A Portable Packaging System

Alistair Crooks, The NetBSD Project

29th September 2004

Abstract

This paper explains the needs for a portable packaging system, and the approach that was taken by the NetBSD pkgsrc team in order first to port their packaging system to another platform. Having learned from this approach, a different and more scalable approach was tried, and has had tremendous benefits. This scalable approach is explained, along with the problems and solutions in supporting multiple different operating systems, and finally the results of this approach are examined and explained.

1 The Need for Portability

A packaging system is an essential part of an architect or administrator's infrastructure, in order to manage the myriad pieces of third-party software that abound. The proliferation of this software is good, but managing a network of even 100 machines in a secure and professional manner can take a huge amount of time. To ameliorate the problems, administrators tend to work with what they know - learning how to work new systems has never been high on anyone's list of things to achieve.

However, even in 1999, the benefits of using an existing packaging system were recognised - to leverage others' work to keep packages up to date with the latest stable versions, and to use the packaging tools to provide a consistent interface for administrators, and to provide a security vulnerability advisory service.

1.1 An overview of pkgsrc

Before a piece of third-party software can be installed on any system, a number of steps must take place:

1. the distribution files, usually in the form of a gzipped or bzip2ed tar file, must be downloaded
2. its integrity is checked against an SHA1 digest stored in the pkgsrc hierarchy
3. any pre-requisite software is also checked to see that a sufficiently up-to-date version is present

4. the source is extracted from the distribution files
5. any patches necessary are applied, using patch(1)
6. the software is configured
7. the software is built
8. the software is installed

Obviously, most of this work is done by shell commands, driven by a large Makefile, and using standard POSIX utilities. pkgsrc automates this process, ensuring that any pre-requisite dependencies are installed and available.

The infrastructure required for pkgsrc itself is a set of packaging tools, collectively referred to as the pkg_install utilities:

- pkg_create registers all the files, links, directories pertaining to a package - this is done at package install time.
- Binary packages of a package installed on a system can be created, either by using pkg_create, or by using a supplementary package called pkg_tarup.
- pkg_add can be used to install a binary package - it can be given a URL, and pkg_add will download a binary package. pkg_add also has been enhanced to use digital signatures of binary packages, if available, to verify the contents of the binary package.
- pkg_delete will delete a package, and the corresponding files from the file system. pkg_delete will calculate an MD5 digest of the file it is about to delete, and will refuse to delete the file if it does not have the same digest as when it was installed.
- pkg_info shows information on the packages that are installed on a system, or gathered together in a separate place
- pkg_admin performs various administrative tasks

There are also a number of standard NetBSD utilities which are needed:

- bmake(1) - the NetBSD make(1) utility
- tnftp(1) - the NetBSD ftp(1) client, used to download distribution files
- digest(1) - a small program to calculate message digests (this was added subsequent to the use of Zoularis)
- pax(1) - the portable archiving tool
- mtree(8) - a tool to create and manage directory hierarchies
- sed(1) - the stream editor
- pkg_install(1) - the NetBSD package management tools

1.2 Security

A benefit of using a standard third-party packaging system such as pkgsrc is that security vulnerabilities in third-party software can be notified to users and administrators of systems automatically. The administrator can then act to mitigate or remove the effects of an intrusion or exploit via the vulnerable package.

Having this available across a range operating systems is another substantial benefit - for more information, see the **pkgsrc/security/audit-packages** package.

2 Porting the Infrastructure

2.1 The First Approach - Zoularis

NetBSD's pkgsrc [pkgsrc2004] grew out of [Freebsd2004] in 1997, and was customised for NetBSD's immediate needs.

These included

- NetBSD platform independence
- and ELF/a.out agnosticism
- and went on to include deterministic package version number comparison, and many others.

There was considerable pressure on the NetBSD pkgsrc team to port their software to other operating systems - primarily Solaris, but also Linux and others - and this was achieved in 1999 by using a heavy compatibility layer for functionality which was missing in the native operating system. The version of Solaris which was used as the target at that time was 2.6, which did not have such functionality as

- strlcpy(3)
- snprintf(3)
- getfsent(3) etc.

Christos Zoulas had done this work already, and so the compatibility layer was called Zoularis.

