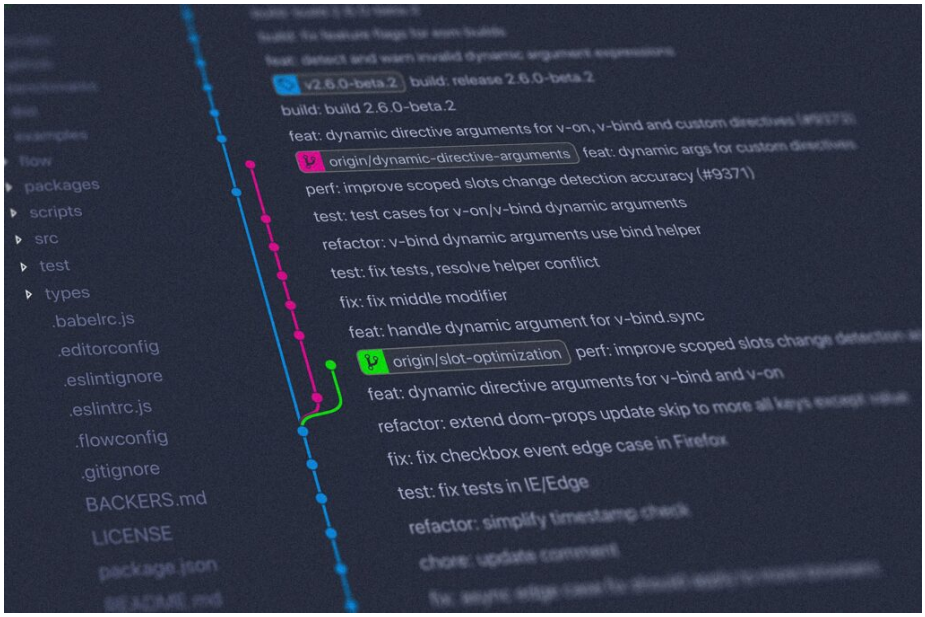
Issue 066: Version Control

Adrian Kosmaczewski

March 4th, 2024



Welcome to the sixty-sixth issue of *De Programmatica Ipsum*, about *Version Control.*

In this edition:

- We analyze the rise in popularity of Git[1] and how it eclipsed everything else.
- In the Library section[2], we review "Pragmatic Version Control Using Git" by Travis Swicegood[3].
- In our Vidéothèque section[4], we watch Linus Torvalds explaining Git[5] in a 2007 Google TechTalk.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Colin Powell, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Dr. Juande Santander-Vela, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, and Szymon Licau.

Enjoy this issue! Please subscribe to our free newsletter[6] to stay updated about new releases, share the articles on social media, or contribute[7] if you would like to support our work with a donation via Liberapay[8].

[1] https://deprogrammaticaipsum.com/twenty-years-is-nothing/
[2] https://deprogrammaticaipsum.com/category/library/
[3] https://deprogrammaticaipsum.com/travis-swicegood/
[4] https://deprogrammaticaipsum.com/category/videotheque/
[5] https://deprogrammaticaipsum.com/linus-torvalds/
[6] https://deprogrammaticaipsum.com/newsletter/
[7] https://deprogrammaticaipsum.com/contribute/
[8] https://liberapay.com/

Cover photo by Yancy Min[9] on Unsplash[10].

[9]https://unsplash.com/@yancymin?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[10]https://unsplash.com/photos/a-close-up-of-a-text-description-on-a-computer-screen-842ofHC6MaI?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Twenty Years Is Nothing

Adrian Kosmaczewski

March 4th, 2024

In a previous edition of this magazine[1], we argued that English was so pervasive in our industry, nobody even questioned its use anymore. The same can be said of Git[2]. It is difficult to imagine that merely twenty years ago, the landscape of source control tools was more diverse, and the choice of one such tool was much more complicated than today. Actually, Git was not even on the map yet. Before debating whether the hegemony of Git is good or bad, let us go back in time for a little while.

In one of the most famous tangos[3] of all time, Carlos Gardel[4] famously sings

> To feel... that life is a breath of fresh air,
> that twenty years is nothing,
> that, feverish, the gaze
> wandering in the shadows
> seeks and names you.

