Bitrig ports: BSD ports, packages, and Uncommon Operating Systems

John C. Vernaleo, Ph.D.
jcv@bitrig.org
Bitrig

Abstract

The BSD operating systems strive to provide a complete, usable system in one coherent place. In theory this works well, but in practice, most people depend on a variety of third party software for their computing needs. This is where ports and packages come in. We will discuss some issues that come up in dealing with packaging third party software on an uncommon operating system (Bitrig) along with how this can help us to improve software portability for all the BSD operating systems (and other UNIX-like systems).

1 Introduction

The various BSD operating systems each strive to provide a complete, usable system in one coherent place with varying focuses of their own. While they mostly succeed at this goal, in practice it is rare for most people to use a UNIX-like system without a variety of third party software. Third party software is arguable the reason UNIX rose to prominence (and for better or worse that is a large part of why GNU choose to mostly follow UNIX).

One could download, extract, patch, configure, compile, and install each program one needs separately. For small programs this even works. But for larger programs (web browsers, editors, etc.) or a large number of programs, this gets very cumbersome, very quickly. It is also almost impossible to stay up to date with software when dealing with each piece individually.

The solution is to have some system of packages (or ports in OpenBSD speak) that is distributed by the operating system vendor to make third party software easy to install and manage. While not strictly part of the BSD system (OpenBSD for example makes it very clear that ports are not audited or held to the same standards as things in base [4]), many users will spend most of their time using the third party software. Third party software also ends up requiring more developers than the base system (compare commit logs between src and ports for any system and count distinct contributors).

There are a few challenges with third party software. For starters, we largely live in a Linux world (as free Unixes go) or even worse an OSX or Windows world. This means that most software was probably not built for the BSD of our choice. If everyone followed standards, this wouldn't be a problem. That's what POSIX is for you might say. Unfortunately, that does not get us very far. POSIX just gets us a mostly compatible set of system calls but does not get us everything. If it did, we probably would not need the configure step when building software.

The problem gets even worse when you are not using one of the largest BSDs. Even more exotic operating systems (Plan 9, GNU/HURD, etc.) likely have problems beyond even that. This paper will discuss some of the problems from the perspective of one of the less common BSDs (Bitrig) and some of the ways we work to deal with it. Hopefully these solutions can help other operating systems along with pushing us all towards more compatible and cross-platform software.

2 Bitrig

The work for this paper was primarily done on the Bitrig operating system so a short introduction to Bitrig is useful. Bitrig is a free, fast, and secure Unix-like Open Source operating system [6]. It started as a fork of OpenBSD but with a focus on modern platforms and a modernized tool chain, development tools, and development practices. The supported platforms are amd64 and arm (with experimental work on armd64 (aarch64)).

In terms of modernizing tools and practices, Bitrig uses clang as the base compiler and git for version control. Github.com is used for hosting and dealing with patches (pull requests really) and bugs reports from users. Through the use of modern software development methods we hope to make Bitrig easier for newcomers to contribute to.

There are a number of other features to Bitrig but they are already described elsewhere [10] and are not specifically relevant here.

3 Ports and Packages

3.1 Bitrig Ports

Here we will briefly describe the Bitrig ports system [1]. Bitrig ports are based on OpenBSD ports. We have made some minor changes, but for the most part the systems remain very similar. Ports is a set of Makefiles to download, configure, compile, and package third party software. The package are then installed, removed, or otherwise dealt with using the pkg_add tools which are a set of Perl scripts also inherited from OpenBSD. Ports also includes any patches or other files needed to build or install the software. OpenBSD assumes 'normal' users will only interact with packages and leave ports to developers [4]. Bitrig makes no such assumption.

From the point of view of most third party software, Bitrig should seem pretty much the same as OpenBSD so one might expect to be able to just build the ports tree and have everything work. Unfortunately, this goes wrong in a few ways.

The first way is because Bitrig is not (quite) OpenBSD. We've made some changes that matter to some ports. For starters, instead of using the GNU compiler as the system compile, we use clang. Modern versions of clang are very good and compile almost anything the GNU compilers can handle (for c and c++, FORTRAN and other languages are a different story). Unfortunately, almost is not good enough. There is still a ton of gcc specific code in the wild. In some cases this could be a failure on clang's part but in every case we've actually hit, it is non-standard C that gcc accepts.