Zoularis itself was a compatibility layer on top of the native operating system. It provided the extra functionality that was missing from the native libc, and allowed various NetBSD utilities to be built, and these could then be used to manipulate packages. Zoularis had to be made into a binary package, and that binary package had to be installed on every system that used binary packages (since they were built with Zoularis as a pre-requisite dependency).

At the same time, the main Makefile which defines the way that packages are built, and which provides all the definitions and targets, was modified to support Solaris. Usually this was by means of such statements as

```
.if ${OPSYS} == "SunOS"
```

One of the main problems was that of the PLIST, which is a file containing the list of files which make up the binary package. pkgsrc already manipulated these PLISTs to support manual pages being gzipped or not. On different operating systems, the package would sometimes install different files, and so the PLIST had to be manipulated accordingly. This was almost always the case with packages which used *imake* in their build process - manual pages became installed with different suffixes on different operating systems.

Another set of packages that quickly showed itself as being different was those which used libtool. Standard Solaris semantics at that time was to create ELF shared libraries, with 3 numeric suffixes. The PLISTs in pkgsrc had all been written for NetBSD's predominantly a.out shared libraries, which commonly used 2 numeric suffixes. Clearly some manipulation of the PLISTs was necessary, as well as modifications to libtool to create shared libraries with 2 numeric suffixes on Solaris. Later still, more manipulation was necessary for Darwin's .dylib libraries.

Zoularis itself was built using a shell script, operating on a checked out version of the NetBSD source tree (for the NetBSD libc functionality, for libedit, and for tnftp, mtree, tar, the pkg_install tools and the other necessary utilities).

It was clear that Zoularis was one solution, and worked very well in practice. What was also clear was that there may well be less intrusive, more lightweight solutions to portable packaging systems.

2.2 Older and Wiser - the GNU autoconf approach

When the request came to the pkgsrc team to port pkgsrc to Darwin, it had become obvious that a new approach was needed. One of the reasons for this is that the only Darwin machine available at the time was located in Atlanta, Georgia at the end of a fairly thin pipe, the engineer doing the porting was in London, UK; and downloading megabytes of NetBSD source over a thin pipe was not the best approach. Zoularis itself was heavily dependent on the NetBSD sources - it used NetBSD header files, and source files from NetBSD's libc, and the list of files needed to compile Zoularis became long and unwieldy. Binary distributions of Zoularis were made available, but they, too, were dependent on NetBSD's libc version. The size of the NetBSD source code at that time was just over 500 MB, and so it was not a particularly practical proposition to port Zoularis to an existing, known-good platform.

2.2.1 The pkgsrc tools

The first step in the new approach was to provide autoconf-ed versions of all the tools needed by the packaging system. This could even mean that cross-compiling and cross-bootstrapping of a pkgsrc environment could be done. The tools which are GNU autoconf-ed are outlined in section 1.1.

Up until now, pkgsrc has managed to use awk scripts which do not contain any GNU-awk specific extensions, and so there has not been a need to provide a portable version of awk(1). However, more configuration and build scripts are using awk over time, and so it is envisioned that we will eventually provide a version of awk, with

GNU extensions, that is appropriately-licensed, but will be able to deal with gawk extensions.

2.2.2 The pkgsrc Infrastructure

The next step was to examine the pkgsrc system itself. Over time, a lot of the support in the main infrastructure had grown machine-dependent .ifdefs (NetBSD make(1)'s equivalent of the C preprocessor's #ifdef, and roughly analogous to gmake's "ifeq"), and these needed to be excised. To do that, OS-specific definitions files were used, which were sourced by make(1) at run-time. So we now have

- defs.AIX.mk
- defs.BSDOS.mk
- defs.Darwin.mk
- defs.FreeBSD.mk
- defs.IRIX.mk
- defs.Interix.mk
- defs.Linux.mk
- defs.NetBSD.mk
- defs.OpenBSD.mk
- defs.SunOS.mk
- defs.Unixware.mk

whilst more - HP/UX, the Hurd, Digital Unix (Tru64) and UWIN - are planned.

2.3 Binary Packages

Most NetBSD users currently build their own packages from source, although we expect that to change over time to be more akin to the situation which is prevalent in the GNU/Linux world, where binary packages are the norm, and people rarely veer from the straight and narrow compiled-in defaults. Even when this does become the norm in the BSD world, some challenges remain:

- package versioning
- library incompatibilities
- tools unpacking one package built for another architecture or operating system

To address these issues, a number of different approaches have been tried:

- "package versioning" is a problem that applies when an attempt is made to install two conflicting packages at any one time. This is a problem for every packaging system, not just pkgsrc, and has been addressed in a number of ways: [Stow2004], [Depot2004] and in package views [Crooks2002].

