Package Views - a more flexible infrastructure for third-party software.

Alistair Crooks, Wasabi Systems, agc@wasabisystems.com

6th January 2004

Abstract

Firstly, conventional systems for installation of third party software, including FreeBSD's ports system, NetBSD's pkgsrc, and OpenBSD's ports system, are analysed and compared. In addition, other approaches and infrastructure layouts within the industry are examined. One of the main problems faced by users of the various systems is the means by which multiple versions of a package, or packages which "conflict" with each other, can co-exist at the same time.

To address that, a new system is proposed to allow any number of different versions of packages to co-exist at any one time, and the importance of dynamic packing lists is discussed. The new infrastructure is then described in detail, including the practical aspects of managing a large number of third party packages across a number of different operating systems.

Finally, the lessons learned from its deployment within the NetBSD pkgsrc infrastructure are drawn.

1 Third-party software

In the BSD world, it is not the norm to have software automatically packaged for you. That is the prerogative of operating environments such as Windows and Linux (although different Linux distributors matter to this equation). Because of this, an infrastructure which takes freely-available software and makes it available to others is desirable. This infrastructure must, at a miniumum:

1. Retrieve the software from its home site or a mirror (assuming you are connected in some way to the Internet), or from a CD or other medium.
2. Verify its integrity.
3. Apply any patches.
4. Configure the software for the host operating system, then build and install.
5. Track all installed files to permit easy removal of software using the packaging utilities.
6. Optionally create a binary package that can be installed on other hosts.

Any prerequisite software will also be automatically downloaded, built, and installed.

There are numerous advantages to having an infrastructure which does this automatically:

1. The packages have already been setup to compile and install correctly on your system, so you don't have to worry about porting the software yourself.
2. The latest versions of a program, and its patches are obtained for you, and sorted out so that the software works with NetBSD.
3. It is easy to use, and quick, even over a dialup connection.
4. You can submit additional software to the packaging system, so that others can benefit from your porting work.
5. You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network.

6. The same ease of use and maintenance applies to both binary and source based packages.

7. All packages are installed in a consistent directory tree, including binaries, libraries, man pages, and other documentation.

8. Optional configuration parameters are controlled by a single central config file, including install prefix, acceptable software licenses, and domestic(US) or international encryption requirements.

9. The packages are sorted into categories, providing useful lists of tools to browse through, all guaranteed to work.

10. Pkgsrc knows about primary distribution and mirror sites for source packages, so you can install even when that URL you memorize doesn't work.

This infrastructure helps out people new to the BSD platform by giving them pre-ported software, and helps out the "old lags" too by lifting the burden of having to duplicate the work that others may have done before them.

This, however, is nothing new. The FreeBSD ports collection has been doing this since 1993.

1.1 What is pkgsrc?

In 1997, NetBSD decided to introduce a third-party software infrastructure, and base it on the the FreeBSD ports collection `http://www.freebsd.org/ports/index.html`. The pkg_install tools were imported into the NetBSD CVS repository in June 1997, followed by the basic bsd.port.mk file to the share/ hierarchy, and then the basic pkgsrc infrastructure in early October 1997. For more information on the NetBSD Packages collection, see the relevant documentation on the NetBSD web site a short-cut to the relevant page on the NetBSD web site. `http://www.pkgsrc.org/`

The figures for the growth of pkgsrc are given by Hubert Feyrer in The Growth of the Packages Collection `http://www.netbsd.org/Documentation/software/pkg-growth.html`. There were 3214 packages in the packages collection at the end of September 2002. This compares to over 7000 for FreeBSD, and around 2000 for OpenBSD (although OpenBSD have a "flavors" enhancement to their ports system which reduces the overall number of packages).

One of the problems of bringing over ports from the FreeBSD ports collection has been that NetBSD is primarily a multi-architecture operating system. To the NetBSD people, a "port" means NetBSD running on a different architecture. (17 different processor families, 23 different architectures, 50+ platforms). Hence the name had to change, and the SI unit for NetBSD's third party software collection came to be known as a "package". A place in the CVS repository to house the infrastructure for the packages had to be found, and so that place came to be known as pkgsrc, modelled after the existing basesrc, xsrc, othersrc directories. "pkgsrc" was born.

