danluu.com

Advantages of monorepos

10-12 minutes

Here's a conversation I keep having:

> **Someone**: Did you hear that Facebook/Google uses a giant monorepo? WTF!
> **Me**: Yeah! It's really convenient, don't you think?
> **Someone**: That's THE MOST RIDICULOUS THING I've ever heard. Don't FB and Google know what a terrible idea it is to put all your code in a single repo?
> **Me**: I think engineers at FB and Google are probably familiar with using smaller repos (doesn't Junio Hamano work at Google?), and they still prefer a single huge repo for [reasons].
> **Someone**: Oh that does sound pretty nice. I still think it's weird but I could see why someone would want that.

"[reasons]" is pretty long, so I'm writing this down in order to avoid repeating the same conversation over and over again.

Simplified organization

With multiple repos, you typically either have one project per repo, or an umbrella of related projects per repo, but that forces you to define what a "project" is for your particular team or company, and it sometimes forces you to split and merge repos for reasons that are pure overhead. For example, having to split a project because it's too big or has too much history for your VCS is not optimal.

With a monorepo, projects can be organized and grouped together in whatever way you find to be most logically consistent, and not just because your version control system forces you to organize things in a particular way. Using a single repo also reduces overhead from managing dependencies.

A side effect of the simplified organization is that it's easier to navigate projects. The monorepos I've used let you essentially navigate as if everything

is on a networked file system, re-using the idiom that's used to navigate within projects. Multi repo setups usually have two separate levels of navigation -- the filesystem idiom that's used inside projects, and then a meta-level for navigating between projects.

A side effect of that side effect is that, with monorepos, it's often the case that it's very easy to get a dev environment set up to run builds and tests. If you expect to be able to navigate between projects with the equivalent of `cd`, you also expect to be able to do `cd; make`. Since it seems weird for that to not work, it usually works, and whatever tooling effort is necessary to make it work gets done[1]. While it's technically possible to get that kind of ease in multiple repos, it's not as natural, which means that the necessary work isn't done as often.

Simplified dependencies

This probably goes without saying, but with multiple repos, you need to have some way of specifying and versioning dependencies between them. That sounds like it ought to be straightforward, but in practice, most solutions are cumbersome and involve a lot of overhead.

With a monorepo, it's easy to have one universal version number for all projects. Since atomic cross-project commits are possible (though these tend to split into many parts for practical reasons at large companies), the repository can always be in a consistent state -- at commit #X, all project builds should work. Dependencies still need to be specified in the build system, but whether that's a make Makefiles or bazel BUILD files, those can be checked into version control like everything else. And since there's just one version number, the Makefiles or BUILD files or whatever you choose don't need to specify version numbers.

Tooling

The simplification of navigation and dependencies makes it much easier to write tools. Instead of having tools that must understand relationships between repositories, as well as the nature of files within repositories, tools basically just need to be able to read files (including some file format that specifies dependencies between units within the repo).

This sounds like a trivial thing but, take this example by Christopher Van

Arsdale on how easy builds can become:

> The build system inside of Google makes it incredibly easy to build software using large modular blocks of code. You want a crawler? Add a few lines here. You need an RSS parser? Add a few more lines. A large distributed, fault tolerant datastore? Sure, add a few more lines. These are building blocks and services that are shared by many projects, and easy to integrate. ... This sort of Lego-like development process does not happen as cleanly in the open source world. ... As a result of this state of affairs (more speculation), there is a complexity barrier in open source that has not changed significantly in the last few years. This creates a gap between what is easily obtainable at a company like Google versus a[n] open sourced project.

The system that Arsdale is referring to is so convenient that, before it was open sourced, ex-Google engineers at Facebook and Twitter wrote their own versions of bazel in order to get the same benefits.

It's theoretically possible to create a build system that makes building anything, with any dependencies, simple without having a monorepo, but it's more effort, enough effort that I've never seen a system that does it seamlessly. Maven and sbt are pretty nice, in a way, but it's not uncommon to lose a lot of time tracking down and fixing version dependency issues. Systems like rbenv and virtualenv try to sidestep the problem, but they result in a proliferation of development environments. Using a monorepo where HEAD always points to a consistent and valid version removes the problem of tracking multiple repo versions entirely[2].