- library incompatibilities are related to the package versioning problem - if the ABI of a shared library is changed, by convention software authors bump the major version number of a shared library. Packaging systems which allow one package which uses the old ABI to use the new ABI let down their users - the tool will fail to function properly, and there may be knock-on effects, with other shared library needing to have their version numbers bumped. This is addressed in RedHat's versioning system via a PROVIDES and REQUIRES form of export/import information. pkgsrc has adopted this in its build information, and this may be used in future to determine ABI compatibility.

- modifications have had to be made to the NetBSD packaging tools to stop naive extraction of a package built for a different architecture

3 Portability between Operating Systems

3.1 Differences between native tools

It was found very early on that some of the things in the Open Source world that we take for granted are simply not available on other operating systems. There are two obvious approaches to it - firstly, to work around it, and code to the lowest common denominator, or, secondly, to provide the open source tools in a proprietary world. Whilst we had gone down the second route using a "broad stroke" approach with Zoularis, we thought it was still a superior approach to programming to the lowest common denominator.

With Solaris, we found that there were many things that we had to use - the POSIX XPG4 utilities that were available in /usr/xpg4/bin were necessary to have some hope of using modern shell and other constructs - and this theme extended itself further as we ported pkgsrc to more and more platforms.

In total, to obtain consistency across operating systems, it's necessary to use tools which have a common and well-specified interface. For us, this means POSIX and XPG4. In practice, this is still not enough - there are differences between Solaris's version of ln(1) and almost everyone else's ln(1), when given a command line of "ln -fs file1 file2". NetBSD and other operating systems will unlink(2) the "file2" file system entry before attempting to create the symbolic link, whilst Solaris's will attempt the command but will fail, and just not report an error to the caller. The solution in this case, unfortunately, is to do a sweep of pkgsrc and all the package Makefiles in the hierarchy for any occurrences of "ln -fs" or "ln -sf" and put an explicit "rm -f" before the symbolic link is attempted.

The open source operating systems use either GNU or BSD utilities - to abstract away the differences for each package, a "tools.mk" file was created, which will automatically add a build dependency on GNU awk if a package needs it to build, and the platform does not provide it natively. (Adding a build dependency means that the

GNU awk package would be built and installed, if not already present, as part of the pre-requisite checking during a package's build).

3.2 Platform-independent Infrastructure

The main "bsd.pkg.mk" file, which is the main file included by every package Makefile, has no platform-specific ".if" statements in it. A similar scheme to the GNU autoconfiguration mechanism has been used, where the desired feature is defined in the platform-specific files, and then the platform-independent files test for that definition.

This has many advantages, not least amongst them the ability to support many operating systems, often with no changes to any of the platform-independent files.

pkgsrc has a single file called "bsd.prefs.mk" which will set all the default values for make(1) definitions, including setting any command line options, and any options specified in /etc/mk.conf. Sourcing "bsd.prefs.mk" in a package Makefile also has the effect of making various convenience definitions, such as ${OPSYS}, available.

3.3 Inventories and shared libraries

At present, pkgsrc uses static PLISTs - these are packing lists, or inventories, of the files associated with the package. These PLIST files are distributed with pkgsrc itself. (The last release of pkgsrc, pkgsrc-2004Q3, includes modifications for pkgviews, which is a means of having multiple "conflicting" packages installed at any one time - see [Crooks2002]. Package views supports dynamic packing lists, so that PLIST manipulation is no longer a problem).

A lot of packages provide shared libraries, and other packages can link these libraries in at compile-time or run-time.

Different operating systems have different formats for shared libraries:

- Darwin (Mac OS X) uses a dylib name for its shared libraries
- AIX uses a "lib.a" name for its shared libraries
- a.out libraries on older NetBSD systems and OpenBSD have different naming conventions to ELF libraries used by the same operating system
- Interix has some interesting semantics for shared objects and dynamically-linked libraries

The PLISTs in pkgsrc have all been written to use the ELF shared object name, and libtool has been modified to produce shared libraries which correspond to the same naming convention. For each operating system, at install time, a check is made for the format of shared library. If the platform does not support shared libraries, all of the dynamically linked library names are removed form the PLIST. If an a.out library format is used on the platform, the PLIST is manipulated to include simply the ".a" form of the library. A similar manipulation takes place for AIX, and Darwin.