1.2 How does it differ from the ports collection?

The NetBSD packages team made a number of significant changes. A full list of these changes can be gleaned from the web interface to the NetBSD CVS repository. the web interface to the NetBSD CVS repository `http://cvsweb.netbsd.org/bsdweb.cgi/`.

1. bsd.port.mk and the mtree files were moved to be in the pkgsrc hierarchy, and use relative paths to refer to files within pkgsrc. This allows us to have a number of pkgsrc trees checked out and in use at the same time.

2. Real CONFLICT handling was added to packages.

3. Wildcard and relative matching of package version numbers was added

4. "just-in-time su(1)" functionality was added, so that people can do everything except package installation as unprivileged users

5. pkgsrc was ported to Solaris, and then to Linux and Darwin, so that people can use pkgsrc on those platforms. This used to be done by means of a compatibility layer called Zoularis, but is now done natively, using the othersrc/bootstrap-pkgsrc generic bootstrap kit. Debugging output was improved at the same time. We have a truly generic bsd.pkg.mk, whereby different Operating Systems use abstract

values defined in a defs.${OPSYS}.mk, and these abstractions are then used within bsd.pkg.mk. This makes it much easier to port pkgsrc to other operating systems.

6. The specification of default values was made the same across all packages, with a single file which is automatically included in the make(1) process - bsd.pkg.defaults.mk - and differences from the defaults can be placed in /etc/mk.conf. One single file was made which can be included by package Makefiles in order to pick up standard defaults, and also any differences from the norm as specified in /etc/mk.conf - package Makefiles simply .include "../../mk/bsd.prefs.mk" before any make(1) .if ... conditionals.

7. “buildlink” functionality was introduced, which ensures that the correct files are used in the build and linking processes.

8. Manual pages are not specified in a package Makefile - if a package has files, they are all included in the package’s PLIST.

9. Simple coarse-grained locking was added to pkgsrc using shlock(1). If a package is being built, subsequent attempts to build the same package will lock, waiting for the first package to finish building.

10. The package tools were given the ability to use digitally-signed packages - if a package has been signed, the user can be prompted whether or not to install a package, depending on whether or not the creator of the binary package is trusted.

11. Message digests of all relevant patch files were generated, so that people using sup or extracting patch files over an existing set of patch files will only get the necessary patches applied. (If the digest doesn’t match, the patch file is not applied). Support was added for message digests other than md5 for distfiles and patches, by using the digest package, and support for SHA256 and SHA512 was added to the digest package

12. The ACCEPTABLE_LICENCE feature was added to /etc/mk.conf, to ensure that people only installed packages with whose licence they agreed.

13. Automatic calculation of the effective date of the pkg_install tools is carried out. If the tools are too old, the user will be told this, and how to fix it (typically, by installing the pkg_install package). Full-pathname symbolic links are adjusted in pkg_create(1) to be relative to ${PREFIX}, if appropriate. This helps with binary packages.

14. An xpkgwedge package was added, which makes packages which would normally be installed in ${X11BASE} be installed in ${LOCALBASE}. pkg_info(1) will find out the installed prefix of a package dynamically, rather than guessing at ${X11BASE} or ${LOCALBASE}

15. The MASTER_SITE_SORT definition was added whereby we can sort the MASTER_SITES so that the nearest toplogically get tried first, and FAILOVER_FETCH was added so that, when retrieving distfiles, the distfile digests can be checked, and, if they don’t match, the distfile will be considered incorrect, and the next site will be tried.

16. The object format for shared libraries is determined dynamically at package install time rather than using a hard-coded table - this is much more dynamic, and allows NetBSD ports to migrate from a.out to ELF with no appreciable changes to the pkgsrc infrastructure. All binaries and shared libs are checked after installation to make sure that any shared libs are found correctly by said binaries and other shared libraries.

17. The audit-package package was added, which uses the relational matching of package names to scan a published list of known vulnerabilities, which is maintained by the NetBSD Security Officer, and published on ftp.netbsd.org as the vulnerabilities file `ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/vulnerabilities`, along with a small script to download it. This allows users to be notified automatically if there is a vulnerability in one of their installed packages, and does away with the need for security advisories for packages.

18. “system packages” have been added to the base system, whereby all system utilities and kernels can be

treated as packages, and deleted, added, matched, updated at will.