Twenty Years Ago

The second edition of "Code Complete" by Steve McConnell[5] was published in 2004. On page 668 of this massive 900-page volume, we find the *only* reference to the subject of source control in the whole book: about three quarters of a page long. Nothing else. ChatGPT

[1] https://deprogrammaticaipsum.com/the-winner-takes-it-all/
[2] https://git-scm.com/
[3] https://en.wikipedia.org/wiki/Volver_(song)
[4] https://en.wikipedia.org/wiki/Carlos_Gardel
[5] https://deprogrammaticaipsum.com/steve-mcconnell/

can easily summarize all of it with one phrase: "Version control software is good and brings several big benefits." Not a lot to phone home about. We are really far from GitOps at this point.

That same year, almost exactly 20 years ago at the time of the publication of this article, Subversion 1.0[6] saw the light of day. What was Subversion? Probably the shortest lived idea-with-good-intentions in the history of computing. See, Subversion (or `svn`) was supposed to be a better CVS (no, not the pharmacy[7], but this other thing[8]). "Better than CVS" meant, back in those days, to be transactional (databases[9], anyone?) and to have a somewhat better support for branches. We did not have higher ambitions back then, kids.

Linus Torvalds, however, did have higher ambitions. In 2004, the Linux kernel developers got in an increasingly strong disagreement over the use of BitKeeper[10], the proprietary, distributed version control system used to manage the kernel source code. So, what is a developer to do? Well, Linus has a tradition of writing the software everybody needs and nobody wants to start. He also has a tradition of naming things after himself[11]. Legend says that the first version of Git was written in a couple of weeks[12].

CVS

Never heard of CVS? It was a source control system that Joel Spolsky described in September 2000 as *fine* (emphasis his) in the first item of his eponymous Joel Test[13] for better software:

> I've used commercial source control packages, and I've used CVS, which is free, and let me tell you, CVS is *fine*.

Yes, the first step for better software was (shocker!) using source control software. (As a side note, I started my professional career as a software developer in 1997, and nope, we did not use source control, not even CVS. Yes, you guessed right: we just saved VBScript files locally and uploaded them via FTP. Too bad if we overwrote changes with one another: you only live once. I had to wait until 2002 to use a source control system for the first time, and for the curious among you, it was Rational ClearCase[14].)

Never heard of Joel Spolsky? Well, he was the co-creator of Stack Overflow, which I guess you *have* used at some point in your career. 24 years ago, Joel was one of the first influencers of the burgeoning field of software engineering. Think Kelsey Hightower, but with more controversial views. Or Steve Yegge, but with less controversial views.

Speaking about Stack Overflow, here is an example of the state-of-the-art of source control in 2008. One of the first-ever questions asked on the site, dated September 8th, 2008, asks precisely what version control system to use[15] for a single-developer workflow. (Interestingly, that question was asked precisely at the same time while in the real world, the 2008 financial crisis[16] was breaking havoc. Our industry lives in a bubble, no doubt about that. But I am

[6] https://en.wikipedia.org/wiki/Apache_Subversion
[7] https://www.cvs.com/
[8] https://en.wikipedia.org/wiki/Concurrent_Versions_System
[9] https://deprogrammaticaipsum.com/issue-61-databases/
[10] https://en.wikipedia.org/wiki/BitKeeper
[11] https://www.wordnik.com/words/git
[12] https://www.linuxjournal.com/content/git-origin-story
[13] https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/
[14] https://en.wikipedia.org/wiki/Rational_ClearCase
[15] https://stackoverflow.com/questions/49601/is-there-a-barebones-windows-version-control-system-thats-suitable-for-only-one
[16] https://en.wikipedia.org/wiki/2007%E2%80%932008_financial_crisis#2008_(September)

digressing, once again.)

> I'm trying to find a source control for my own personal use that's as simple as possible. The main feature I need is being able to read/pull a past version of my code. I am the only developer.

The replies to the question consist of a long catalog of pretty much every version control system known to mankind at that point in time.

Windows' Version Control Odyssey

But let us return to the year 2000: a few days before Joel Spolsky published his Joel Test, Mark Lucovsky[17] gave a talk titled "Windows: A Software-Engineering Odyssey"[18] at the 4th USENIX Windows System Symposium[19] in Seattle, Washington. Mr. Lucovsky was a member of the original Windows NT team from 1988 to the mid-2000s. The PowerPoint slides of the talk are still available online[20] at the time of this writing, and I seriously recommend you take a look at them.

Because part of the "odyssey" was, you guessed it, source control. On slide 14 you can learn that Windows NT 3.51 used an "internally developed" system... which was "on life support" by the time of Windows 2000:

> To keep a machine in synch was a huge chore (1 week to setup, 2 hours per-day to synchronize)

Oops. Not a great way to onboard new team members. Now you know why the Agile Manifesto[21], published in 2001, was so revolutionary. Thanks to Raymond Chen[22], arguably the most important lecturer of Windows history, we learn the name[23] of said internally developed system:

> In the early days, Microsoft used a homemade source control system formally called Source Library Manager, but which was generally abbreviated SLM, and pronounced *slime*. It was a simple system that did not support branching.