Build systems aren't the only thing that benefit from running on a mono repo. Just for example, static analysis can run across project boundaries without any extra work. Many other things, like cross-project integration testing and code search are also greatly simplified.

Cross-project changes

With lots of repos, making cross-repo changes is painful. It typically involves tedious manual coordination across each repo or hack-y scripts. And even if the scripts work, there's the overhead of correctly updating cross-repo version dependencies. Refactoring an API that's used across tens of active internal projects will probably a good chunk of a day. Refactoring an API that's used across thousands of active internal projects is hopeless.

With a monorepo, you just refactor the API and all of its callers in one commit. That's not always trivial, but it's much easier than it would be with lots of small repos. I've seen APIs with thousands of usages across hundreds of projects get refactored and with a monorepo setup it's so easy that it's no one even thinks twice.

Most people now consider it absurd to use a version control system like CVS, RCS, or ClearCase, where it's impossible to do a single atomic commit across multiple files, forcing people to either manually look at timestamps and commit messages or keep meta information around to determine if some particular set of cross-file changes are “really” atomic. SVN, hg, git, etc solve the problem of atomic cross-file changes; monorepos solve the same problem across projects.

This isn't just useful for large-scale API refactorings. David Turner, who worked on twitter's migration from many repos to a monorepo gives this example of a small cross-cutting change and the overhead of having to do releases for those:

> I needed to update [Project A], but to do that, I needed my colleague to fix one of its dependencies, [Project B]. The colleague, in turn, needed to fix [Project C]. If I had had to wait for C to do a release, and then B, before I could fix and deploy A, I might still be waiting. But since everything's in one repo, my colleague could make his change and commit, and then I could immediately make my change.
>
> I guess I could do that if everything were linked by git versions, but my colleague would still have had to do two commits. And there's always the temptation to just pick a version and "stabilize" (meaning, stagnate). That's fine if you just have one project, but when you have a web of projects with interdependencies, it's not so good.
>
> [In the other direction,] Forcing *dependees* to update is actually another benefit of a monorepo.

It's not just that making cross-project changes is easier, tracking them is easier, too. To do the equivalent of `git bisect` across multiple repos, you must be disciplined about using another tool to track meta information, and most projects simply don't do that. Even if they do, you now have two really different tools where one would have sufficed.

Ironically, there's a sense in which this benefit decreases as the company gets larger. At Twitter, which isn't exactly small, David Turner got a lot of value out of being able to ship cross-project changes. But at a Google-sized company, large commits can be large enough that it makes sense to split them into many smaller commits for a variety of reasons, which necessitates tooling that can effectively split up large conceptually atomic changes into many non-atomic commits.

Mercurial and git are awesome; it's true

The most common response I've gotten to these points is that switching to either git or hg from either CVS or SVN is a huge productivity win. That's true. But a lot of that is because git and hg are superior in multiple respects (e.g., better merging), not because having small repos is better per se.

In fact, Twitter has been patching git and Facebook has been patching Mercurial in order to support giant monorepos.

Downsides

Of course, there are downsides to using a monorepo. I'm not going to discuss them because the downsides are already widely discussed. Monorepos aren't strictly superior to manyrepos. They're not strictly worse, either. My point isn't that you should definitely switch to a monorepo; it's merely that using a monorepo isn't totally unreasonable, that folks at places like Google, Facebook, Twitter, Digital Ocean, and Etsy might have good reasons for preferring a monorepo over hundreds or thousands or tens of thousands of smaller repos.

Other discussion

Gregory Szorc. Facebook. Benjamin Pollack (one of the co-creators of Kiln). Benjamin Eberlei. Simon Stewart. Digital Ocean. Google. Twitter. thedufer. Paul Hammant.

Thanks to Kamal Marhubi, David Turner, Leah Hanson, Mindy Preston, Chris Ball, Daniel Espeset, Joe Wilder, Nicolas Grilly, Giovanni Gherdovich, Paul Hammant, Juho Snellman, and Simon Thulbourn for comments/corrections/discussion.