19. The BUILD_DEPENDS semantics were changed to match the existing DEPENDS syntax - the first component is now a pkg_info(1) recognisable package name (with possible relational or alternate matching)

20. Special handling for the installation of rc.d scripts, create users, and install example files has been added

21. A new framework for handling info files generation and installation was added

22. A “replace” target was introduced, which updates a package in place, modifying any packages which use it. There’s also an “undo-replace” target

23. A finer-grained INTERACTIVE_STAGE definition was introduced, so that builds can continue better unattended

and many, many more enhancements, improvements and speedups.

OpenBSD have also made a number of separate improvements (they have made many more, these are simply some of the main ones)

1. They were the first to speed up the building process by eliminating the use of .USE macros.

2. They have implemented their “FAKE” functionality to provide staged installations (similar to Debian packages).

3. They implemented “flavors” functionality, whereby a package can be built in a number of ways, for example with or without X11 functionality.

FreeBSD have continued to grow their ports collection, and still have the most packages - near 7000 at the last count.

2 The current situation

The conventional *BSD ways of installing software (NetBSD’s pkgsrc, FreeBSD/OpenBSD ports system) install directly into ${LOCALBASE}, possibly overwriting existing files. This has certain disadvantages:

1. There can only be one version of a piece of software installed at any one time. There are numerous occasions when it is desirable to have a newer copy of software to be evaluated, whilst still using the production copy of this. One approach is to install the package to be evaluated by using a different prefix, but there are numerous package management problems with this approach, and it does not scale well.

2. It is often possible that a package overwrites a working version of another unrelated package simply because they contain commands or libraries header files with the same name. Whilst this may seem trivial, and a simple choice has to be made as to the more appropriate package to have installed under these circumstances, it is often more complicated than that. Other packages may demand certain choices be made, which may not be convenient for individual users.

3. Problems can arise when some third party software is upgraded, and a lot of other software depends upon it (libpng, jpeg, zlib). All of the packages which use the updated package have to be re-linked, and the only feasible way to do that is to de-install all of them, with the ensuing problems that that can bring. Various attempts have been made to work around this situation (retiring packages, pkg_hack, “make replace”), but none of these has addressed the fundamental problem.

It is desirable to have a means whereby two packages with the same file system entries can co-exist. As explained above, one method of doing this is to install the newer package into a ${LOCALBASE} in a different location, but this does not scale at all well, and we run into problems with the metadata files in ${PKG_DBDIR}. It is clear that a different approach is needed.

3 Other approaches

Some other approaches to the problems outlined above have been tried:

1. Using separate machines (where they are available) to install newer versions of packages, test their sta-

bility and functionality, and then finally deploy them across a network of machines.

2. "Retiring" packages (where shared objects are retained under a differently-named package) will only work properly when the major number of the shared objects are changed on ELF platforms.

3. OpenBSD's "FAKE", a staged installation approach similar to Debian's, will only allow one version of a package to be installed at any one time. In this case, a binary package is created in the staging area, and that binary package is added to the destination. This has the benefit of creating a binary package which can then be installed on other systems, and otherwise manipulated.

4. CMU's depot software `http://andrew2.andrew.cmu.edu/depot/` is a large piece of software which creates a tiered environment for third party software packages. Some consider it too unwieldy for use in a packaging environment, and mandates an interesting bootstrap procedure and difficult management and configuration problems. One of our pkgsrc developers used this for his own packaging system before switching to pkgsrc (on Solaris). Maintenance was the main reason he cited for this.

5. GNU's stow program `http://www.gnu.ai.mit.edu/software/stow/stow.html` is a very useful program, which uses a tiered approach to software installation. Unfortunately, stow is written in Perl, which again provides us with some bootstrap problems. The program is also distributed under the GPL, and we'd rather not go there.

6. Various other packaging efforts `http://www.encap.org/` also use a tiered approach to the installation of software

After much consideration, it was decided that the approaches outlined above could be improved. Some experiments were made with a staged installation approach, similar to OpenBSD's "FAKE" approach, but other problems with this method encountered. Three approaches to installing a package into a staging area were identified:

1. The package's build mechanism already provided a means of installing into a staging area - packages which have been modified for Debian's ${DESTDIR}, for example, and newer X11-based packages which also installed into ${DESTDIR}. This approach was known as the "DESTDIR" approach.

