# The great packaging problem—the hard part

Ben Klemens

22 May 2011

Entry #043 discussed the problem of how a package, library, or whatever set of files on a given system gets installed and used. The gist is that you need to know where everything goes, via a number of mechanisms, which is invariably expressed as a set of directories to search—a path. There are a few differences due to technical details and historical glitches, but no need to rehash those.

But if the library isn't on the system at all, then it's on to the global problem of finding a package or library from out in the world and installing it in the right place, which turns out to be much more difficult. Technically, it doesn't raise any issues more difficut than the local problem, but the political problems are more complex.

I'll start with the decentralized solution, which will naturally lead in to the centralized.

In a decentralized world, when something is missing, you tell the user, and the user goes out and gets the item and installs it. We consider this to be lacking. Modern users are allowed to be lazy, and cuss at the screen and mumble 'if you know I need a package, why don't you just get it for me instead of telling me to do the work.'

You could actually write into the install script that if a library is missing, run a quick script to download and install, like

```
wget http://sourceforge.net/libwhatever;
./configure;
make;
make install.
```

Consensus seems to be that this is a bit ad hoc, and potentially fragile, because the dependency is not just on the library, but on the unknown repository that holds the library. If the library to be downloaded releases a new version, will the makefile pull the right version? If the library to be downloaded has its own dependency issues, will users have a clue as to which step in the dependency chain they have to focus on to get things running again?

Which leads us to a system that can query a centralized repository holding a registry of packages, each of which knows exactly what other packages it depends upon.

By the way, I am referring to both package managers for individual languages or coding platforms, like Perl's CPAN, R's corresponding CRAN, Ruby's gem system, &c; and package managers for entire systems, like Linux distributions such as RPM, Debian, Pacman, Portage, or any of the other several dozen on Wikipedia's list[1]. The

[1] `http://en.wikipedia.org/wiki/List_of_software_package_management_systems`

general package managers do not really care what is in the package. Each package is expected to include an install script that finds the right libpath or works whatever magic is needed to put things in the right place. Conversely, a language-specific library manager can get all the language-specific details right. R has the most stringent package management I've seen, requiring that package authors document every user-visible function, include tests, et cetera.

Say that I just wrote a module in Perl, and it calls out to a C library, so the Perl package will have to check for the library and install it if possible (which might mean installing sub-dependencies). Ideally, it would work out the right way to do this for a Debian system, a Red Hat system, OS X, and all the others. After all, the procedure in all cases is to find the right paths and programs, then put them all together in the same sequence, using a standard shell script. But what I describe here is absolutely impossible, several times over: your Perl package manager doesn't know how to install a C library, your C library probably knows how to work out the paths it needs but not how to resolve dependencies; you'll need to write a new and potentially painful new pakage manifest for every operating system variant.

Let me waste a paragraph stressing how common cross-platform writing is (and should be). Every scripting language I've ever seen has a throwaway or two in the documentation of the form *if you get into trouble, just code your procedure in C*. So at the least, it makes sense to have a mechanism to depend on C libraries. In the communist world, C libraries have a standard means of installation on all systems (the miracle of modern science that is Autotools), so let's not pretend that there are no standards to rely upon. Or what if you want to use TCL/TK for a quick window front-end, or read in and clean data via Perl and then make pretty graphs with R?

For that Perl module, you could start with the general package manager, and then the general install script will call the language-specific install script. Conversely, you could start with the language-specific package manager, and then write dummy packages for the off-language dependencies, like a Perl module with zero lines of Perl code but a full C library hidden in a subdirectory.

But at this point, things are a mess. I've actually made some efforts to use many of the above package managers (Apt, RPM, whatever R's is called, Python's distutils), and prepping the metadata for three out of four of them them were a real pain. For the system to be robust, the author has to fill out a lot of forms in just the right way, and if there's a central repository, there's the social problem of convincing a gatekeeper that this isn't just a homebrew project and that the forms were filled out correctly. I made the blithe suggestion in the last paragraph that if you need to do cross-silo installation, just write two package manager specs, but I admit that that's like suggesting that if you're into two people, then you can simplify your life by just dating them both at the same time.

Calculating dependencies and knowing the URL to pull from, querying the system to get the right paths and commands, writing a shell script that uses the gathered info to do the install—these are all closer to technical annoyances than the Great Engineering Challenges of our time. So what makes things so darn difficult?