The best solution here is to fix the code to work with clang (assuming clang is correct). If that is not possible (some software such as emacs pretty thoroughly assume GNU compilers) the program can still be built with gcc. The ports system can normally handle this with a simple variable (more or less setting CC temporarily) but in some cases, software will hardcode gcc in the Makefiles (cmus is an example I recently came across). This requires patching the Makefile and is something people should not do if they want things to be portable at all. FORTRAN developers have known that hardcoding a specific f77 compilers is a problem for portability for a long time, but in the C/C++ world, gcc has been dominant for long enough that people do hard code it.

Bitrig uses a different C++ library (libc++ and libc++abi instead of libstdc++). This can be fixed in the ports with a simple find and replace and is really just specify to the ports infrastructure. Other changed libraries are handled similarly.

The most common problem is simply that Bitrig is not named OpenBSD. Most nontrivial software (at least the traditional compiled ones) use some combination of configure, automake, and other tools. These were meant to deal with multiple platforms (largely stemming from the days when there were more UNIXes than today). The common mantra is to test for capabilities not names. Unfortunately, no one seems to do that. The vast majority of software seems to test for OS name (and sometimes version!) during the configure step. This is a major problem for any uncommon system. In the case of Bitrig one can usually just modify the config scripts by adding a test for Bitrig in the same place as the OpenBSD test. It is unfortunate that every new OS needs to do this, but without people getting significantly more skilled with configure, we are stuck with it. To avoid this in their own software, developers should either test for capabilities with configure or at least have sensible defaults for unknown operating systems.

3.2 Other Ports/Package systems

It has been suggested by several parties that Bitrig should switch to a different ports and package system. FreeBSD has pkgng [5] and NetBSD has pkgsrc [3]. Both of these have a number of positive features and pkgsrc does work on Bitrig at this time, but they don't solver the fundamental problem. Ultimately, no packages system, no matter how good it is will make software build on something if the code does not support it without patching the code.

That being said, there are benefits to using a packaging system shared by other OSes. Shared packaging systems allow systems to share effort. For smaller systems, this can be a huge benefit. In the case of Bitrig ports it would eliminate the time consuming job of importing changes from the OpenBSD ports repository. This is potentially a big win but there is a price. No system will hold back the larger (and frequently hardest to update) packages such as Firefox and Chromium for one of the less common operating systems they support. It wouldn't make sense for their other users. This means that packages will end up broken and since the packaging system is shared, there is no way for a operating system such as Bitrig to stay on an older version. This leaves developers to either live with potential broken packages or with a rush to test or fix things. This is not a failure of those packaging systems, it is just the reality that there is a price to pay for shared package systems despite the benefits of them which a large part of why Bitrig has not completely adopted such a shared system.

OS	ports/packages
Bitrig	5,000 [1]
OpenBSD	9,451 [9]
NetBSD	14,132 [8]
MacPorts	16,500 [11]
FreeBSD	25,580 [2]
Debian	48,608 [7]

Table 1: Relative number of ports and packages for different systems.

While on the topic of other systems, it is interesting to look at the number of packages or ports supported by each system. Table 1 shows the approximate number of packages each BSD has. This includes macports since that has some similarity to the various BSD systems and Debian for comparison. There are many caveats for these numbers since they vary in time and by platform or architecture. Also the counts cannot be completely comparable since different packagers will break ports up differently or split different flavors of the same software up in different ways. The numbers still give enough of a sense to be useful. There are of course other OSes that could be included (Dragonfly-BSD, Fedora, Gentoo, Nix, cygwin, etc.) but these are the ones the author was most familiar with.

It is tempting to assume that the best system is the one with the most options, in which case we should all use Debian GNU/Linux (by a wide margin). While Debian is a very good OS for a lot of reasons, obviously that is not a conclusion everyone has come to. One of the philosophies that has driven Bitrig (and many of the BSDs) is that minimalism has a value. Maybe dropping packages that have minimal value is a better way forward. If not, those numbers are rather intimidating (particularly when one considers the number of people who actively maintain the packages and ports for most BSDs). Ports based on interpreted languages (Perl, Python, LaTeXpackages, and a few others) usually work untouched on all operating systems improving the numbers although even those sometimes make incompatible assumptions.