2. A number of wrapper scripts were written, to enable install(1), ln(1), cp(1) and other programs which are used to install packages into ${LOCALBASE} to take the same arguments as at present, but modify these arguments internally to point to the staging area. This approach was found to be applicable in most circumstances, although we also encountered problems with packages which used GNU libtool, perl and other utilities to install their files, and a surprising number of wrapper scripts had to be written.

3. by setting ${LOCALBASE} to include a specific ${DESTDIR} component, and passing that down to sub-make invocations within the package build and installation procedures.

However, these experiments showed that this approach was simply were papering over the cracks - the base problem (that you can have only one version of a package installed at any one time) still existed, and had not been worked around in any way by this.

4 The aims of package views

Having studied the problem, it was obvious that a better method of installing packages into a destination was necessary. The main aim was that multiple versions of a package should be capable of being installed at any one time. There were also subsidiary aims, too:

1. to allow any number of different versions of packages to co-exist at any one time

2. to allow the testing of different versions of packages on a single machine at any one time

3. to allow more dynamic conflict detection at install time

4. whilst continuing to use the existing pkg_install tools.

4.1 Dynamic Packing Lists

It was subsequently realised that if a package was installed in its own hierarchy, then dynamic PLISTs could also be supported. From its inception, pkgsrc has used a static list of files which constitute the package. This list of files is called a “PLIST”, which is short for “Packing LIST”. Over the years, the PLISTs have taken up more and more time in package maintenance, requiring manipulation for:

- gzipped or standard manual pages
- shared object and library differences by platform and by object format (ELF or a.out)
- changes to reflect other packages installed on a machine (which may not be desired or necessary)
- the machine architecture
- the version of the operating system software
- the version number of the package itself

If PLISTs could be created at installation time, a lot of this extra maintenance would disappear. Dynamic PLISTs require no manual maintenance, and remove a barrier from anyone wishing to create a pkgsrc entry for a new package. Dynamic PLISTs also mean that the manipulations described above do not have to be performed. There are other packaging systems in existence which use dynamic packing lists (Amdahl’s PSF, included in UTS 4.3.3, for example) from which many lessons can be drawn.

5 Package Views

From the basic tenet that multiple versions of a single package need to be installed, it was obvious that a single ${LOCALBASE} directory was insufficient - multiple ${LOCALBASE}s were necessary. It then became obvious that some form of layering would be needed to accomplish this aim. We also observed the way that multiple versions of packages were installed on machines manually by seasoned administrators.

- The basic idea of package views is that a tiered approach, which was later found to be similar to the encap packaging system.
- The basic package is installed into ${LOCALBASE}/packages/${PKGNAME}. This is called the depot directory.
- A custom built shell script is used to build the upper tiers of symbolic links in separate "views", pointing to the files and directories in the depot directory.

Using these ideas, we build up small hierarchies per package. Symbolic links are made to each of the files and symbolic links which constitute a package, and those symbolic links are referenced, rather than the original file within the small hierarchy of the package. For example, with a number of packages installed, the contents of the ${LOCALBASE}/packages directory is shown in Figure 1.

Within each of these “depot” directories, the hierarchy is shown in Figure 2.

As can be seen, the package’s metadata files are kept in the depot directory - this is so that the pkg_install utilities work when used with a ${PKG_DBDIR} value of ${LOCALBASE}/packages (so that relational matching of package names and version numbers continue to work). Once the files have been installed in the depot directory, we then create a “view” of that package’s entries under ${LOCALBASE}. This is called the default view.

We make a “linkfarm” of symbolic links to the entries under ${LOCALBASE}/packages/${PKGNAME} for each of the files and symbolic links in the package. If there is a package-specific directory in the depot directory, it will be created as a directory in ${LOCALBASE}, provided it does not yet exist. If there is already an entry under ${LOCALBASE} with the same name, that symbolic link is replaced by the new symbolic link. This is not such a drastic move as it is at the present time - since the entry under ${LOCALBASE} is merely a symbolic link, the entry in the other depot directory is not touched in any way.

The linkfarm is created by an extra Bourne shell script, and was written to do the same work as the GNU stow program, except for the folding of directories. The linkfarm script takes the same (long and short) arguments as stow, and performs the same job.