**Politics** At this point, you may have noticed just how not-Internet this system is. It's centralized. There's typically a gatekeeper at the central repository, who may impose lots of rules. Pretend you got a letter from Steve Jobs, referring to the iTunes Music Store, the archetype of the centrally-controlled system.

> Dear author:
>
> I have written a somewhat popular platform, which you have used and for which you have written useful software. That platform includes a distribution system, which is included by default with every copy of my platform. So if your program is in my registry, then every user will have easy access to your work.
>
> When I put your program in my registry, I will check for dependencies, but *all dependencies must be in my registry*. If you need an XML parser, and don't find one in my registry, you will have to fill in that hole in my registry before I distribute your code. It is important that I maintain party cohesion, and packages that depend upon commonly-installed libraries using other platforms will break the user's impression that mine is the One True Platform.
>
> I reserve the right to impose coding standards of my choice; for example, all programs loaded onto an iPhone must do garbage collection in the manner the Apple specifies, and may not depend on any of the still-running and still-debugged libraries originally written in FORTRAN. Of course, my automated tests can't really detect software quality, just adherence to our rules, so the quality bar is really sort of vacuous. Nonetheless, I'm going to bill my repository as *Comprehensive*, so if you don't conform to my standards, you'll look like an idiot.
>
> Yours,
>
> The Author of the One True Platform

I doubt that any one language's package manager or the maintainer of any one platform fully fits my fictional stereotype (maybe not even Apple). But the incentives are there for somebody promoting a distro or a language or other platform to build a silo, and offer access only to those who agree to stay within the silo's walls.

Silos bother me, whether they are for sinister purposes or just lazy writing that built obstacles without thinking. I don't need yet another reason for somebody to tell me that their language is so cool it's absolutely impossible for them to use any other.

Centralization isn't inevitable. To pull an example from a nearby issue, revision control systems are another great unsolved problem in computing, with dozens of competing systems. In the last few years, there's been a push toward decentralized RCSes that look more like the Internet at large and less like the iTunes Music Store. Like a peer-to-peer file sharing system, each repository of the code history in a network has met at least one other (to get the code to begin with), but effort is made to not allow one to proclaim itself central and canonical. If the CIA took down Linus Torvalds and his favorite server, the code base and version history of Linux would chug along via all

the other repositories with an equal claim to centrality. Similarly, a repository system can do more or less to promote decentralized package distribution.

There are programs like alien[2] that will do an OK job of translating the metadata from an RPM-formtted package into metadata for a Debian package, or vice versa.[3] There's nothing keeping the users of R or Perl from using a general package manager and writing install hooks to run tests and documentation checks. If the R and Perl people were using the same general package management system, there'd be nothing keeping the CRAN from depending on CPAN packages or Portage source packages.

The signs of a technical problem are different from the signs of a political problem. The technical problem might involve data structures that are difficult to translate or procedures that undo each other, but we've got none of that here. The reasons why we have so many entirely incompatible systems, the reasons we're collectively in package manager purgatory and can't cross platforms easily, are political: developers of a platform want to dictate what is an acceptable extension, competitors have decided that it's more beneficial to build silos around their own systems than to build bridges, and for the minor technial glitches—¿libc-devel or dev-libc?—everybody is just waiting for somebody else to do the grunt work of making things compatible.

The quickest way to bolt on a package manager to a new programming language would be to instruct the package author to use an existing general build system, with a few language-specific hooks provided by the author if need be. But that would give up any hope of siloing authors of new packages, and some level of (mostly political, to a small extent technical) independence. The language maintainers don't get to be gatekeepers any more and can't reject packages that don't live up to their technical, æsthetic, or political ideals. Package authors and users would benefit from easier and more connected package management systems, but the users aren't the ones who design the system.

[2] `http://kitenet.net/~joey/code/alien/`

[3] As I understand it—and I would love somebody to correct me on this—alien has trouble translating dependencies. Part of this is that package names change across systems: the GSL's development package may be libgsl0-dev, libgsl-devel, dev-libgsl, or if the system weren't so pedantic, the dev package would be just a part of libgsl. Differences in naming are really the perfect example of a social problem blocking technical solutions.