The great packaging problem–the easy part

Ben Klemens

16 May 2011

A *library* is a collection of functions and data structures. Given a set of libraries installed in one place, functions in one library can readily call functions or structures in another library. For example, a textbook recipe organizing program could use a textbook XML library to save the recipes, so the author of the recipe program would not rewrite any XML parsing routines, but just call them from the other library. In such a setup, sets of functions will be organized into libraries, for the convenience of users, who can pick those that they need.

[I'll lean on *library* for now, though many systems call them packages, Ruby calls them gems, &c.]

The problem statement: how does a function in one library find a function in another? This remains one of the great unsolved problems of modern computing. It breaks down into two problems.

- Local: given a program installed on the hard drive, how does one find and load the requisite file?
- Global: given all the computers of the world, how does one find a needed library?

Next time, I'll be writing about the global problem, and discussing why it's so broken; as a warm-up, this time I'm starting with the local problem to show you just how solved the local problem is. If you've never put thought into it, this may also help you with debugging next time an installation fails.

It is solved in the same manner on every platform I've ever known: when a library is needed, a small set of directories are exhaustively checked for libraries. The implementation is even pretty similar in all cases, wherein an environment variable, like R_LIBS, LD_LIBRARY_PATH, PERLLIB, CLASSPATH, lists those directories that should be checked, and when a new library gets called in by a running program, the system checks each directory on the path in turn. If you're using a system with some sort of global registry (which is effectively a gigantic pile of environment variables), then the path may be listed there. [Since this is a web site for *Modeling with Data*, let me mention that Appendix A has more on getting and setting paths.]

The problem of installing a new library on an unknown system becomes the problem of knowing exactly where everything is. If a file needs to be generated, what compiler is available; if there are files that need to be modified, where are they; what is the right libpath to use for the given system?

A few libpaths are handed to you, like the ones that actully have an environment variable set. Or you could have the platform self-report its environment, like

a `newlang --get_environment` command whose output the script could then use, or you could force the user to have an environment variable on hand, or you could use a local registry, or depend on the Linux Standard Base (an effort to define the right paths once and for all), or use Autoconf's voluminous hard-coded knowledge about system-specific details, or just install in /usr/local/share no matter what. Everybody has their custom, due to differing opinions about what is technically optimal and historical glitches. But once you've found the right way to query the local system for where everything is, you'll have no problem putting everything in its right place.

GNU autotools asks the dependent library author for the name of the dependent-upon library, and a sample function. It then produces a small program—basically just

```
int main(){your_function();}
```

—and then compiles it via `gcc -lyour_libname sample_program.c`. If that works, that we haven't just asserted that the library is somewhere on the libpath, but have actually tested that the library can be loaded and used, which is pretty cool. The compiler will search its own libpath, so Autotools is implicitly searching the libpath by free riding on another system that already does so.

So that's the whole local library use problem:

- Search the libpath.
- If you found a match, optionally test that it's valid, or just load it and hope it doesn't break.
- If it isn't found, then it's on to the global problem—search the planet Earth for the library, and install it on the right path.

For installing a new system, we need to preface this with an initial step where we work out what the local paths are. There are diverse solutions to that step, but just read the platform's manual and you'll be fine. Next time, I'll cover that last step of searching for and installing a library or package from somewhere else.