Systems Notebook
Socially Distant Projects

CORY LUENINGHOENER

Cory Lueninghoener makes big scientific computers do big scientific things, mainly looking at automation, scalability, and large-scale system design. If you don't see him hanging out with the LISA and SREcon crowd, he's probably out exploring the mountains of northern New Mexico.
cluening@gmail.com

I don't know about the rest of you, but for me the last several months have been really weird. At the start of March, my daily routine stopped being one that involved getting up, riding my bike to my office, talking to my coworkers, and hopefully getting some technical work done. Instead, I started walking into my garage every morning, sitting at my workbench-become-desk, and interacting with all of my coworkers via WebEx, Skype, BlueJeans, Zoom, and just about any other online meeting package that's ever been invented. While being socially distant has resulted in fewer interruptions, and I feel like I have gotten a lot more done each day, it's also made it clear that projects frequently require socialization to make progress. It turns out that most technical projects benefit from some level of social closeness.

Getting Stuff Done, Together

Let's take a look at a project I have been recently working on, one that started back in the days when we could sit closer than two meters apart from each other. This project, which is still ongoing, is a long-term effort to replace the aging software stack we use to manage many of our scientific computing clusters with something more modern. To say the system management stack that we started with was outdated would be an understatement: one of the main tools we have been using for a long time last had a public release in 2012, and the domain name of the company that was founded to support it was recently for sale—$2999 (CHEAP) and it could be yours! But the stack, which also included Cfengine 2 and SVN, was solid and well known on our production teams, so despite its age making it a liability, there was reluctance to change.

Anybody who has worked in computing long enough has faced the same decision we had to make around a year ago: do we continue dragging our current software stack forward, hoping that it can continue to serve us for a few more generations of systems? Or do we start the long process of updating, knowing that we will face unexpected challenges and potentially introduce instability during the process? While we have faced this question in the past and have always decided to wait a little longer, this time we decided to attack it head on.

Now, to be honest, our environment isn't *that* complex, and this column isn't going to be about the technical details of our solution. I will mention that it involves Git, Ansible, and a yet-to-be-determined provisioning tool, but the work we are doing with them is pretty standard. Standard enough that a motivated team of three or four people could probably have replaced most of the aged components with about six months of solid work. But if those four people had hidden away in their offices for those six months, only eating cheese and pizza that we slid under their doors for them, and they emerged at the end of their metamorphosis with a beautiful new software stack that was perfect in every way, the project would have been a total failure. The problem we had was partly a technical one, but also a social one. We needed to move an entire organization of technical people from one software stack to another, and we needed to do it in a way that respected the fact that some teams wanted to be involved in the development, but a lot of the teams just wanted to be the end users of a stable product.

Cha-cha-cha-cha-changes!

How do you introduce a big change to a big organization? It involves transparency, iteration, and building trust. It involves being social. This starts out all the way at the start of the project, when you need to sell the idea to your immediate coworkers, and continues through selling that same idea to members of other teams, managers, program managers, and everybody else who might be affected by the change. It involves sharing your code, whether that is actual code in Go, Python, or some other language, or it is a set of YAML configuration files. And it involves two-way conversations: presenting your ideas and your code for review, and accepting feedback that others give in return.

We used that recipe to great effect with this project, and we started out small in the beginning. Our initial social circle was just a few of us who had been thinking about the project for a long time, and we started by merging our ideas into an initial project plan. But instead of acting on the plan, creating a new system management stack, and then trying to get others on board, we started out by talking about our plan with our managers and fellow tech leads to make sure we wouldn't create something that would be dead on arrival. Meanwhile, we started a proof of concept where we could try out ideas and incorporate feedback from our colleagues, developing it in an open way that built understanding and trust. Once we knew we had the backing of a sufficient number of stakeholders, we built a small development team with motivated members from each of the teams that needed input, and we started working on the real project.

If you just read that and thought, "Wow, that must have taken a while," then you are totally correct. But by doing a lot of the socialization work up front, we were saving time along the way and preventing failure at the end. As we started the technical work on the project, we knew we needed to find ways to keep the project collaborative. Since the development team was made up of representatives from a variety of other teams, we needed to build ourselves up as a meta-team that could work on this together. How did we do that?

