by daniel chudnov
Information Technology Specialist
Office of Strategic Initiatives
Library of Congress

How to Document Your Software

MY GOAL HERE IS A BASIC TEMPLATE TO USE FOR DOCUMENTING SOFTWARE WHEN YOU WANT TO PACKAGE IT FOR SHARING.

Last night I tried to install some software, but it didn't work. The process was painful because the software's installation documentation left a lot to be desired. I wasn't sure which order I should use to perform certain steps. Some of the steps assumed I knew things I could only guess at. Many steps were implied, instead of being explicitly spelled out in detail. The documentation provided never made it clear how to know when the installation was complete and working properly. The result was failure. I still haven't gotten it working at all.

Does this sound familiar?

Let's talk about how to write good documentation. My goal here is a basic template to use for documenting software when you want to package it for sharing. If you're more of the consumer-of-software type, you can use the following descriptions as a baseline of what to look for when you're installing somebody else's software. If a package you download to install and review has the following things, you might have a good chance of getting it running; if it doesn't, you should be wary and prepare yourself for an adventure.

If you're not the software-sharing or software-installing type, you can use this document as a guide to what you should expect your peers to be able to compose or consume. Preparing documentation like this isn't a trivial task—it's a critical step in sharing. Creating and maintaining installation documentation is time-consuming; it requires precision and clarity. It should be budgeted for and scheduled as a major part of the process from the outset of any software project.

The Basics—What to Aim For

Software should, at a minimum, come with three text files (each preferably a plain text file, because that's the easiest thing to review and maintain). Each file should explain one thing:

1. README: What is this software and what's it for?

2. LICENSE: What are your terms if I want to use, copy, modify, or redistribute it?

3. INSTALL: How exactly do I install it?

The README file can be a simple line or two explaining what your software does, if it's simple software. For example, "A command-line application that converts files from format A into format B." "A web application written using language Foo and web framework Bar that allows users to edit metadata in format Baz." If your software is more complex than that, tell us about it. What are the key pieces? How do they fit together? Is there some background reading potential users of your software should do so they can be prepared to use it properly? Who are you and how should people get in touch with you with questions and comments? Keep it concise, but these topics are all fair game because anybody would be glad to see this kind of information in a README file.

The LICENSE file should explain who wrote the software and what rights are given to users.

Even if it's a free/libre/open source software package or it's proprietary, if it's your software, you should disclose your authorship and determine and communicate the rights you wish to assign to other users before you distribute it.

The INSTALL file can take various forms. If it's a standard command-line or server application that builds with a makefile, the standard, generic installation instructions for "configure; make; make install" might be perfect. (In case you don't know what that means, a lot of system-level software packages are built using common configuration and compilation techniques. They can all be configured and compiled with the exact same instructions, which is something we could all aspire to!) If it's a one-line script, you can demonstrate common usage patterns with the options and arguments it accepts. If it's a complicated application with multiple pieces that must be configured separately and connected correctly, though, you have more work ahead of you. I'll focus on this case from here on out.

Installation Details—What to Do, Precisely

Let's assume we have a web application we want to share with other people. It has a web front end and a database back end, it requires having a few support libraries installed, and it uses a popular web framework. The INSTALL document should start with a statement just like that: "This is a web-based bibliographic metadata management application written in Ruby using the Rails 3 web framework. It uses a MySQL relational database back end and the 'marc' ruby library for processing MARC records."

There. In that one line, anybody who needs to install it knows exactly what they're getting into.

Next, we pick a common deployment operation system and state exactly which version of that OS the rest of the instructions are relevant for. For example: "These instructions assume that you're running Windows XP with Service Pack 6 and the Apache 2.2 web server." Or "These instructions are for installing on Ubuntu 10.04 LTS." If users don't have this exact setup, they know that 1) they should get a setup like it before they install, or 2) they should expect that the details to follow will need some amount of tweaking on their end. It's fair to pick a single platform like this for docs—it lowers the maintenance burden, and choosing something common lets anybody looking closely at your software try it out on an environment just like yours.

Next, we explain what needs to happen to prepare the system, starting with dependencies, with exact version numbers known to work. It's enough to say something like this: "Install these dependencies: ruby-1.87, rails-3.0.3, MySQL-5.0.67, MySQL/ruby-2.8.2, and apache-2.2.17." Again, we're explaining exactly the versions of these packages that are known to work together. It's prose, so we're assuming that somebody will know how to install these packages, and we're not describing where to install them. That's okay, but we can do better. If we're providing Ubuntu install instructions, say, give a specific command:

```
% sudo apt-get install ruby rails
mysql-server-5.0  libmysql-ruby
apache2
```

That's even better because we give people something they can copy and paste, and what they're pasting is a common, well-known, preferred binary software installation technique that everybody who knows Ubuntu will recognize.