In addition, the PLISTs have all been modified to check for ".la" archives, if LIBTOOLIZE_PLIST is set, and to generate the appropriate PLIST entries for shared

objects and libtool archives. This means that a single unified PLIST can be used across multiple operating systems with different shared object names and semantics.

3.4 Manual Pages

In a similar manner to shared libraries, manual pages have different characteristics between operating systems. Some always install pre-formatted manual pages with a ".0" suffix, others include a letter suffix to denote the subsystem being used. pkgsrc recognises the operating system being used, and manipulates the PLIST accordingly.

3.5 Packages Specific to One Architecture or Operating System

There are times when packages, often third-party binary packages, are specific to one version of the operating system. In cases like these, a build should not take place if the relevant criteria are not fulfilled. pkgsrc uses a triple:

```
(Operating System, Architecture, Version)
```

to determine whether or not a package can be built and used. The triples are attached to ONLY_FOR_PLATFORM and NOT_FOR_PLATFORM definitions in the package's Makefile in the pkgsrc hierarchy, and can be specified multiple times for multiple versions or platforms.

This example is taken from the **pkgsrc/net/gkrellm-wireless** platform:

```
ONLY_FOR_PLATFORM=   *BSD-*-* Linux-*-*
```

and this one is taken from the **pkgsrc/net/hping** Makefile

```
NOT_FOR_PLATFORM=    NetBSD-0.* NetBSD-1.[0-4]*
```

3.6 Selecting the correct headers and libraries

One of the situations that arose fairly early on was the situation where a package was installed on the system, and a different version of that package was needed to build, or that a system library should always be used in preference to a library installed via the packaging system. To illustrate this problem, consider a build machine where ncurses is installed - by default a number of packages will detect ncurses's presence in ${LOCALSRC} via the GNU configure scripts, and build and link with ncurses. This may not be wanted - ncurses is quite a large package, and to proscribe its installation in a binary package when all that is wanted is curses functionality is to ensure that ncurses will be installed on every machine.

The buildlink framework was introduced by Johnny Lam as a means of making sure that a package being built was compiled with the right header files, and linked with the right libraries, no matter what libraries and headers were installed on a machine. This is done by populating the desired headers and libraries into the build framework in their own hierarchy, and then convincing the build framework to search that hierarchy before any existing system ones.

The initial buildlink implementation worked well, but had some flaws - each occurrence of /usr/local was caught by the infrastructure and converted to be the buildlink hierarchy, which did not work very well for all cases, especially when installing some files into the regular destination, still containing references to the intermediate buildlink hierarchy.

Buildlink2 was written to address these problems, and involves the extension of this idea to provide wrapper scripts around cc(1), as(1), etc. These scripts fixup the necessary paths and then hand off the correct paths to the real tools. This still has some problems, which are especially relevant here, and buildlink3 has been written to address these. It also includes portability enhancements, which is the reason for its mention here.

Originally, Zoularis mandated gcc as the compiler of choice. When the autoconf-ed tools were introduced, there was no longer any reason to mandate gcc - different compilers could be used, such as the Sun Pro compiler. The trouble with using alternative compilers is that the options are not the same - -Wl,-R${LIBDIR}/lib will do exactly what you expect it to do with a gcc/GNU ld combination - a different construction must be used for the Sun Pro compiler.

Buildlink3 extends the abstraction a bit further - a cc script is provided, and the script is called with the normal gcc arguments. If the compiler being used is actually the Sun Pro compiler, then the arguments are fixed up (as well as the paths), in order to provide portability. The same is true for the MIPS pro compilers, and other platform-specific proprietary compilers.

An additional use of buildlink is to ensure that all necessary pre-requisites have been specified in the package Makefile. The way that buildlink3 works is to create bin, include and lib directories inside the ${WRKDIR} - where the package is being built - and to populate those directories with symbolic links to the utilities, header files, and libraries which it will need. The path in the environment is then modified to search the buildlink directories first. Some of the build utilities are invoked via wrapper scripts, which massage the arguments as necessary, making every compiler appear to take the same arguments as gcc, for example. Additional steps are taken to remove all traces to buildlinked files on package installation. If a utility or library is not found, then the package will not build correctly, and so all necessary pre-requisites are found at package compile time.

