this month In FreeBSD

BY DRU LAVIGNE

In this issue's Ports Report column, Frederic Culot notes that the FreeBSD Ports Collection will turn 20 on August 21, 2014. To commemorate this anniversary, I recently interviewed JORDAN HUBBARD, its creator.

DL *What prompted you to create the Ports System and what were your design goals? For example, what else, if anything, was available at that time for managing software on BSD or other UNIX variants and what features did your design want to address?*

JH I created the Ports Collection largely as a simple experiment in automation. At the time, I couldn't help but notice that all the software I tended to add to any new FreeBSD installation followed the same basic steps: Go grab the source tarball from some known location, extract it into a working directory, configure it, build it, and install the results.

This process was also generally considered "easy enough" that no one had really tried to automate the process, at least not in the sense of simply impedance-matching to whatever source location(s) and configuration/build processes the software used "natively" (vs. entirely repackaging it into some closed ecosystem), and this made me curious: Why were we being forced to remember all those various URLs for the source tarballs and manually do the same fetch/extract/configure/build steps—often many times—for each new FreeBSD install? It wasn't particularly hard, but it also seemed like one of those repetitive tasks for which computers were invented!

So anyway, that was really about it as far as the motivation was concerned. I just wanted to see if the process could be automated, and I used make(1) because it was already essentially designed to do that exact type of job-automate repetitive task. I just needed to write the make macros to allow one to create a port's makefile relatively succinctly.

I also didn't really approach the experiment with any "design process" in mind—I simply kept porting things, going after some of the more unusual/difficult to port pieces of software as a test of the overall concept, and evolved the make macros to meet the needs of the various challenges I ran into along the way. Once I got to about 200 ports or so, I concluded that the concept had been adequately proven and my curiosity entirely satisfied, so I approached Satoshi Asami, who was already contributing a lot of ports and seemed far more motivated than I to carry the ball forward. He's really the one who substantially bootstrapped the Ports Collection into what it is today. They didn't call him the first "portsmeister" for nothing!

DL *What was your long-term vision for ports? For example, did you expect this to become the de facto way of installing software on BSD, or were you expecting this to be a stage one in a longer-term plan?*

JH I'm not sure I can say that I had any long-term vision specifically for the Ports Collection, but I certainly had some longer-term goals for the notion of "software husbandry"—as I like to refer to it—as a whole. The Ports Collection and the corresponding package management tools (I also wrote the first version of those) were largely just components of those goals.

What I really wanted was for third-party software to be really easy to discover and install by end-users, with the "messy details" of actually getting it onto their systems (or off again) being abstracted away to the point where all they needed to do was fire up some package browsing tool, click on a suitable category (or search in a search box), and select the things they wanted from a menu in order to install (or deinstall) it. I also wanted the framework used to build that third-party software to be fully exposed and accessible to developers, since someone obviously needs to feed the other end of the pipeline in order to have a third-party software collection at all. I knew the collection would not grow unless it was relatively easy to add to and update. Some of those goals were fulfilled, others not, and still

others only with caveats, and that's a good segue to your next question!

DL *Knowing what you know now, is there anything you wish you had included in the design of the Ports System?*

JH Oh yes, definitely! I could probably list hundreds of things I wish I'd done differently, but for the sake of brevity, I'll just list what I consider to be the most significant ones:

1. I wish I had not used make(1). I knew the Berkeley make macro system backwards and forwards so it was very easy for me to use it at the time, but what I failed to consider was the fact that makefiles are also very difficult to manipulate programmatically, so you can't easily sweep through the entire collection and make wholesale changes when you want to rearrange/refactor things or do any kind of real analysis of the Port Collection as a whole. It also made "automation of the automation" (like "easy ports creator" tools and such) a lot harder, since makefiles don't lend themselves to introspection. As a data description format, they leave a lot to be desired.