In slide 24 of Mark Lucovsky's PowerPoint slides, we learn that Microsoft took the decision to migrate the source code of Windows 2000 to something called "Source Depot". Raymond Chen agrees[24]:

> Shortly after Windows 2000 shipped, the Windows source code transitioned to a source control system known as Source Depot, which was an authorized fork of Perforce.

Why Perforce? The choice had to do[25] with the gigantic size of Microsoft Windows' source code base:

> The justification is perhaps less relevant than it once was, but Perforce tends to perform better on large repositories than Subversion. This is one of the rea-

[17] https://www.linkedin.com/in/mark-lucovsky-5280034/
[18] https://www.usenix.org/legacy/events/usenix-win2000/invitedtalks.html
[19] https://www.usenix.org/legacy/events/usenix-win2000/
[20] https://www.usenix.org/legacy/events/usenix-win2000/invitedtalks/lucovsky_html/Lucovsky.ppt
[21] https://agilemanifesto.org/
[22] https://devblogs.microsoft.com/oldnewthing/
[23] https://devblogs.microsoft.com/oldnewthing/20180122-00/?p=97855
[24] https://devblogs.microsoft.com/oldnewthing/20180122-00/?p=97855
[25] https://softwareengineering.stackexchange.com/a/85891

> sons Microsoft acquired a source license to Perforce to build Source Depot; NT's repository is a monster, and not many products, commercial or otherwise, could handle it.

Mark Lucovsky's summarized the benefits of Source Depot in two bullet points on slide 24 of his presentation:

- New machine setup 3 hours vs. 1 week
- Normal sync 5 minutes vs. 2 hours

Is the Microsoft Windows team still using Source Depot today? Apparently not. In 2017, we learned that Microsoft migrated all 300 GB of Windows source code to Git in an article on Ars Technica[26], which contains another gem describing the "Microsoft odyssey" in source control systems:

> Long ago, the company had a thing called SourceSafe, which was reputationally the moral equivalent to tossing all your precious source code in a trash can and then setting it on fire thanks to the system's propensity to corrupt its database.

(I can confirm. Sadly, I should say.)

Microsoft's adoption of Git, however, was not without hurdles[27], and led to the creation of the Git Virtual File System (GVFS) project:

> But Git isn't designed to handle 300GB repositories made up of 3.5 million files. Microsoft had to embark on a project to customize Git to enable it to handle the company's scale.

The Age Of Git

The infatuation of Microsoft with Git reached its peak in 2018, when it swallowed GitHub[28], the platform that arguably made Git mainstream. Three years prior, sensing *l'air du* temps, they had released Visual Studio Code[29], with integrated Git support.

GitHub introduced the concept of Pull Requests to the world as early as February 2008[30], a feature later adopted and adapted by GitLab[31], Gitea[32] (and its recent fork Forgejo[33]), and BitBucket[34], and which became the bread-and-butter for code reviews during the past 15 years. But the matter of fact is that GitHub also created a paradox in the world of Git: suddenly, a distributed source control system... became centralized. Some are understandably aghast[35] by this state of things.

We are in 2024, and Git is everywhere. The long evolution that led to the Git supremacy in the 2010s and, apparently, also the 2020s, can be summarized as a sequence of open-

[26] https://arstechnica.com/information-technology/2017/02/microsoft-hosts-the-windows-source-in-a-monstrous-300gb-git-repository/

[27] https://arstechnica.com/information-technology/2017/05/90-of-windows-devs-now-using-git-creating-1760-windows-builds-per-day/

[28] https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

[29] https://en.wikipedia.org/wiki/Visual_Studio_Code

[30] https://github.blog/2008-02-23-oh-yeah-there-s-pull-requests-now/

[31] https://docs.gitlab.com/ee/user/project/merge_requests/

[32] https://docs.gitea.com/next/usage/pull-request

[33] https://forgejo.org/docs/latest/user/pull-requests-and-git-flow/

[34] https://support.atlassian.com/bitbucket-cloud/docs/tutorial-learn-about-bitbucket-pull-requests/

[35] https://blog.edwardloveall.com/lets-make-sure-github-doesnt-become-the-only-option

source programs, one replacing the other: SCCS[36] in the 1970s, RCS[37] in the 1980s, CVS in the 1990s, and Subversion in the 2000s. To ensure smooth migration paths, Subversion could import CVS repositories, and Git can import Subversion repositories. But most importantly, version control systems migrated from local-only systems like SCCS and RCS, to client-server architectures (CVS and Subversion), to distributed systems, like Git and others, most notably Mercurial[38].

