

Creating Debian packages from CPAN

by Jeremiah Foster

jeremiah@jeremiahfoster.com



The Perl programming language works on a huge number of operating systems and computer hardware. A great deal of work has gone into Perl to make it cross-platform and its behavior consistent on each platform. Debian, a free, community-developed operating system, uses Perl for a variety of tools including essential parts of Debian's base system. Debian has a great deal of Perl as its DNA; perhaps that's one reason Debian calls itself the "universal operating system" (another reason is that it runs on a wide variety of hardware).

Debian also contains a large amount of the Comprehensive Perl Archive Network (CPAN) in the form of *debs*, which is Debian's way of integrating software into the operating system. In this article I provide an overview of the Debian packaging system and how a CPAN module becomes a Debian package.

■ CPAN is upstream from Debian -----

Uploading my module to CPAN is just the beginning of its journey to the machines of users. There is a lot more to do before someone can run my code. CPAN, of course, has tools that can install any well-formed Perl module onto a machine that can run `cpan` or `cpanplus` or has tools to handle tarballs. There are limitations to what CPAN installation tools can do however, namely, software dependency resolution or installing specific software that resides outside of CPAN.

Perl has wrapped so many C libraries and other tools that often when I install a module from CPAN, I need to install a C library with it. CPAN handles Perl software dependencies, but it is a huge job tracking down every software dependency and library for every operating system that perl can run on. Enter Debian's package management system, which can do just that for Debian.

Debian's package management system is a set of tools that work together to build, install, remove, or update packages on Debian or a Debian-based operating system. The set of tools is extensive, from `dselect` and `dpkg` to `apt` and `aptitude`, with many of these tools written in Perl, or prototyped in Perl then re-written in C or C++. The most familiar tools are probably `apt-get` and `aptitude`, which people can use to install and remove software on their systems. APT stands for "Advanced Package Tool" and is the tool that does automatic dependency resolution. It finds whatever library a Perl module depends on, gets the appropriate version of that library, and installs it. This task is not trivial and requires a carefully crafted package to enable these features. It is the package itself that might be considered the atomic unit in Debian because it holds the keys

to how it gets installed and what additional software it needs to function properly, or what software it conflicts with.

So how does one build a Debian package? Well, "there's more than one way to do it"; in Debian there are lots of ways to build packages, and plenty of tools to help you do that. The fundamental notion to keep in mind though is that Debian aims to keep the source code of the package pristine. There should be no changes by the package maintainer to the software that he is packaging. While there are exceptions to this rule, they are only allowed when there are security or other special considerations at hand, such as adjusting the resulting binary to the requirements of the File Hierarchy standard. Instructions for how to build and install the package should be in a special directory, called *debian*, where all customization is done. To demonstrate package creation I'm going to use the module `Test::File` from CPAN.

Choosing something to package is actually quite important. I chose `Test::File` because I find it useful and have some familiarity with it—two things a package maintainer needs to generate the interest and motivation required to make updates when there are bug reports or new features. Packaging is actually considerable work over time, and a stale package is both a potential security risk and quickly forgotten.

■ The packaging process -----

We begin the packaging process with a tool called `dh-make-perl` (the `dh` stands for Debian Helper), which is a wrapper around the `cpan` tool and a surrogate for `dh-make`, the traditional Debian make tool, plus a whole lot more. I call it the same way as I would call `cpan`, giving it a module name. It then goes to CPAN for the source code of our package, since source code always has to be included in Debian. That is one of the basic features of Debian—I can get the source code for nearly any package. Should I have to fix a bug in the source code, I should also pass my patch upstream into RT, Perl's bug tracker.

Here's an example call to `dh-make-perl` with several arguments to show some of its features:

```
% dh-make-perl --cpan Test::File \  
--desc "Test file attributes with perl." \  
--arch all --version 1.25 \  
-e jeremiah@jeremiahfoster.com \  
--dh 7 --requireddeps --build
```