The GitHub Blog

April 7, 2020

Community, Open source

Celebrating 15 years of Git: An interview with Git maintainer Junio Hamano

Jeff King

In celebration of Git's 15th anniversary, GitHub's Jeff King (@peff) interviewed Git's maintainer Junio Hamano about Git's 15 years and what's coming next. Jeff King is a distinguished Software Engineer at GitHub and has worked on scaling and maintaining Git on GitHub since 2011, but he's also been an active contributor to the Git project since 2006. Both Jeff and Junio serve on Git's project leadership committee at Software Freedom Conservancy.

Junio, thanks so much for chatting with us as we celebrate Git's 15th anniversary. For the benefit of our readers, and

because I got this wrong for the first several years of working with you, what is the correct pronunciation of your name?

It's pronounced like the sixth month of the year in Spanish, which by the way is not really my name. It's actually the name of the same month in some other language, but I tell it only to those who I meet in person.

How long have you been working with Git? What's your first memory of Git?

I wish I could claim that I've been here from the very beginning, but I think I probably got involved about a week after Linus announced that he had something that remotely resembled the first version of Git on April 7 in 2005. I got the tarball soon after hearing there was something "interesting" happening, and then read everything in a single sitting. It was small enough to read in two hours and understand all that was going on in the code. I remember that I was very impressed by the simplicity of the design and the clarity of the code.

There were many new version control projects starting around 2005. What made you choose to use and work on Git?

Along with my employer at the time, I personally was benefiting from the work done by the Linux kernel project. Linus was taking a "vacation" from it, out of necessity, to build some version control they could use, and I knew a bit about patch and diff. My initial motivation was to join other people who were helping that effort, hoping that it would help Linus declare victory sooner and go back to his kernel work.

In that sense, there wasn't any "do I help Git, or do I help some other system?" choice. It was purely "chase Linus out of the distraction back to the kernel", although Linus seemed to have also enjoyed the intense ride in the first three months until he gave the maintainer role to me at the end of July 2005.

How did you become the Git maintainer? What has that experience been like?

By starting as an individual contributor, working hard, and waiting for an eventual announcement from Linus that he's picked a successor. 😉

In earlier days, there were a lot of holes in the feature set. There were many great developers, many from the Linux kernel community, and there's always more than one way to design a

wanted feature, so the development was sometimes like a rather chaotic competition. Not only did you need to come up with a well-designed feature that's implemented well, you also needed to present it better than others who may be working on a similar feature. And preferably, you'd do it faster than other features that may contradict what you were doing.

What do you believe are the best features of Git? What's something you're most proud of as Git's maintainer?

I think the best features are yet to come 😉 but among those that I was heavily involved in, rename and rewrite detection in the diff engine and the way blame works to follow the origin of lines across file boundaries (when told to do so) are my favorites.

But what I'm more proud of, as the maintainer, is how our development community came to be, full of great developers (I won't name names) from different backgrounds, working for different employers, having different agendas, but still work together to make progress. I'm also proud about how I, and other longer-time contributors, trained ourselves and others to describe the changes we're making. As a result, our "git log" has become very helpful for developers.

There are many software tools that build on top of or integrate with Git. Do you have a favorite Git-related project?

As a non-GUI person, good old "tig" (curses based UI for viewing "git log" and running "git blame", among other features) has always been, and still is, my favorite.

If you had a magic wand, what part of Git would you fix or change?

I'd probably go back to 2005 and tell Linus that it's not a good idea to reuse the command line revision parser he wrote for "git rev-list" (underlying machinery of "git log") to "git diff", and the command line should explicitly check for and forbid "diff A..B" notation. The command line syntax that allows "diff A..B" cannot be easily taken away, unfortunately. End-users who learned Git in the beginning, or end-users who saw somebody who learned Git from the past use that notation, and continue to use it even after being told that it's illogical to express comparison between two endpoints A and B as if it's a range starting at A and ending at B.

Everything else, like making the index a hierarchical data structure instead of a flat one, can still be a disruptive change and would take a lot of work to discard what we've done and redo it in a new way. But I think we can replace it without disrupting end users—not that command line syntax, unfortunately.

How do you think Git has transformed how developers build software?

I don't know 😉. From very early days, we kept saying that our difference is that you can work in a distributed way, and sometimes people took it to mean that you'd sync before boarding a plane, work locally and then sync back after landing. While it's still true, I'm not sure if that kind of "distributed" matters as much as before, especially in the parts of the world where connections are ubiquitous.

Sometimes people took "distributed" to mean that it's very easy to fork, and you're not limited by who hosts the authoritative copy of the project, and other constraints. While this is true, what you forked eventually would need to be merged back if you, and those who worked or are still working on the original, were to collaborate. It shouldn't be surprising that projects still depend on having a central repository and everybody collaborates to advance its (often single) history.

Another beauty of a "distributed" development style is that it allows us to completely separate the act of committing and making the result public, and I think that aspect of "distributed"-ness had the biggest impact. It helped imperfect (read: all of us humans) programmers pretend as if they were perfect, by allowing them to take snapshots while they make progress in the form of their "private" commits, and then make the end result presentable in a separate "polishing" session, using interactive rebases, etc. before publishing their achievements. This may not be the ideal straight-line trajectory to the final solution, but may be more like a drunk stumbling around until arriving at the right solution by chance. Moderately skilled programmers with good discipline took advantage of these tools and learned to pretend to be super programmers who do not make silly mistakes in public. After doing so for a long time, they no longer need to pretend and actually have become super.

Of course, tools alone won't make users better, so programmers without good discipline or good taste may not be taking full advantage of the chance to proofread and polish using interactive rebases, and their output may have stayed just "moderately good". But overall, I think the average quality of software would certainly have gone up by having a tool and a chance to easily polish before giving it to the public.

If you're interested in learning more about Git development and its community, subscribe to Git's mailing list or check out the Git repository on GitHub.

Related posts

January 21, 2021

Community

The best of Changelog • 2020 Edition

 Michelle Mannering

December 23, 2020

Open source

Highlights from Game Off 2020

 Lee Reilly

December 22, 2020

Open source

Let’s talk about securing open source projects

 Hauwa Otori

GitHub

Product

Features

Security

Enterprise

Platform

Developer API

Partners