(Speaking about Mercurial, did you know that the Firefox developers recently decided to drop it[39] and use Git instead?)

These days, we are used to *cloning* an entire project on our computer, after which we can safely plug it off the network and continue writing software in a completely disconnected way. This simple paradigm was utterly and completely unthinkable 20 years ago. And guess what: your local repository also contains the full history of every single change ever made to your project. This was a feature that, naturally, client-server systems could never provide (spoiler alert: the server had the full history, while clients had only the `HEAD`, so to speak).

Git (and its cheap branching facilities) had a lasting impact on developer workflows. Vincent Driessen published in January 2010 a seminal article titled "A successful Git branching model"[40] introducing the world to the controversial[41] concept of git-flow. Why controversial? Well, because most opinions in the software industry are such. Now there is a GitHub flow[42] and an Atlassian Gitflow workflow[43] and many more branching workflows available.

Git repositories have become eventful, with every push, merge, or tag operation triggering a workflow somewhere. A whole industry has sprung up, including names such as Argo CD[44], GitHub Actions[45], GitLab CI/CD pipelines[46], and Gitea Runner[47], providing a new level of automation and convenience. The influence of Git is so strong in this space that the term GitOps[48] now refers to a whole subset of our industry. But you should not be using branches for deployments, you have been warned[49].

The question is simple now: what comes after Git? At this point, it is probably impossible to challenge the immense popularity of Git. I say "*probably*" because in our industry, it is impossible to predict the future. There are two interesting contenders worthy of mention: Pijul[50], written in Rust (although the project started with OCaml[51] a decade ago) or Fossil[52], written by the creators of SQLite. In this last case, the SQLite team provides a list of reasons[53] why not to use Git:

> Let's be real. Few people dispute that Git provides a suboptimal user experience.

[36] https://en.wikipedia.org/wiki/Source_Code_Control_System
[37] https://en.wikipedia.org/wiki/Revision_Control_System
[38] https://www.mercurial-scm.org/
[39] https://glandium.org/blog/?p=4346
[40] https://nvie.com/posts/a-successful-git-branching-model/
[41] https://web.archive.org/web/20111007075151/http://scottchacon.com/2011/08/31/github-flow.html
[42] https://docs.github.com/en/get-started/using-github/github-flow
[43] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
[44] https://argoproj.github.io/cd/
[45] https://docs.github.com/en/actions
[46] https://docs.gitlab.com/ee/ci/pipelines/
[47] https://about.gitea.com/products/runner/
[48] https://www.gitops.tech/
[49] https://codefresh.io/blog/stop-using-branches-deploying-different-gitops-environments/
[50] https://pijul.org/
[51] https://github.com/8l/pijul
[52] https://fossil-scm.org/
[53] https://sqlite.org/whynotgit.html

> A lot of the underlying implementation shows through into the user interface. The interface is so bad that there is even a parody site that generates fake git man pages[54].

While we wait for a better alternative, let us read the `man 7 giteveryday` page and call git-extras[55], SourceTree[56], TortoiseGit[57], Fugitive[58], Codeberg[59], and Magit[60] to the rescue. It seems that, whether we like it or not, we will probably be storing our source code in Git repositories for the next twenty years.

Cover photo by Praveen Thirumurugan[61] on Unsplash[62].

[54] https://git-man-page-generator.lokaltog.net/
[55] https://github.com/tj/git-extras
[56] https://www.sourcetreeapp.com/
[57] https://tortoisegit.org/
[58] https://github.com/tpope/vim-fugitive
[59] https://codeberg.org/
[60] https://magit.vc/
[61] https://unsplash.com/@praveentcom?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash
[62] https://unsplash.com/photos/a-book-and-a-small-figurine-on-a-desk-KPAQpJYzH0Y?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Linus Torvalds

Adrian Kosmaczewski

March 4th, 2024

The tech industry is a fertile ground for anecdotes starred by individuals qualifying as brilliant jerks, psychopaths, and other atrocious types of personality. Suffice to say that, from the height of their positions of supposed leadership, many chose the easy path of advancing a twisted agenda that feeds into their hubris, in the detriment of the wider advancement of society.