Let's Get Together

To start, we had meetings. No, really! A well-managed meeting is a *very* effective way to share information with multiple people at once. While we were still able to meet in person, we met once a week as a team. Around once a month, we used those meetings as "broadcast" meetings—making announcements, working through administrative details, and generally keeping everybody on the team up-to-date. The rest of the meetings were used for social coding activities: group code reviews, giving presentations on recent work, and triaging development tasks. Two important aspects that made these meetings successful were having agendas and finishing on time. Both of these aspects are based on the same idea: respect others' time. By ensuring we had agendas (and that we stuck to them!), we made it easier for team members to prioritize their time and be ready for the topics that would be discussed that day. By finishing on time, we kept our discussions bounded and didn't steal time from other work.

This model hit a snag in the middle of March, when the world changed and we all started working remotely. No longer were we able to follow our normal routine of getting together weekly to talk about the details of an Ansible deployment. Since our meetings were designed around in-person interaction, we decided to cancel them until things got better. However, as of late May, we recognized that we would likely be working under social distancing restrictions for a longer term than initially anticipated, and as I am writing this (June 2020) we are starting to spin the project back up. Luckily, we had another way to work collaboratively at a distance waiting in the wings.

Enter GitLab

Very early in the process, we had started hosting our work on a local GitLab instance. While our existing system management stack was backed by SVN and we used a separate issue tracking system for our day-to-day work, we recognized early on that adopting an integrated repository browser, issue system, and code review system would provide a new level of insight into our initial coding project as well as the changes that were happening in our systems.

Git has spawned a variety of collaboration tools, from full-featured services like GitHub to locally hosted tools like Gitea. In between is GitLab, which can be used as a remote service or hosted locally. All of these tools promote working on projects in the open, and all of them follow the same general concept of a "merge request" workflow: to make a change, you create a branch, make your changes, push the branch up for review, and then merge the results into the master code branch. These tools provide tight integration with an internal issue tracking system and a web-based front end, providing a great deal of transparency into a team's development process. In our case, GitLab most closely met our needs, and we enthusiastically embraced its use.

As we have started spinning this project back up, we have begun using GitLab's integrated features in earnest. Our weekly in-person meetings have moved online, and we now use GitLab as the main driver of our meetings. Whereas we had previously used a separate meeting agenda to decide on discussion topics, we now use our existing tasks and issues to drive the meetings. While we have replaced the meeting room projector with a shared WebEx screen, more people tend to interact with GitLab on their laptops during the meetings than before. The tooling has stayed the same, but the way we use it to interact with our code and with each other has changed to meet our new needs.

But Wait, There's More!

One final note about the benefits of making a project that is strong both technically and socially: an unexpected outcome of this effort was finding other teams that were starting down the same path on similar projects at about the same time we were doing this. We had originally set out to build a repository that could manage scientific computing clusters, but as we socialized our plans, the core team working on this project started picking up members of other teams who wanted to build on our work. We took this into stride as a group, and used the opportunity to make our work more flexible and accessible to more teams.

To do this, we split our Ansible repository into two parts. Each individual team has their own Ansible *inventory* directory, which contains their team-specific host definitions, variable definitions, and playbooks. Meanwhile, all teams share an Ansible *roles* directory, which contains reusable building blocks that install and configure things like NTP, `rsyslog`, and authentication in a standard way across our environment. Had we done this project in isolation, none of us would have recognized the utility in splitting the repository out like this until it was much too late to implement it. And by using GitLab as a central collaboration point, we have a very social roles repository that multiple teams can edit and review, but also the flexibility for each team to build their own team-specific work on top of that.

And the Beat Goes on

So where are we currently with this project? As I mentioned at the start, this is an ongoing project, and we are only partway through its implementation. I'm happy we started the project out socially, as it has benefited from that, especially when we had to start doing it remotely. We've begun to start the project up again after we paused it for a while, and as I am writing this, we're just beginning to see how the project will work using text chat, WebEx sessions, and GitLab's integrated tooling. So far, it is very promising. It was a large and sudden change to our workflow, and I don't think it would have worked out as well had we not started out with a social and collaborative approach to this project.

Being socially close despite being physically distant is important beyond this time of isolating ourselves for the sake of society. Most of the USENIX community spends some amount of time working remotely with colleagues, whether they are employees of the same company, salespeople who live in different cities, contributors to open source projects, or any number of other people we benefit from working with without sharing physical space. And as we start migrating back to our normal office life, keeping projects social will help keep them running smoothly, especially when they involve large changes that we need to convince lots of people to make.