From there, we build up the pieces the app needs in layers matching the dependencies above, using command suggestions that are as precise as possible. We could say "create a mysql database for your app," but then everybody will have to remember how to do that. Not that that's hard, but if we spell it out precisely, we can save everybody time and give them a recognizable process. It can help, too, to explain how much we assume people will do for themselves. Here's an example:

"Now create the database and a user. This assumes that you've already secured the root user with a password.

```
% mysql -u root -p [enter your root password]
mysql> CREATE DATABASE myapp;
mysql> CREATE USER 'joe'@'localhost' IDENTIFIED BY 'CHOOSEPASSWORD';
mysql> GRANT ALL PRIVILEGES ON 'myapp'@'localhost' TO 'joe'@'localhost';"
```

Again, this gives even experienced sysadmins precise details on what to do—and rather than be bored or annoyed, any wise hacker will be grateful for the attention to detail and reassured that the instructions are likely to work.

Next, we give instructions with exact details for getting the app itself. If the best way is to download it as a single zipped file, we give the exact URL to that file. This might seem redundant. If the INSTALL file is with the software, you already have the software, right? Well, remember that just like everything on your new computer, even a successfully running installation will grow old over time. If it ever needs to be revisited and changed or moved, anyone assigned that task will be glad to have that exact URL readily at hand, because it might not be clear where the software even came from originally. Think of yourself 2 years from now. Will you remember the URL you got this package from? Chances are you won't.

If the best way to get the software is to check it out from version control and then to build it with some custom commands, give each of those exact commands, precisely, in order. If it's in SVN or Git, we give a sample command to check out or clone the exact URL. If tests should be run to make sure the dependencies are working, we provide the command for how to run the tests. This makes them able to be copied and pasted. If we have one or more configuration files users will need to customize, we provide a clearly named sample configuration file and specify which fields need to be customized with values appropriate to a local installation. If there are any other system- or application-level customizations or tweaks somebody needs to perform, we explain them and provide exact commands or code or text that can be used to make those changes. We do the same for any other optional customizations, making it clear that those are optional.

Finally, after stepping the user through all of these pieces, we give an exact set of steps they can follow to turn everything on and make sure that the app is running as expected. For our example, this would include pointing Apache to the app where it's installed, starting MySQL and Apache, visiting the site in a web browser, logging in, and performing a common function on the site. If users get this far, they know it's working.

I don't claim to be expert at writing these kinds of docs, but if you're looking for an example, you can review what I've written for the unalog application I maintain (http://tinyurl.com/unalogreadme). In this case, I merged the README and INSTALL docs, but I'll probably split them out sometime.

Testing the INSTALL Doc

How do you know if your INSTALL instructions are good enough? Test them yourself. Get a clean machine (a virtual one is just as good) and follow your instructions line by line. Aha: You left something out, didn't you? And those URLs, they're handy to have right there, aren't they? And the commands you can cut and paste? Pretty handy. If you can start with a clean, unmodified computer and go all the way to the point where your app is running as expected using just the INSTALL file, you're in a good place. Next, give it to a friend or two and ask them to try it too.

The Benefits

This approach appeals to me for many reasons. First, it works for me to explain my own software to myself. Whenever I want to bring up a new test instance or development environment, I follow my own instructions. I want them to be right, and I don't want to have to sleuth missing details. So whenever I find a discrepancy, I try to fix it quickly and test it again to be sure it's right. Second, this approach gives you and whoever else might try running your code a solid foundation for discussing whatever might go wrong. If they're using the same setup you described, you can step through it together and figure problems out. If they're using something different, you can zero in on the differences. Finally, when you go through the process of preparing full installation documentation, you realize where the rough edges of your app are. If your app needs a dependency library built by hand instead of the one that Ubuntu ships by default, well, maybe there's something you can change in your code so you can rely on the default Ubuntu version. That kind of change shrinks the footprint of your application and makes the foundation of commodity-packaged software underneath your application stronger. You'll end up with a tighter application that's easier to maintain and easier for others to use. And we'll all end up with fewer frustrating experiences like mine. ■

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.