Let us name a few such individuals: Larry Ellison[1], Bill Gates[2], Mark Zuckerberg[3], Steve Jobs[4], Richard Stallman[5], John McAfee[6], Peter Molyneux[7], Travis Kalanick[8], Adam Neumann[9], Scott McNealy[10], Elon Musk[11], Jeff Bezos[12], Peter Thiel[13], and Evan Spiegel[14]. The list could go on and on, but for the sake of brevity, we will stop here. It should be no surprise to anyone that Linus Torvalds (or at least, his pre-2018 persona) belongs to this group, too.

Just like Reed Hastings[15] and Satya Nadella[16] have decided to get rid of brilliant jerks in their

[1] https://bobsutton.typepad.com/my_weblog/2007/03/the_assholiness.html

[2] https://www.businessinsider.com/after-talking-to-teens-last-year-about-anger-management-bill-gates-admits-that-some-of-his-famous-microsoft-tantrums-might-not-have-been-wise-2019-1?r=US&IR=T

[3] https://www.chieflearningofficer.com/2019/02/28/brilliant-jerks/

[4] https://www.businessinsider.com/steve-jobs-jerk-2011-10?r=US&IR=T

[5] https://thenextweb.com/news/free-software-icon-richard-stallman-has-some-moronic-thoughts-about-pedophilia

[6] https://www.wired.co.uk/article/dangerous

[7] https://www.somethingawful.com/news/five-biggest-jerks/

[8] https://www.forbes.com/sites/richkarlgaard/2014/12/10/do-jerks-always-win/?sh=80e106123251

[9] https://www.businessinsider.com/is-your-ceo-brilliant-jerk-or-both-how-to-tell-2019-10?r=US&IR=T

[10] https://www.zdnet.com/article/scott-mcnealy-has-a-big-mouth/

[11] https://www.leadershipnow.com/leadingblog/2020/05/no_brilliant_jerks_how_to_deal.html

[12] https://sea.mashable.com/tech/15683/how-bezos-became-an-asshole-and-7-other-things-we-learned-from-amazon-unbound

[13] https://www.electronicsweekly.com/blogs/mannerisms/genius/good-asshole-2014-10/

[14] https://techcrunch.com/2014/05/28/confirmed-snapchats-evan-spiegel-is-kind-of-an-ass/

[15] https://www.inc.com/jim-schleckser/why-netflix-doesn-t-tolerate-brilliant-jerks.html

[16] https://www.businessinsider.com/nadella-brilliant-jerk-phenom-in-tech-is-done-2019-5?op=1&r=US&I

own corporations, our world at large would benefit from discouraging people from treating others like shit. To a large degree, the types of personality that engage in such behavior do not realize the harm caused around them, in particular, in the long term. In the short term, such words can cause laughter, but they also impregnate the minds of those who attentively listen, and create a tacit sense of approval to engage in similar behavior.

This is a terrible, horrendously vicious circle, and it has to stop.

The Vidéothèque movie of this month[17], published in 2007, is a sad example of this situation. It was one of the Google TechTalks we mentioned last year[18] in this magazine. This one features Linus Torvalds explaining the history and architecture of Git, the most widely used source control system at the time of this publication but an entirely new kid on the block when the video was filmed.

The Good Parts

Let us start with the more benign parts. After diving into the history of the Git, the architectural talk is divided into three sections, each highlighting a particular aspect of the design and architecture of Git: its distributed nature, its high performance by design, and its absolute reliability. Taken together, these three aspects provide a simple answer to the enduring success that Git has had in the past 17 years, certainly explaining its becoming the *de facto* standard for source code management in our industry.

Speaking about its distributed aspect, and apart from the obvious (and dismissive) contrasts with CVS ("tarballs and patches are a much superior source control management system[19] than CVS") and Subversion, Linus mentions the Monotone[20] version control system as an interesting inspiration, but lacking in performance[21]. He also mentions Mercurial[22], as having mostly the same design as Git[23]. He highlights the importance of being offline[24] and how it literally means working in your own branch (we take branching for granted today, but 17 years ago, this was quite a revolutionary concept). Finally, he talks about how Git entered the enterprise space, first used as a client to access Subversion repositories[25] (something I have personally done in the past when Subversion was still around). Also, related to corporations, he highlights an important, often overlooked aspect: that in a corporate environment, centralized systems work better[26].

Regarding the performance part, there are two significant points. First, that the mistake made by the designers of Subversion was to focus on the wrong problem: branching was not the issue; merging was[27]. Second, that Git is fast because its code and data structures are really, really simple[28].