In practice, buildlink3 provides a tremendously valuable abstraction, since most open source software assumes gcc to be the compiler.

3.7 GNU config.* files

Packages will often include their own config.guess and config.sub files, as part of their own GNU autoconfiguration mechanism. With old packages, these files can be considerably out of date. pkgsrc has a means of overriding these files within a package, so that, even if a package has no means of configuring itself for NetBSD/sh5 (it is unlikely that the package authors will have even heard of such an architecture or operating system), the package will still be able to be configured appropriately on that platform.

3.8 Native or Packaged Software

Occasionally, an operating system will come with certain utilities as standard, whilst such a facility may only be available as a third-party package on other platforms.

As part of the buildlink work, this whole area has been the target of some abstractions, and this has meant that the whole question of "native versus package" is moot.

If the package being built is already part of the native operating system, by default, pkgsrc will prefer it to the packaged version. This can be changed on a package-by-package basis, or as a complete entity.

- the PREFER_PKGSRC and PREFER_NATIVE definitions can be set to the names of a number of packages
- the PREFER_PKGSRC and PREFER_NATIVE definitions can also be set to "yes" or "no" values.

Preferences are determined by the most specific instance of the package in either PREFER_PKGSRC or PREFER_NATIVE. If a package is specified in neither or in both variables, then PREFER_PKGSRC takes precedence over PREFER_NATIVE.

3.9 Platform-dependent definitions

Having reviewed a number of package Makefiles, we found that it was often the case that we were setting a number of compiler flags in an operating system-dependent manner. Typical package Makefiles would look like:

```
.include "../../mk/bsd.prefs.mk"

.if ${OPSYS} == "SunOS"
LIBS+= -lnsl -lsocket
.endif
```

and that was changed to support

```
LIBS.SunOS+=            -lnsl -lsocket
```

This example was taken from the **pkgsrc/net/jwhois** package Makefile.

4 Related Work

The Solarpack project grew out of some work done by Julien Letissier for Sun in 2001, looked a number of different packaging systems, and ended up choosing pkgsrc. Over summer 2001, various packaging systems were examined, and pkgsrc was eventually selected as the packaging system of choice - one of the early attempts was released as [solarpack2002].

[fink2004] wants to bring the full world of Unix Open Source software to Darwin and Mac OS X, using Debian tools like dpkg.

There was a fledgeling project called OpenPackages which also used the NetBSD packaging tools as its base, but it appears that this project is no longer being actively developed.

5 Future Work

The NetBSD project continues to expand and develop the NetBSD packages collection. In December 2003, a release branch of the CVS repository was created, the branch named "pkgsrc-2003Q4". Since then, a branch has been "cut" every three months, and it is intended that future releases will take place at regular three month intervals.

In order to enhance portability using buildlink3, the buildlink3 infrastructure was merged to the pkgsrc branch prior to the branch being created - we learned from previous CVS branches that many commits were made by the team after a branch had been created, and that the infrastrcuture needed to be in place prior to branching so that any subsequent branch could better be maintained.

It is expected that Sun will use some form of pkgsrc as their packaging system of choice for Solaris, starting with the next release.

Further platforms will be added to pkgsrc over the next few months - patches have already been received, and are stored in the NetBSD GNATS database, for HP/UX and GNU/Hurd. There are also modifications to support Tru64 both in GNATS and available through the NetBSD tech-pkg mailing list archives. The problem with providing support for extra platforms is that pkgsrc developers do not, themselves, have access to any hardware or simulators needed to run the operating system in question.

6 Benefits of a Portable Infrastructure

There are a number of benefits which accrue to a portable infrastructure:

- leverage the work of others in porting the software
- leverage the work of others in maintaining the software
- using standard means of security vulnerability notification and fixing
- using standard tools across a range of different machines, with the same interface and behaviour

References

[Stow2004] http://www.gnu.org/software/stow/stow.html

[Depot2004] http://asg.web.cmu.edu/depot/

[Crooks2002] Package Views - a more flexible structure for third-party software, Proceedings of the European BSD Conference, 2002, Amsterdam

[pkgsrc2004] http://www.pkgsrc.org/

[solarpack2002] http://solarpack.sourceforge.net/

[fink2004] http://fink.sourceforge.net/

[Freebsd2004] http://www.freebsd.org/ports/