, a bug tracker, a forum, and even a web interface. When you run Fossil, it starts a local web server, and you can manage your whole project through a browser if you want to. And because it is built on top of an S-Q-L-ite database, your entire project history is just one single file on your hard drive.

**Corn**

That sounds much more manageable for a solo creator. If I want to move my project to a new laptop, I just grab that one file?

**Herman**

Exactly. No hidden dot git folders scattered everywhere, no complex setup. And it has a feature called autosync. In Git, you have to manually pull and push changes to a server. In Fossil, the moment you commit a change, it automatically tries to sync with your server if you have one set up. It feels much more like the seamless cloud experience we are used to with things like Google Docs, but with the power of a real versioning system.

**Corn**

That sounds perfect for technical documentation. But what about people who still want to use GitHub because that is where their community is, but they hate the Git interface? Is there a middle ground here in twenty twenty-six?

**Herman**

There absolutely is, and it is the biggest trend in developer tools right now. It is called Jujutsu, or just J-J for short. It is a version control system that is fully compatible with Git repositories, but it completely reimagines the user experience. It was created by a developer at Google who wanted to fix everything that makes Git frustrating.

**Corn**

How does J-J make it easier?

**Herman**

It gets rid of the staging area entirely. In J-J, your working directory is always a commit. As you type, it is essentially taking snapshots in the background. You never have to remember to git add. It also handles conflicts much more gracefully—it does not stop your workflow. It just records the conflict as a special state that you can fix whenever you are ready. It is the perfect tool for someone who wants the power of Git's ecosystem without the Git-headache.

**Corn**

That is a great middle ground. But let us look at the documentation side of things, because Daniel specifically mentioned using GitHub for blog posts. For a lot of writers and researchers, the version control they really need is not about code, it is about ideas.

**Herman**

Right, and that is where we see a shift toward tools that integrate versioning into the writing environment itself. Have you seen how people are using Obsidian Sync lately?

**Corn**

I use it every day! For those who do not know, Obsidian is a markdown-based note-taking app. It has a plugin called Obsidian Git, which basically runs Git in the background for you. It automatically commits your notes every few minutes and pushes them to GitHub.

**Herman**

See, that is a great middle ground. You get the safety and the cloud backup of GitHub, but you never actually have to touch a terminal. It handles the snapshots for you. But even simpler than that is just using the built-in version history in apps like Notion or even Google Docs. For most solo blog projects, the ability to see a list of versions from the last thirty days and click restore is usually enough.

**Corn**

But is that enough for someone who wants lifecycle control? Daniel mentioned wanting to track the evolution of a project. If I use Google Docs, I can see the changes, but I cannot really branch off and try a new experimental version of a blog post without cluttering up my workspace.

**Herman**

That is the trade-off. Branching is the killer feature of Git, Fossil, and J-J. It allows you to say, I am going to try writing this entire chapter from the perspective of a different character, and if it sucks, I can just delete that branch and my main work is untouched. Most cloud-based writing tools do not have a concept of branching. They are linear.

**Corn**

So if you want branching without the Git-pain, where do you go?

**Herman**

Honestly, I would point Daniel back to Fossil or a tool called Radicle. Radicle is a peer-to-peer version control system. It is designed to be decentralized without needing a middleman like GitHub. It is very focused on the sovereign developer—someone who wants to own their data completely and collaborate without a corporate gatekeeper.

**Corn**

I like the sound of that. But let us talk about the second-order effects of these tools. When we make version control too complex, does it actually stifle creativity? I have found myself hesitant to make a change sometimes because I do not want to deal with the potential mess in my repository.

**Herman**

That is a profound point, Corn. It is the friction of the tool. If the tool is supposed to be a safety net, but the safety net is covered in barbed wire, you are going to be afraid to jump. This is why I think for solo projects, we should prioritize low friction over maximum power. If you are spending twenty percent of your time managing your tools, that is twenty percent of your time you are not spent writing or coding.