Finally, Linus mentions Git's reliability as a result of an actual break-in attempt[29] in the

R=T

[17] https://www.youtube.com/watch?v=4XpnKHJAok8
[18] https://deprogrammaticaipsum.com/google-techtalks/
[19] https://youtu.be/4XpnKHJAok8?t=171
[20] https://en.wikipedia.org/wiki/Monotone_(software)
[21] https://youtu.be/4XpnKHJAok8?t=721
[22] https://www.mercurial-scm.org/
[23] https://youtu.be/4XpnKHJAok8?t=1942
[24] https://youtu.be/4XpnKHJAok8?t=883
[25] https://youtu.be/4XpnKHJAok8?t=1993
[26] https://youtu.be/4XpnKHJAok8?t=2324
[27] https://youtu.be/4XpnKHJAok8?t=3174
[28] https://youtu.be/4XpnKHJAok8?t=3327
[29] https://youtu.be/4XpnKHJAok8?t=3567

Linux kernel source code. This resulted in a natural outburst of paranoia, which in turn brought the choice of hashing every single change with SHA-1 (it is worth remembering that this algorithm was considered cryptographically secure[30] in 2007, but "SHAttered"[31] in 2017). Git is not only designed to handle as many files as needed, but also so that developers can trust each one[32] of them, at all times. This was yet another revolutionary concept back then.

The Unnecessary Parts

Mixed with the juicy technical bits, there is a third aspect of this talk that makes it relevant, but for all the wrong reasons: a seemingly endless stream of criticisms, negativity, and (supposedly) self-deprecating remarks that end up being punches against anyone or anything Linus happens to dislike.

Let us enumerate some gems: "the designers of Subversion were complete morons[33]"; "nobody actually creates perfect code the first time around except me[34]"; "I think most of you are completely incompetent[35]"; "the KDE people are... well..."[36]; "when I merge from somebody, I trust them, but on the other hand, they might have stopped using their medication[37]"; *und so weiter*.

Yes, there is a "content advisory" at the beginning[38] of the talk. But none of this is needed, so let us focus on the good parts, of which there are many.

Balancing Acts

Maybe the issue at the core is that I am getting old. Because yes, I vividly remember laughing loudly when I watched this talk for the first time in 2007, and engaging in similar remarks and ways of thinking. Older entries of my own blog[39] are filled with such vitriol. I have given talks (some of which are on YouTube[40]) with such remarks. So, yes, I plead guilty to condoning such behavior, your honor.

I do not approve this anymore. With hindsight, I realize that we (as a society) do not need any more of this. And please refrain from arguing on Reddit or elsewhere that this is just me lacking humor. First, yes, I never had any. And second, it is objectively repugnant, no matter how you look at it. So, you have been warned; definitely watch this month's Vidéothèque entry[41] of Linus Torvalds, but be aware of foul remarks here and there.

But I have to be fair. 17 years have passed since this video, and in 2018 Linus has apologized[42] for years of ranting, and I think that as a result, the whole IT field (and in particular, the Linux galaxy) is a far better place because of that.

[30] https://youtu.be/4XpnKHJAok8?t=3465
[31] https://en.wikipedia.org/wiki/SHA-1#SHAttered_%E2%80%93_first_public_collision
[32] https://youtu.be/4XpnKHJAok8?t=3437
[33] https://youtu.be/4XpnKHJAok8?t=3081
[34] https://youtu.be/4XpnKHJAok8?t=1339
[35] https://youtu.be/4XpnKHJAok8?t=1685
[36] https://youtu.be/4XpnKHJAok8?t=2770
[37] https://youtu.be/4XpnKHJAok8?t=3250
[38] https://youtu.be/4XpnKHJAok8?t=480
[39] https://akos.ma/
[40] https://www.youtube.com/@akosma
[41] https://www.youtube.com/watch?v=4XpnKHJAok8
[42] https://arstechnica.com/gadgets/2018/09/linus-torvalds-apologizes-for-years-of-being-a-jerk-takes-time-off-to-learn-empathy/

Even more: I am positively in awe when faced with the incomparable talent of Linus and the brilliancy of his creations. His work has been a definitive positive source of change and wealth for the whole planet; mankind is *collectively better* because of his work. The Linux kernel *and* the Git version control system are both incommensurately valuable and brilliant, and have had (and will have) a transforming impact on the history of mankind. Each in itself would have been a remarkable achievement; taken together, they confirm the magnificence of the creative mind at their origin.