2. The use of make(1) also broke a cardinal rule that I wasn't really aware of at the time (hey, it was the '90s—we were still idealistic). It mixed the active and the passive data together in one place, or if you prefer an English language metaphor, the "verbs" and the "nouns" describing a port and its actions were hopelessly intertwined in the same metadata. This made it effectively impossible to audit what a port was doing—or what it wished to do in advance—at build time. Since a lot of ports also need to execute at least their installation phases as root, that was a really unfortunate design choice from a security perspective. A dedicated port building machine can always do its work in a chroot or jail sandbox, but most users simply execute "make install" as root on their machines directly, and even if a port is not malicious in nature, it can still suffer from flaws in its construction that lead to unintended consequences.

3. I mixed the process of installing software with the process of building software. At the time, it seemed like the last logical step in the build process was to support the actual installation of the software (otherwise, what would be the point?), but that didn't really properly distinguish between the responsibilities of a build framework and a package-management framework. I later did some hacky impedance matching between the package tools and the Ports Collection such that either "make install" or "make package; pkg_add <resultingpackage>" would yield the same result, and leave behind the same registration information, but what I should have done in the first place was just make the output of the Ports Collection always be packages and never actually touch the host system, leaving that task entirely to the package management system.

DL *Looking to the future, what are your thoughts on software management? Do you see ports as part of that future?*

JH Well, if I had to do it all over again today, I would probably describe a port in some machine-parseable format so that large-scale modifications of the tree could be done with less pain. I would make it possible for ports to be a bit more object-oriented (with inheritance and mix-ins). I would make the build machinery always execute in some kind of sandbox, with a variety of possible targets, e.g., not necessarily the same host architecture or release version of the builder. The end-result would always be a package, and I would also only allow packages to execute some finite set of actions at install (or uninstall) time, e.g., not just let them execute arbitrary, unauditable shell commands. That would mean all of the metadata for a package would be essentially "passive" until explicitly acted upon by the package management machinery, which could then also implement various security policies about which packages (or users) were allowed to do certain things.

I would also give a lot more thought to the actual runtime environment of the software being packaged. It's not the same Internet we had in the 1990s, obviously, and we can't just arbitrarily trust third-party software anymore—coding errors, bad actors, the whole security environment has changed. This means we need to establish provenance for everything we install on our machines, and even once we've established provenance, we need to still sandbox the heck out of it such that it can only access its own data, or that data to which we carefully grant it access, and not just run wild with our own permissions, or worse, on the system. All of those requirements are going to affect how software is audited, built, signed, distributed, and installed, and all the tools that FreeBSD is using need to evolve accordingly.

DL *Any other points of interest or trivia regarding ports or software management?*

JH If software ecosystems like the Ports Collection have taught us anything, I think it's that some of our fundamental notions

about software husbandry need a serious rethink. If you look at the Ports Collection today, it's a forest of individual software dependencies and versions, many of which are mutually incompatible, and even a well-organized junkyard is still a junkyard. It just keeps getting bigger and less wieldy as time goes on, and I'm not sure how much further it can scale out before it starts to collapse under its own weight. Some might argue that such collapse is already under way.

Moreover, by making it trivially easy to link things together in arbitrary ways, we've also only encouraged the perpetuation of some of the software industry's worst practices. Linking together lots of disparate software components into a single address space to create an application, whether it's targeted at some embedded role or running in a feature-rich environment like a desktop, might be easy and rather tempting to do, particularly when you're coding to a deadline, but it's also obviously fragile and fraught with potential peril.

The recent events with OpenSSL have amply demonstrated this, as if we needed more demonstration, and as significant as the fallout from the Heartbleed vulnerability has been, I think we've really only seen the tip of that iceberg. We're going to have even wider-spread vulnerabilities, and suffer even more pain before the industry stops thinking of dynamic libraries and loadable modules as "handy software ICs" and more as scary things for which they need to use opto-isolators everywhere. Software packages like OpenSSH and Postfix have been using multiprocess, privilege-separated models for years now, but we need to think of ways to get everyone to do this as a matter of course, even if it requires the creation of fast and secure IPC mechanisms (and handier APIs that make those things easier to use) to facilitate it.

In that kind of world, it's just possible that software management frameworks like the Ports Collection could actually be part of the process of driving such best practices rather than the current worst practices. One can only hope! ●

Dru Lavigne is Director of the FreeBSD Foundation and Chair of the BSD Certification Group.