bitquabit

Unorthodocs: Abandon your DVCS and Return to Sanity

February 27, 2015
11:00 am

Programming

Hi. My name is Benjamin, and I'm a DVCS apologist.

I've pretty much *always* been a DVCS apologist. I know quite a few people who've been using DVCSes since Mercurial and Git, and a few who go back to **BitKeeper**, but I can totally out-hipster you. I was there for **Monotone**. I actually remember struggling to grok **tla**, and being happy that someone took the time to write **baz**. I remember the promise and the failure that was **Darcs**. I remember thinking that even *Darcs* was comically primitive, because it was nothing but a poor imitation of Smalltalk DVCSes like **Monticello**, themselves mere iterative improvements on classic Smalltalk changesets.[1]

You merely adopted the DVCS. I was born into it, molded by it. I didn't see CVS until I was already a man[2]; by then it was nothing to me but a cause for self-inflicted blunt force trauma to the head.[3]

To me, the arrival of Git and Mercurial was a godsend, not because they were radically different than what I was used to, but because it finally meant that I had a sufficiently advanced general-purpose DVCS that I *at last* could ~~cleanse the ground with the blood of my enemies~~ obliterate the use of Subversion, CVS, that thingie that Microsoft made before TFS[4], and anything else that might cross my path, and replace them with Mercurial (or, in a truly desperate situation, Git) goodness. Hell, **I built a whole product** centralized around making DVCSes easy and simple to use, and then **went on the lecture circuit** explaining how to best use them in your workflow.

These aren't the droids you're looking for

Somewhere along the way, I think I lost sight of the forest for the trees. I was so actively trying to argue why DVCSes were so superior to the existing centralized source control systems that we had that I never really stopped that long to think about if maybe, just *maybe*, they in fact *weren't* well suited to every single situation that involved source code.

And as a result of the efforts of people like me, we're now seeing some truly insane "best practices" in the name of adopting Git.[5] And mind you, we insist they're "best practices"; they're not workarounds, oh *no*, they're *what you should have been doing since the beginning*.

And that's bullshit.

Today, I'm putting my foot down. I helped start this nonsense, so I'm going to help stop it. If a DVCS is great for your workflow, fine. If the trade-offs it imposes are good for you, great. But let's stop claiming that they're free, because they have a cost, and the cost is sometimes not worth it.

Why we fell in love with DVCSes

The *actual* reason is because GitHub let coders cleanly publish code portfolios, and GitHub happened to use Git as the underlying technology to enable that, but that's a blog post for another time, so let's instead pretend that this was purely due to technical reasons.[6]

It came down to atomicity. CVS treated the atom as the file: a given *file* had a history, and a *file* had versions, but the *repository* really didn't. Oh sure, there were tags and branches, and those *did* really operate at the repository level, but those were the tools you reached for around release time—not part of your daily flow.

Subversion treated the atom as the entire repository…*almost*. I mean, Subversion *claimed* that's what it was doing, and the entire repository had one single monotonically increasing version number, but Subversion went at once too far and not far enough: it went too far by saying that *every*thing is really just a convention-over-configuration use of a directory, and that in turn meant it didn't go far *enough* because it forced Subversion to think of merging and branching at the *file* level.

Git and Mercurial (and, for that matter, most other DVCSes) fix that problem by *actually* treating the whole state of the repository as the atom. As fall-out, they treat merging and branching as *repository-level* operations. In this, they were a *massive* improvement over Subversion: by treating the whole repo state as the atom *for real*, they could do things like sanely track renames, replay merge resolutions, realize when deleting and recreating a file was an actual conflict, and more. This suddenly meant that long-lived branches could be sane, which in turn meant that feature flags could finally die,[7] and all was good in the world, amen.

But here’s the rub

Note: we fell in love with DVCSes because they got branching right, but there’s nothing inherently *distributed* about a VCS with sane branching and merging. You could absolutely make a centralized VCS that got merging just as right as any of these others—and, indeed, long after the horses bolted, Subversion is closing the barn doors by **adding better merge tracking**, which should land literally any year now. No, the only thing that a DVCS gets you, by definition, is that everyone gets a copy of the full offline history of the entire repository to do with as you please.

Let me tell you something. Of all the time I have ever used DVCSes, over the last twenty years if we count Smalltalk changesets and twelve or so if you don’t, I have wanted to have the full history while offline a grand total of maybe about six times. And this is merely going *down* over time as bandwidth gets ever more readily available. If you work as a field tech, or on a space station, or in a submarine or something, then okay, sure, this is a huge feature for you. But for the rest of us, I am going to assert that this is not the use case you need to optimize for when choosing a VCS. And don’t even get me *started* on how many developers seem to assume that they can’t get work done if GitHub goes down. Suffice it to say that I think most developers are pretty fundamentally unaware of how to use their DVCS in a distributed manner in the first place.

That’d be fine if you got the distributed part “for free”, but you don’t. Because until **Pied Piper** makes a source control system, part of saying that you have the whole history means that you have an awful lot of data, and that causes Problems™. Let’s explore them, shall we?