(There is also a possible argument to be made here, that Linux and Git are so good *precisely* because Linus cares so much, and is so passionate about his creations. I do not know him in person, but I think there might be a certain truth in this possibility, too. I certainly hope so.)

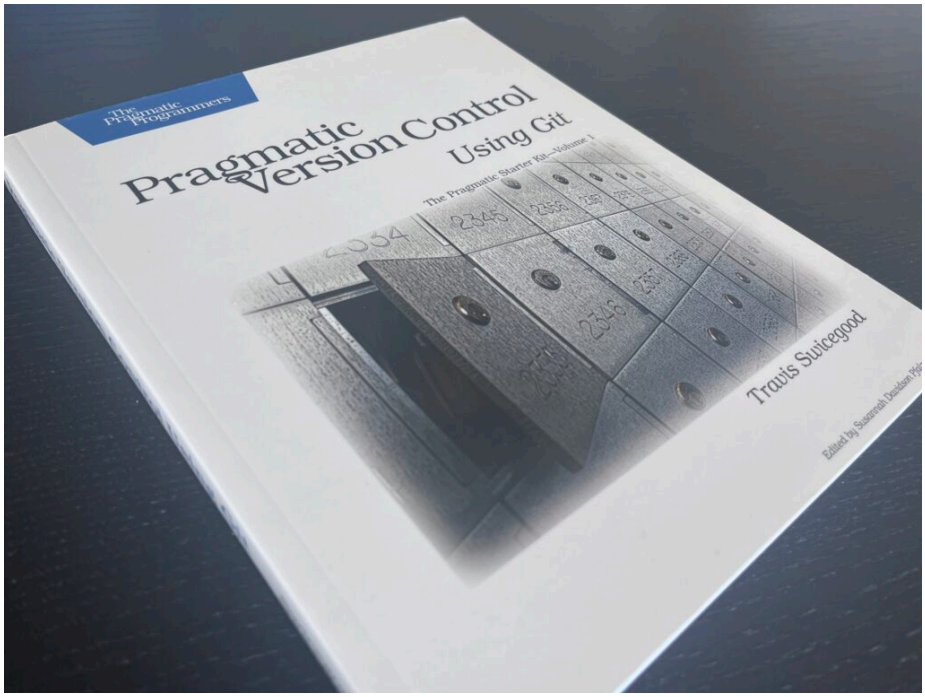
However brilliant, we do not need any more brilliant jerks spewing abusive thoughts towards other creators, particularly when their only sin was that of coming up with ideas that you just happen to dislike. We do not need such statements neither on record nor in other public or private spaces, no matter how good they are, or how big or small their impact on this planet may be. We should collectively let our work speak for ourselves, direct our cynicism and anger toward the resolution of actual world problems (there are a lot of those to choose from), and remember to have some respect for one another.

Cover snapshot chosen by the author from the video.

Travis Swicegood

Adrian Kosmaczewski

March 4th, 2024



A quick review of previous entries in the Library section of this magazine shows that it does not feature any book from The Pragmatic Programmers[1], for no other reason than gross oversight. We have discussed books from MIT Press, Addison-Wesley, O'Reilly, and many other publishing houses, and now it is time to solve this issue. This month we will elaborate on "Pragmatic Version Control Using Git"[2], a 2008 book by Travis Swicegood[3].

In other circles than software, such as physics or literature, the year 2008 would undoubtedly be called an *annus mirabilis*. While the "real world" was grappling with one of the most severe financial crisis[4] in history, the software industry saw a miraculous conjunction of factors, seemingly happening within months of one another. Let us enumerate a few: in March, the availability of the iPhone SDK. In April, the launch of GitHub. And in September, three major events: the announcement of the Google Chrome web browser on the 2nd, the launch of Stack Overflow on the 15th, and the release of Android 1.0 on the 23rd.

Couple all this with the meteoric rise in popularity of AWS and cloud computing in general, the phenomenon of social media, and the explosion of Web 2.0 and "Ajax"[5], with the ubiq-

[1] https://pragprog.com/
[2] https://pragprog.com/titles/tsgit/pragmatic-version-control-using-git/
[3] https://github.com/tswicegood
[4] https://en.wikipedia.org/wiki/2007-2008_financial_crisis
[5] https://en.wikipedia.org/wiki/Ajax_(programming)

uitous presence of libraries such as jQuery[6] and Prototype.js[7]. You could feel that software engineering was never going to be the same from that point onwards. It was undergoing a profound transformation, live, in front of our eyes.