**Corn**

Exactly. And the history of these tools shows that they were built for massive collaboration. But the needs of a single human living in a house in Jerusalem, trying to organize their thoughts, are very different from the needs of ten thousand developers working on a kernel.

**Herman**

It is the difference between a library and a personal notebook. A library needs a strict indexing system, a checkout process, and security. Your notebook just needs a pen and a way to flip back a few pages. I think for most of our listeners, the best version control is the one that happens automatically. Whether that is a script you write to back up your folder to the cloud every night, or using a dedicated tool like Fossil or J-J that stays out of your way.

**Corn**

I also want to touch on the idea of technical documentation specifically. Daniel mentioned cloud-based tracking. In the professional world, we are seeing a move toward what they call docs-as-code. This is where you write your documentation in markdown and store it in a repository alongside your code.

**Herman**

Yes, and that is where Git really shines because it allows your documentation to stay in sync with your software versions. But for a solo blogger, I would actually suggest looking at platforms like Ghost or even some of the newer A-I-integrated writing platforms. They often have versioning built into the database level. You do not need to manage files at all.

**Corn**

It is funny how we have come full circle. We started with Linus Torvalds manually applying patches from emails because he did not like the existing tools, then we went to this hyper-complex Git world, and now we are looking for ways to hide all that complexity behind simple interfaces again.

**Herman**

It is the cycle of technology. We build powerful abstractions, and then we build simpler abstractions to hide the powerful ones. But knowing the history helps you appreciate what is happening under the hood. When you see a conflict in Git, you are seeing the ghost of the BitKeeper controversy. You are seeing a system that was built to ensure that no single company could ever hold the world's software hostage again.

**Corn**

That is a great way to put it. It is a tool of liberation, even if it feels like a tool of frustration sometimes. So, if we were to give Daniel a concrete takeaway, what would it be?

**Herman**

I would say, Daniel, if you want the power of versioning without the Git-headache, download the Fossil executable. Spend thirty minutes learning its basic commands—open, commit, and U-I. It will give you a single-file database for your project, a built-in wiki for your notes, and it will never ask you to resolve a complex rebase unless you really, really go looking for trouble.

**Corn**

And if you want to stay on GitHub but want a better experience, look into Jujutsu. It is the future of how we interact with Git repositories.

**Herman**

Precisely. And remember, version control is there to serve you, not the other way around. If a tool makes you want to work less, it is the wrong tool, no matter how industry-standard it is.

**Corn**

That is a lesson that applies to so much more than just software. Well, I think we have covered the bases here. From the four-day miracle of Linus Torvalds to the eight-megabyte simplicity of Fossil.

**Herman**

It is a fascinating evolution. And honestly, I think we are going to see even more changes in the next few years as A-I starts handling the merging and conflict resolution for us. Imagine an A-I that looks at our two different versions of a blog post and says, hey, I see what you both were doing here, here is a version that combines the best of both.

**Corn**

That would certainly save me some late-night headaches. Before we wrap up, I want to say a big thank you to Daniel for sending in this prompt. It gave us a great excuse to dig into some tech history that we both love.

**Herman**

Absolutely. It is always fun to talk about the dramas that shaped the digital world we live in.

**Corn**

And for everyone listening, if you have been enjoying My Weird Prompts and our deep dives into these topics, we would really appreciate it if you could leave us a review on your podcast app or on Spotify. It genuinely helps other curious people find the show.

**Herman**

It really does. We love seeing the community grow. You can find all our past episodes at our website, myweirdprompts.com. There is also a contact form there if you want to send us a prompt of your own.

**Corn**

We are also on Spotify, so make sure to follow us there so you never miss an episode. We have a lot of interesting topics lined up for the coming weeks.

**Herman**

Thanks for spending some time with us today. This has been My Weird Prompts.

**Corn**

Until next time, stay curious and don't let your tools get in the way of your work. Goodbye everyone!

**Herman**

Bye!