by daniel chudnov

I COULD NEVER GET MY WORK DONE WITHOUT GOOD VERSION CONTROL, AND I'LL BET YOU'LL FEEL THE SAME WAY.

Better, Faster, Stronger: Version Control for Everybody

Last month I wrote about Lucene, the widely used general-purpose information retrieval toolkit, and Solr, the Lucene-based search application that makes using Lucene easier than ever before. Sometimes you get lucky like this—when the tools you depend upon get better and better over time. Problems that were hard yesterday become more manageable tomorrow, and the most time-consuming technical tasks shrink down to a few lines of configuration. This month I'd like to introduce you to another area of software development where an important set of previously difficult and time-consuming tools have recently become vastly easier for more people to use: version control software.

Sounds exciting, huh? Actually, it's incredibly exciting to me. There might be no more important tool in the programmer's toolbox than a reliable version control system. I'll tell you here and now that if you do anything technical, ever, whether it's writing code or even just editing webpages or maintaining a configuration script or two, you should be using version control too. Have you ever copied "index.html" to "index.html-old"? Have you ever copied "index.html" to "index.html-old3"? You have, haven't you? I know you have because I have too. This is exactly the kind of easy tweak we do every day because we think we don't have enough time to "do it right." And we all know that this is wrong because we've all been burned by this kind of tweak later on, after the fact, when the differences between "old2" and "old3" are long forgotten and something important gets lost, written over, or deleted forever.

The right way to do this is with proper version control software. Fortunately for all of us, "proper version control software" is neither as difficult nor as boring as it sounds.

Why Use Version Control?

There are many reasons why you should use version control. One is that we all make mistakes. Sometimes you need to make a change, or you make a change that you want to "undo," but your software won't undo what you did 3 weeks ago. Track Changes may work for the occasional Word document, but it does not work for your HTML pages or your configuration files; and it certainly doesn't work for your code. Most importantly, version control is easy, and if you want to be professional in how you do your work, you need to manage your work professionally. Professionals everywhere use version control software, and you can too.

Here's a simple scenario that describes how most people work with version control most of the time. Imagine that you have an hour to update two webpages to 1) change some text, 2) change two colors, and 3) update an image, all before a meeting with your boss, who wants to review the changes. Here's what you do:

- Run an "update" to make sure you have the latest version.

- Tweak the text. Review it to be sure it's right, and when it is, "commit your changes" (i.e., save a new version).

- Twiddle the colors. You may realize that you forgot which colors were there before, so double-check what they were in the earlier revision. Make your choice and commit those changes too.

- Update that image. Review it and commit it.

- Meet with your boss and show him or her your updates.

We all know what happens next. Your boss says that the new text is good, but they want to bring back one sentence that you removed and put it before the new text; one color is good but the other color should go back to the way it was; and they don't like the new image—bring back the old one, but keep the new one around just in case (whatever that means).

At this point, if you don't use version control, you may begin to panic because you might have deleted the old text, the old colors, and the old image permanently. Or even if you didn't, you might have renamed them confusingly, and you might not be able to untangle the "old2" from the "old3." Maybe you've been burned before—you know to make a backup copy of the whole directory named with a date and time before making any changes (in which case you can just reach back into your latest backup copy for what you need). But what about that one time you mistype the date or forget to copy the directory recursively?

If you use version control, just one or two commands can pull back that old sentence, that one old color, and that other image. One more "commit" saves the new changes just the way your boss wants it this time. You're done in 2 minutes. If your boss' boss swoops in and demands more revisions, you're ready for anything.

Sounds good, right? It is. But it wasn't always this good.

A Brief History of Version Control

There are several well-known and widely used commercial version control systems on the market. One is called Perforce; another is called Visual Source Safe. Both are so widely used in so many ways that they're actually called "software configuration management" systems, and version control is just one of the many things they do. They're widely used because they work, and lots of big places need all those configuration management features. If you're in one of those big places, you'll know what I mean. I'm guessing that most of you are like me, though, which is to say that you're not in one of those big places. You just want something that helps you get your work done, track changes you've made over time, and keep your boss and your boss' boss happy.

There have been free software packages for text-oriented (i.e., the plain text files we use on servers) version control for decades. In the 1990s, a newer one called CVS became popular because it added features on top of one of these tools that helped people work with the same version control repository across the internet. This better matched how people wanted to work on software once email and the web were widely available. CVS was easier than the tools it was built on. But it was still fairly difficult to configure and maintain; some common tasks like moving or renaming directories and deleting files were rather awkward. It was very widely used, so much that enough people wanted to fix those awkward bits. In 2000, a project called Subversion was started to "replace" CVS. Since then, Subversion has been the de facto choice for most people like me, working alone or on small teams within small organizations (such as libraries) or with peers across the network. Subversion is easier to install and easier to use than CVS, and it makes deletions and directory changes a breeze. As of a year ago, most people I interacted with were using Subversion most of the time, with a few CVS projects still out there. This is still true today, but there's a new set of next-generation version control tools, and many of us are learning them.