It was both a terrific and terrible time to be a software developer, and Git appeared as one of the hottest new tools to check out. But Git was also substantially different from anything many of us had used up to that point. At least for me, the whole idea of a distributed source code management system was completely alien.

Up to that moment I had been a happy Subversion[8] user, hosting some of my code on Google Code[9] and even running Subversion repository servers on my own machine–I admit, a rather clumsy setup for a single developer. Before Subversion, I had used various source code management tools: Rational ClearCase[10] from 2002 to 2003, Visual SourceSafe[11] from 2004 to 2005, and CVS[12] from 2006 to 2007. (Yes, the list shows a clear regression; it seemed that the further I advanced in my career, the dumber and more unstable the SCM of my employer would be. *Anyway.*)

All the systems enumerated above are client-server, and thus, centralized; there is a server, somewhere, and a small client on my laptop allows me to check code out, commit changes, and voilà. Need to blame a file? Talk to the server. Want to see the full history log? Talk to the server. Want to create a branch? You get the gist.

I remember attending a talk about Git during a community meetup in March 2008, and rushing back home to order a book about it on Amazon (Kids: e-books were just starting to appear on our radar, as the original Kindle[13] was launched in November 2007. So, physical book it was.) There were not many books published about Git (yet) and I had always enjoyed reading books from The Pragmatic Programmers, so choosing this month's Library book was a no-brainer for me.

Needless to say, this was the perfect book to open my mind to the possibilities of Git. Just like in many other Pragmatic Bookshelf titles, the introduction to the subjects is gentle, complete, yet entertaining. The first chapter was particularly helpful, with its explanation of the differences between distributed and centralized version control, and the resulting workflows that were possible in each case.

The conceptual part was the most challenging one for me, but the author made it abundantly clear that it was not *that* hard. And the highly practical nature of the book, filled with examples, made understanding Git fast, straightforward, and simple. The detachable "Git Command Quick Reference" sat on my desk for months, providing a much-needed reference until most commands were engraved in my mind.

The rest is history. When I started my own business[14] in 2008, I got a paying GitHub account and used it to host the private Git repositories of all of my customers. But some of those customers were not entirely used to Git or distributed source code management for

[6] https://en.wikipedia.org/wiki/JQuery
[7] http://prototypejs.org/
[8] https://subversion.apache.org/
[9] https://code.google.com/
[10] https://en.wikipedia.org/wiki/Rational_ClearCase
[11] https://en.wikipedia.org/wiki/Microsoft_Visual_SourceSafe
[12] https://en.wikipedia.org/wiki/Concurrent_Versions_System
[13] https://en.wikipedia.org/wiki/Amazon_Kindle
[14] https://akos.ma/blog/running-akosma-software/

that matter, and asked me to push code into their Subversion repositories; no problem at all, `git svn` to the rescue[15].

Git was a godsend during those years of independent consulting. I was very frequently on the go, either on a plane or on a train, without any network connectivity, and Git allowed me to work perfectly well, committing, diff'ing, blaming (mostly me) and branching as much as I needed, at any time, in any place. As soon as a network connection became available, a simple `git push` would share those changes with my customers, and even better, trigger some automated task in a CI/CD system somewhere.

Beyond my own developer experience, one thing that changed in the software industry from 2008 onwards was that Git became the all-encompassing source code management tool, while others slowly disappeared into oblivion. New developers joining the workforce learned Git exclusively. New businesses started adopting Git and collaborating on GitHub or GitLab repositories. Popular software projects started opening up shop on GitHub. Text editors started providing built-in Git support. Everyone started assuming that Git was the big default SCM of the world of software. Well-known SCM packages slowly faded away. Companies providing commercial SCMs went bankrupt or pivoted to other markets.

During the past 16 years, I almost exclusively used Git for my code and my projects, both personal and professional. I did use Mercurial too, for a while at least, but for no other reason that my employer at the time was a heavy Python[16] user. They ended up migrating to Git at some point, the power of networking effects being too difficult to counter. Those forces even bent BitBucket[17], originally a Django web application hosting only Mercurial repositories, to drop Mercurial support in 2020. *Sign o' the times.*

The writing was on the wall. Git is the *de facto* standard, and it has become as ubiquitous as the English language, or water. Travis Swicegood's book still sits on my bookshelf; admittedly, I do not consult it so often as I did in those early days, but it represents a pivotal moment for my career–and truly, for the whole industry.

Cover photo by the author.

[15] https://git-scm.com/docs/git-svn
[16] https://deprogrammaticaipsum.com/the-state-of-python-in-2021/
[17] https://en.wikipedia.org/wiki/Bitbucket