4 Upstream It!

The best solution is to get any changes made for your OS accepted into the upstream package. The cost of carrying around patches within your package system is extremely high. Diffs are very fragile by their nature. Minor changes to the original files will break them. This means updating software becomes a time consuming and dangerous task. Equally important, patches kept in a packages system won't benefit other systems.

Of course one could ask why would an upstream author even want patches to support an operating system they do not use (and very possibly few people use). While this might seem fair, we would ask the inverse question: Has any software actually been made worse by being more portable? There are certainly a few cases where a mess of #ifdefs leave unreadable code, but those should be exceptions. If your patches depend on that you should probably reevaluate what you are doing and if it can be done better.

Once you do submit packages upstream, there seem to be three possible actions from the upstream developers:

1. Accept it.
2. Reject it.
3. Ignore it.

In the first case, your work is done. The next time you update the package in your ports system, no patch will be needed. In our experience with Bitrig, this seems to be the most likely outcome. Most authors want their code to work in as many places as possible and are happy to accept patches.

In the second case, you may be able to work with the upstream author to get your patch accepted or you may be stuck supporting the patch forever. We've seen both cases, but generally this doesn't seem to likely (despite the threat of rejection appearing to be a major reason why people do not try to upstream patches).

The third case is the most problematic (and happens regularly enough). The upstream authors just do not respond at all. This might mean the project is abandoned (Sourceforge is almost a graveyard for projects) or it might just be an unresponsive author. It is possible to just keep the patch local as in case two, but that is frequently not the best solution. Better is probably to either fork the software or drop it completely. The common github workflow seems to encourage the forking solution since submitting patches starts with a fork of the repository. If the package is something you (or the project you are working on) actually uses, sometimes forking it and taking over an otherwise dead project is a good idea but for large or complex problems it is often more work then it is worth. The other option is to just drop the software. Sometimes it is better to not keep things that have been abandoned by their authors and will never be updated again. While we all love to point out how many packages are available on our systems (or at least those of us who package software do (see Table 1)), some packages are just not needed any more and probably have security flaws that will never be fixed. We have tried to be fairly cut-throat about removing abandoned packages with Bitrig and even that could probably be done more aggressively.

5 Conclusion

In this paper we described the Bitrig ports system for third party packages. We discussed the challenges of making third party software work on an uncommon system. Systems that support multiple operating systems were briefly discussed as well as their limitations (from the point of view of a packager rather than a user). The need to modify third party code and the benefits of getting those changes into the upstream software was discussed.

6 Acknowledgments

This work was made possible by by the rest of the Bitrig team for all their work on Bitrig, my employer, Company 0, for always encouraging Bitrig, and NYC*BUG for a number of helpful discussions on these and similar topics.

References

[1] DEVELOPERS, T. Bitrig ports. `https://github.com/bitrig/bitrig-ports`. Accessed: 2016-01-06.

[2] DEVELOPERS, T. P. Freebsd ports overall. `http://portsmon.freebsd.org/portsoverall.py`. Accessed: 2016-01-04.

[3] FOUNDATION, T. pkgsrc: portable package build system. `https://www.pkgsrc.org/`. Accessed: 2016-01-04.

[4] OPENBSD. The openbsd packages and ports system. `http://www.openbsd.org/faq/faq15.html#Intro`. Accessed: 2016-01-06.

[5] TEAM, T. Pkg primer. `https://wiki.freebsd.org/pkgng`. Accessed: 2016-01-04.

[6] THE BITRIG DEVELOPERS. Bitrig. `https://www.bitrig.org/`. Accessed: 2016-01-02.

[7] THE DEBIAN PROJECT. All debian packages in "jessie". `https://packages.debian.org/stable/allpackages?format=txt.gz`. Accessed: 2016-01-04.

[8] THE NETBSD PKGSRC DEVELOPERS. The netbsd packages collection. `http://ftp.netbsd.org/pub/pkgsrc/current/pkgsrc/README-all.html`. Accessed: 2016-01-04.

[9] THE OPENBSD PORTS DEVELOPERS. Openbsd ports. `http://ports.su/`. Accessed: 2016-01-04.

[10] VERNALEO, J. C. Bitrig. `http://www.netpurgatory.com/web_stuff/media/2015-05_bitrig.pdf`, 2015. NYC*Bug Meeting.

[11] WIKIPEDIA. Macports. `https://en.wikipedia.org/wiki/MacPorts`. Accessed: 2016-01-04.