PROFESSIONALS EVERYWHERE USE VERSION CONTROL SOFTWARE, AND YOU CAN TOO.

The new thing in version control is "distributed" version control. To understand what distributed means in this case, it helps to know that both CVS and Subversion require having a centrally configured and maintained repository. The benefit of a central repository is that you can focus your maintenance costs on that repository server, with backups, mirrors, or your one reliable system administrator (or whatever works in your environment). The main drawback of a central repository, though, is that you have to configure the server every time you want to add a new developer with "commit" privileges. There's also the laptop problem—every commit has to happen when you're online because it has to write to that central server. In other words, if you're working with Subversion, you have to be able to commit before you can publish your changes to the server. And every time you do commit, you also, by definition, publish your changes to the server.

This sounds logical and good at first, but it raises important issues in

»

real practice. Sometimes you want to make a bunch of changes locally and commit little chunks of those changes for your own recordkeeping, all before you push your changes up to a server. Often this may happen just because you're not on the network, you're on a laptop on a plane or on a train, or your network's just down. Sometimes you might be online, but you might also just want to be able to commit changes to somebody else's code without either of you having to deal with the hassle of securing access privileges. If you're new to version control, these might sound like obscure cases. But believe me, they both happen more than you might imagine, and they both really complicate things.

Why 'Distributed'?

The first innovation of distributed version control is that it separates committing changes from publishing changes. When you get a copy of the latest revision from a Subversion repository, you just get that latest revision and information about reconnecting to the server to get updates or to commit changes. When you get a copy of a distributed version control repository, you get the whole repository—the latest revision and every version that came before it. And you also get the right to commit your own changes back into your copy of the repository anytime you want. You can receive or send changes back to the original source only when you explicitly ask for it. This too might sound like a strange circumstance, but this single difference makes version control much easier and much more powerful. It solves the "who gets to commit" problem because, suddenly, everybody can commit—just not to the original source repository. It solves the "disconnected laptop" problem because you don't have to be connected to commit since usually you're going to commit locally a lot more often than you'd want to commit back to a server.

I can't overstate how important and useful this is. It means you can always keep working and commit whenever you want to (including before, during, and after those meetings with the boss). It means you can publish your changes whenever you want to. It means you can give collaborators access to your published code through a simple URL; that access gives them all these same benefits too. Most importantly, it gives everybody with a copy of the repository a copy of the whole history of the repository. As a librarian interested in preserving digital artifacts, this appeals to me. More practically, though, it's great to have in case a server goes away, the network goes down, a company goes bust, or somebody just takes things in a direction you don't like. You can always continue from right where you are, with the ability to review all the changes ever made at any time and to republish those as you desire (in accordance with any licensing restrictions).

Getting Started Where You Are

If you've read this far and can see the many potential benefits of today's distributed version control tools, you're ready to go. One trick, though, is that, unlike before (when lots of people just used CVS or Subversion), there are now many tools to choose from. Some people wanted different features of distributed version control, or they wanted to implement them in different ways. So a few years ago, several different projects were started. Fortunately, a handful of these have matured into stable, reliable tools and are now widely used. The ones to look for are Bazaar (also known as bzr), Mercurial (hg), Git, and SVK. (Each one is easily found online with a search like "git version control.") It's hard to pick just one—I've tried them all, and they each have benefits. If you're already a happy Subversion user, SVK is a lot like Subversion but with distributed features added, so try SVK first. Git is probably the most widely used since it was built for developing the Linux kernel. Git is also the new tool of choice for the Koha project (follow the link to "Git Version Control" on the Koha Developer wiki at http://wiki.koha.org to learn more). Mercurial is also used for big, well-known software industry projects. For now, though, I've settled on Bazaar because one of its main goals is ease of use, and it's definitely easy to use.

If you are a bit confused about all of this, a great document to review is an overview of the various possible workflows from the Bazaar project (http://bazaar-vcs.org/Workflows). It shows how well these tools scale down to just one or two users and how they can also scale up to many more. If you read just one document about distributed version control (well, after this one), read the one from the Bazaar project, and share it with your colleagues. It talks about Bazaar in particular, but the workflows can be set up and managed with any of these tools. Bazaar also has an easy tutorial (http://doc.bazaar-vcs.org/latest/en/mini-tutorial), which makes it a good first tool to try. But the other ones have similar documentation, and I can recommend them all.

More than anything, though, I can recommend that now is the time for you to start using version control. If you already do, now is a good time to revisit your options. I could never get my work done without good version control, and I'll bet you'll feel the same way. ■

Daniel Chudnov is a librarian working as an information technology specialist in the Office of Strategic Initiatives at the Library of Congress and a frequent speaker, writer, and consultant in the area of software and service innovation in libraries. Previously, he worked on the Dspace project at MIT Libraries and the jake metadata service at the Yale Medical Library. His email address is daniel.chudnov @gmail.com, and his blog is at http://onebiglibrary.net.