Say goodbye to blobs (or your sanity (and sometimes both))

Take blobs (a.k.a. binary assets). Blobs are a part of most programs. You need images. You need audio. You need 3D meshes. You need to bundle a ton of fonts, because Android has like three of them, none of which happen to be Wingdings. These are large, opaque files that, while not code, are nevertheless an integral part of your program, and need to be versioned alongside the code if you want to have a meaningful representation of what it takes to build your program at any given point.

That’s fine for a centralized system. You only have one copy at any give point, so the amount of disk space you need is basically the size of the current (or target) version of the repository—not the whole thing.[8] We don't really need to care about how the history is stored.

With a DVCS, though, we have a problem. You *do* have the whole history at any given point, which in turn means you need to have *every* version of *every* blob. But because blobs are usually compressed already, they usually don't compress or diff well at all, and because they tend to be large, this means that your repository can bloat to really huge sizes really fast.

A sane engineer at this point would say, "Well, if you're doing that kind of thing, maybe you should use a centralized system," but we're in the DVCS cult now, so we don't do that. Oh no, not us. We instead tell you that blobs are totally different from source code, and they really should be versioned in their own store, and then we invent insanity like **git-annex** so that you can have a non-distributed separately-configured non-Git-like store alongside your source code.[9]

You know what kind of external binary asset management system works really well? Subversion. Turns out it works tolerably for source control, too. Maybe you should take a look at it.

Say goodbye to sane version synchronization

It's not just blobs that DVCSes don't handle well. It's also repositories with really big histories and/or tons of files. That's both because the repository can again get extremely large—Emacs, a single application with no bundled dependencies, clocks in with a 313 MB repository, for example, and Linux itself happily runs along at about 1 GB—but also because the structures traditionally used by Git simply don't scale well to very large repositories. In Git, for example, directories (which Git, in what I assume is a nod to ecoterrorists, calls trees) are identified by their SHAs, which are in turn determined by their contents, which in turn is defined by the SHAs of any trees they contain, recursively. This means that changing a file in a deep directory will require generating new trees for every directory up the chain—and, of course, that figuring out *what* changed in a given directory requires loading up all the trees going *down* the chain. Wondered why `git blame` runs slow as hell on big repos? Now you know.

But this isn't a problem to us DVCS apologists. Nosiree! We instead told you to chunk up your project into a gazillion repos, and that this was *Better*™, because *it forced you to break up your code base into tiny pieces*. That's right: you should be happy that DVCSes can't scale to this size, because it made you code better.

This is stupid. This is like saying that chopping off your arms is good because it forces you to get really good at tying your shoes with your teeth.

As much as I am a big fan of the Zen saying about the sound of no hands clapping,[10] this argument is specious at best, justifying why a weakness is acceptable by claiming it's superior.

But what makes it more absurd is that not even DVCS proponents *really* believe themselves when you get down to it. Instead, they design all kinds of tools to navigate around this issue. Google wrote **repo**. Git gained **submodules**, and Mercurial gained **subrepositories**. Build systems suddenly learned how to speak Git and Mercurial due to the inevitable repository explosion that would accompany any decent-sized project. All to work around a nominal design *improvement*.

Facebook and Google know that keeping all your source code in one single repository is good, because it turns out that using an SCM to manage your source dependencies by this concept of, you know, *versioning the source code*, is kind of the whole point of using them in the first place. Now of course sometimes you *do* genuinely want separate repositories, and these align with exactly when you wanted them in Subversion and CVS, too. But let's admit that saying "small is good" is a complete misfeature. You're *giving something up* by going that route, not gaining something in the form of BDSM dependency management.

Say what again, I dare you, I double dare you

All of the above might be almost tolerable if DVCSes were easier to use than traditional SCMs, but they aren't.

You needn't look further than how many books and websites exist on Git to realize that you are looking at something deeply mucked up. Back in ye days O Subversionne, we just had **the Red Book**, and it was good, and the people were happy.[11] Sure, there were other tutorials that covered some esoteric things that you never needed to do,[12] but those were few and far between. The story with CVS was largely the same.

And then Git happened. Git is so amazingly simple to use that **APress, a single publisher, needs to have three different books on how to use it**. It's so simple that **Atlassian and GitHub both felt a need to write their own online tutorials** to try to clarify the main Git tutorial on the actual Git website. It's so transparent that developers routinely tell me that **the easiest way to learn Git is to start with its file formats and work up to the commands**. And yet, when someone dares to say that Git is harder than other SCMs, they inevitably get yelled at, in what I can only assume is a combination of Stockholm syndrome and groupthink run amok by overdosing on five-hour energy buckets.

