Tip 2: Use libraries

Ben Klemens

3 October 2011

level: You have the syntax down and want to get real work done
purpose: Use 40 years of prior scholarship to your advantage

Twenty years ago, it was evidently pretty difficult to pull down a good library of functions and make use of them in your current project. I say this because I couldn't find any C tutorials from the period that show you how to use a non-standard library to do real work. Which is why you can find C detractors who will say self-dissonant things like *C is forty years old, so you have to write every procedure from scratch in it.*

Now, it's easy. We have the GNU to thank for much of this, because free libraries now outnumber for-pay libraries, and so there are package managers and other such systems to let you pull down a library with a few mouse clicks. If you have to create windows for your program, deal with XML, encode audio streams to MP3, or manipulate DNA sequences, ask your package manager for a library before you start from zero.

Besides the politics of free/open source/libre/whatever, the GNU also has a set of tools that will prepare a library for use on any machine, by testing for every known quirk and implementing the appropriate workaround, collectively known as the `autotools`.

Writing a package to work under autotools is, um, hellish, but the user's life is much easier as a result. Also, it's a logical extension to Tip #1, because now that you know that having a makefile will simplify compilation, it's only logical that you'd use a tool to generate a makefile for you.

To do:
Let's try a sample package, shall we? The GNU Scientific Library includes a host of numeric computation routines. If you ever read somebody asking a question that starts *I'm trying to implement something from* Numeric Recipes in C. . . , the correct response is *download the GSL, because they already did it for you.*

One of the things I ♡ about using POSIX[1] is that I can give people unambiguous and quick tech support over IM. No *click here, then look for this button* stuff, just paste this onto the command line (assuming you have root privileges on your computer):

```
wget ftp://ftp.gnu.org/gnu/gsl/gsl-1.15.tar.gz #download
tar xvzf gsl-*gz    #unzip
cd gsl-1.15
```

[1] UNIX is a trademark of AT&T; POSIX (Portable Operating System Interface (the X goes uneXplained)) is a more general descriptor for things that are UNIX-like, including Linux, BSD, Mac OS X, &c.

```
./configure          #determine the quirks of your machine
make                 #compile
make install         #install to the right location---if you have permissions
```

If you get an error about a missing program, then use your package manager to obtain it and start over. [Package managers are one of those places where I can't just tell you what to type, which is one solid reason why I'm not using one here.]

If you are reading this on your laptop, then you probably have root privileges, and this will work fine. If you are at work and using a shared server, the odds are low that you have superuser rights. If you don't have superuser rights, then hold your breath for two days until Tip #3.

Now you'll need to indicate in your makefile that you will be linking programs you write to the library you just installed. The makefile from Tip #1 had a blank LDLIBS line; this is where you start filling it in.

If you have pkg-config on hand, then use it like so:

```
LDLIBS=`pkg-config --libs gsl`
```

When you add new libraries, add them to the list like so:

```
LDLIBS=`pkg-config --libs gsl sqlite3 apophenia`
```

If you're on a system without pkg-config, you'll need to explicitly specify which libraries you need:

```
LDLIBS=-lgsl -lgslcblas -lm
```

Every time you install a new library, you will always need to add at least one item to the `LDLIBS`, like the `-lgsl` part. The GSL has a quirk that it requires a BLAS (basic linear algebra system), and `-lm` is the standard math library.

Did it install? The numeric integration documentation[2] has a sample program that integrates the function specified at the top of the file. It's a good example because I get the impression that numeric integration is the sort of thing that I feel people often re-implement in C (and I already have you reading the manual—there's a lot there). Paste it into a file and use your library-improved makefile to test and install it.

[2] `http://www.gnu.org/s/gsl/manual/html_node/Numerical-integration-examples.html`