Memories of Centralized SCMs

Adrian Kosmaczewski

2021-12-17

It might sound incredible to younger developers out there, but there was a time when Git did not exist. In retrospect, the fact that Git has reigned supreme in its category *for over 15 years* was previously unheard of. SCM systems came and went in a steady succession since the early 1980s.

Nowadays, Git has become the *de facto* standard tool; it is hard to remember a time when, during the onboarding of a developer, we had to learn the ins and outs of yet another Source Code Management (SCM) system. "This is how you checkout the project, this is how you commit a change, have fun."

In my first decade as a software developer, I worked with at least five different SCM tools *before* I even used Git for the first time: Rational ClearCase, Microsoft Visual SourceSafe, Microsoft Team Foundation Server, CVS, and Subversion. That's a new SCM tool every two years in average. All of them were centralized systems.

Yet, since 2008, I have seen only a handful of teams *not* using Git. The SCM wars have ended long ago, and Git has won. Yes, some teams are still using other tools; for example, I did work with a lonely team using Mercurial, but it was simply because it was written with Python, and for some teams that's a good enough excuse to adopt a tool instead of another.

Git has won, and in consequence we have switched the conversation to a higher level of abstraction. Teams debate about using either GitHub, GitLab, Gitea, or even Bitbucket for their GitOps needs. Because, yes, don't say DevOps out loud, that's soooooo 2009.

(By the way, Bitbucket now conveniently advertises itself as a "Git solution for professional teams" when they actually started as an alternative to GitHub based on Mercurial.)

Git's triumph brought some self-enforcing stabilization in the world of IDEs. Gone are the days when Mac OS X 10.2 "Jaguar" developers used Project Builder, an IDE featuring… native CVS integration. Xcode, the heir of Project Builder born in 2003, only included native Subversion support in version 3.0 (that's around 2009, iPhone OS 2.1, anyone?); it later added a native Git client during the 2010s, and now even features GitHub and GitLab integrations.

Git is so convenient and so prevalent these days that Visual Studio Code supports it off-the-box. Of course, you can add GitLens on top of it, but I don't think it's such a good idea. There's plenty of useful desktop Git clients, anyway: Tower, GitKraken, TortoiseGit, Sourcetree, Magit, fugitive.vim…

Then in 2011 Vincent Driessen invented git-flow and the Internet went bezerk with people screaming at each other how they were doing Git branching wrong. Because dogmas are like that.

What people don't remember, and I actually think it's a good thing, is how it felt working with non-distributed, centralized SCM systems like Visual SourceSafe or Subversion. Because, well, it was a sport.

We have to understand that businesses in particular were very afraid of conflicts caused by two developers editing the same file. Back in the 90's, "enterprise" SCMs allowed developers to "lock" a file in the server, so that nobody else but them could edit it, at all. Can you imagine that? I know, it's hard to picture, so here's a sequence of commands of the `scm-tool`, a ficticious… SCM tool that never made it big in this market.

```
$ scm-tool lock scm://server/project/file.cpp
$ vim project/file.cpp
$ scm-tool commit file.cpp -m "Changed things"
$ scm-tool unlock scm://server/project/file.cpp
```

Having seen this, here's a *Gedankenexperiment*: what happens if the last command in the snippet above is forgotten? And even worse, what if this happens on a Friday evening before the developer leaves for their well-deserved, long-dreamt two-month trip to the Kerguelen Islands? You guessed it! Nobody, and I say nobody in the team, will be able to edit that file until said colleague returns. Well, maybe the sysadmin could unlock it with some `sudo` magic, but what if this person is also in the Kerguelen Islands?

Ah, the 1990s. Well, should the scenario above happen, I guess we would have watched yet another episode of Friends, and shrugged it all off. After all, the Agile Manifesto had not been written yet, so there was no need to feel bad if we weren't releasing software at every heartbeat.

And think about the benefits: no more conflicts! You would never have to drill down and fix those lines starting with `<<<<<<<` in your source code ever again! A beautiful side effect.

The thing that we did lose forever with distributed SCMs was, thankfully, database corruptions and its malevolent twin, the complete project history loss. Yes, when you had a centralized server, particularly like in the case of that miserable failure of a system that was Microsoft Visual SourceSafe, you ran the risk of losing your entire project history because of the lack of atomic transactions. And, unlike Git, you did not *clone* projects, but you merely checked out the last version; which means that the whole history of the project was at risk at every single commit.

Visual SourceSafe was so prone to these failures that systems like Rational ClearCase, Subversion, and SourceGear Vault based their marketing around the fact that their commits, unlike their Microsoft brethren, were actually transactionally safe. How about that.

The world of SCMs was such a uncontrolled mess of conflicting options and crappy software, that Joel Spolsky explicitly stated that using it was the number one step for better code. Most teams just didn't use an SCM system at all. At my first job, we didn't use one. What a time to be alive.