Here’s a tip: if you say something’s hard, and everyone starts screaming at you—sometimes *literally*—that it’s easy, then it’s **really** hard. The people yelling at you are trying desperately to pretend like it was easy so *they* don’t feel like an idiot for how long it took *them* to figure things out. This in turn makes *you* feel like an idiot for taking so long to grasp the “easy” concept, so you happily pay it forward, and we come to one of the two great Emperor Has No Clothes moments in computing.[13]

Do you hear the people sing, singing the song of angry men

There is one last major argument that I hear all the time about why DVCSes are superior, which is that they somehow enable more democratic code development, which is code for “I think GitHub pull requests are the Alpha and the Omega of software development, all hail Xenu.” Indeed, to listen to this, you would believe that open-source development was impossible, or at least absolutely horrible, before GitHub sprang into melodious existence.

This is like someone with hardcore Stockholm syndrome complimenting their kidnapper’s pancakes. Prior to GitHub, to send a patch to a project, you needed to

1. Get a copy of the source code
2. Make your change
3. Generate a patch with `diff`
4. Email it to the mailing list
5. Watch it get ignored

Whereas with the GitHub pull-request model, you instead need to

1. Fork the repository on GitHub
2. Clone your fork of the source code
3. Make sure you’re on the right branch that upstream expects your patch to be based on, because they totally won’t take patches on `master` if they expect them on `dev` or vice-versa.
4. Make a new local branch for your patch
5. Go ahead and make the patch
6. Do a commit
7. Push to a new branch on your GitHub fork
8. Go to the GitHub UI and create a pull request
9. Watch it get ignored

I see this workflow done for the tiniest of tiny projects on the grounds that “it makes things easier”. Yet I watch OpenBSD, an entire freaking *operating system*, get by just fine with CVS—*CVS*—and patch bombs. Hell, Mercurial itself, which is, you know, a DVCS, does development via patchbomb emails, not pull requests.

And this extra complication doesn't really get you anything. You still frequently need to rebase patches when they don't merge cleanly, just like you used to have to tinker with `patch` fuzz factors. You still get ignored, as **250,000+ open PRs that haven't been touched in at least two months can testify**. In fact, the only actual advantage I can see is that you get your name explicitly in the commit history instead of in a `THANKS` or a `Changelog` or a `README`. I mean, good job if that's what you want, but maybe admit that it's about vanity and not about tooling.

Oh let's go back to the start

We aren't going to abandon DVCSes. And honestly, at the end of the day, I don't know if I want you to. I am, after all, still a DVCS apologist, and I still want to use DVCSes, because I happen to like them a ton. But I do think it's time all of us apologists take a step back, put down the crazy juice, and admit, if only for a moment, that we have made things horrendously more complicated to achieve ends that could have been met in many other ways.

Thankfully, this whole point may be moot soon. Facebook and Google are putting in a tremendous amount of effort to **make Mercurial scale to gargantuan code bases with ease**—which, while it largely removes the D from DVCS, also means that you will be able to use at least Mercurial in a completely sane way with the completely reasonable workflows you want to manage code. Git may get there someday, too.[14] So at the end of the day, we'll end up, a decade later, right back where we could have been at the beginning: having a centralized SCM that can scale to insane sizes, and that also has sane branching and merging.

Long live the centralized SCM.

---

1. Whether Smalltalk changesets count as a DVCS, or as patches, is a bit complicated. They're technically instructions that will be executed by the VM, and therefore they're almost scripts, but they usually contained the whole method-by-method history for how the image came to be, which meant that they could function similarly to a DVCS. ↩

2. This is actually not true. ↩

3. This is actually also not true, but very, *very* truthy. ↩

4. The storage format was that of Mordor, which we will not utter here. ↩

5. And it is Git, because Git "won", which it did because GitHub "won", because coding is and always has been a popularity contest, and we

are very concerned about either being on the most popular side, or about claiming that the only reason literally everyone doesn't agree with us is because they're stupid idiots who don't know better. For a nominally nuanced profession, we sure do see things in binary. ↩

6. It was *also* due to technical reasons, mind, but claiming that DVCSes arose purely due to technical concerns would be like claiming we have web servers running Windows because Linux servers didn't have virus scanners we could trust at the time. ↩

7. And subsequently get resurrected when people remembered that they actually serve a purpose in real life where ideological purity gets in the way of making kosher turkey bacon. ↩

8. This isn't quite true. Subversion and CVS actually both store a complete, redundant, uncompressed copy of everything in your source tree to allow you to revert without talking to the server. This is why Subversion checkouts can actually be *larger* than Mercurial and Git versions of a given repository. That said, this distinction quits mattering in repositories with reasonable numbers of blobs, for reasons we'll see in a second. ↩

9. I'm picking on Git here because it's more popular, but **Mercurial's equivalent largefiles extension** makes basically exactly the same trade-offs, albeit in my opinion with a better UI. ↩

10. Close enough. ↩

11. For being programmers, anyway. ↩

12. Like cloning a full repository for offline use. ↩

13. We'll save the other one for another day. ↩

14. Sike. ↩

Want to comment on this post? Join the discussion! ***Email my public inbox.***

Archive • **RSS** • **JSON**