When the linkfarm has been created, a +VIEWS metadata file is added to the depot directory. This file contains the views which have been built on top of the depot directory.

```
[15:54:02] agc@sys1 /usr/vpkg/packages 13 > ls -al
total 150
drwxr-xr-x 147 root wheel 4096 May 13 16:09 .
drwxr-xr-x 19 root wheel 512 May 7 20:23 ..
drwxr-xr-x 10 root wheel 512 May 7 15:52 9wm-1.1
drwxr-xr-x 10 root wheel 512 May 7 22:12 GConf-1.0.9
drwxr-xr-x 10 root wheel 512 May 7 17:34 Mesa-3.4.2nb1
drwxr-xr-x 13 root wheel 512 May 7 22:12 ORBit-0.5.15
drwxr-xr-x 10 root wheel 512 Apr 25 09:55 Xaw3d-1.5
drwxr-xr-x 13 root wheel 512 May 13 11:37 a2ps-4.13.0.2
drwxr-xr-x 13 root wheel 512 May 7 16:36 abiword-personal-0.99.5
drwxr-xr-x 13 root wheel 512 Apr 26 19:23 autoconf-2.13
drwxr-xr-x 13 root wheel 512 Apr 26 19:23 automake-1.4.5nb1
drwxr-xr-x 13 root wheel 512 Apr 25 11:15 bison-1.35
drwxr-xr-x 10 root wheel 512 May 7 22:12 bonobo-1.0.18nb1
drwxr-xr-x 10 root wheel 512 May 7 17:34 control-center-1.4.0.4
drwxr-xr-x 13 root wheel 512 May 8 21:40 curl-7.9.6
...  etc ...
```

Figure 1: the contents of the ${LOCALBASE}/packages directory

There is one default view, and all packages have a view in the default view.

Any number of other views can also be created. For example, a “devel” view could be created specifically for packages which have to be tested and evaluated before being put into production use. In a similar way, “kde2”, “kde3” and “gnome2” views could be created in order to appraise those specific groups of packages. We are occasionally asked about putting all GNU utilities under a separate ${PREFIX} in pkgsrc - with package views, these packages can quite simply be pulled up into a “gnu” view.

It should be noted that all packages, even the X11-based ones, need to install into the same ${LOCALBASE} directory. This means that xpkgwedge is obligatory (xpkgwedge puts a package which would normally be destined to be installed under ${X11BASE} into the normal ${LOCALBASE} hierarchy). This has other benefits too, since xpkgwedge preserves the sanctity of what some consider to be system libraries, and reduces the impact upon the installed package hierarchy when a new version of X11 is installed on the computer, although some re-linking may be necessary.

A package may not be deleted from the depot directory if there are any views of that package in existence. This is to preserve the cleanliness of the views model, to keep a principle of cleaning up after ourselves, and to preserve the sanity of system administrators everywhere. The standard pkg_delete(1) command can be used to delete a view, as can the linkfarm script. pkg_delete(1), and linkfarm(1), can also be used to delete a view itself. pkg_info(1) can be used to view packages in the depot directory or in views.

When the next version of the package comes along, because it has a different package name, it gets installed into a different depot directory. The two different versions exist side by side. If the old view in ${LOCALBASE} still exists, the linkfarm script can be used to delete the old view, before making the new view for the new package version. This ensures that packages linking to the package will pick up the entries in the new version of the package.

At the current time, packages link with pre-requisite packages in ${LOCALBASE}. Over time, we may migrate this to link directly to files in the depot directories, so that packages are built with one canonical version, but doing this has other ramifications, such as the ability to have wildcard dependencies on other packages.

```
[16:01:00] agc@sys1 ...vpkg/packages/pth-1.4.1 > ls -al
total 22
-rw-r--r-- 1 root wheel 693 May 7 15:52 +BUILD_INFO
-rw-r--r-- 1 root wheel 374 May 7 15:52 +BUILD_VERSION
-rw-r--r-- 1 root wheel 32 May 7 15:52 +COMMENT
-rw-r--r-- 1 root wheel 224 May 7 15:52 +CONTENTS
-rw-r--r-- 1 root wheel 1063 May 7 15:52 +DESC
-rw-r--r-- 1 root wheel 6 May 7 15:52 +SIZE_ALL
-rw-r--r-- 1 root wheel 6 May 7 15:52 +SIZE_PKG
-rw-r--r-- 1 root wheel 17 May 7 15:52 +VIEWS
drwxr-xr-x 10 root wheel 512 May 7 15:52 .
drwxr-xr-x 147 root wheel 4096 May 13 16:09 ..
drwxr-xr-x 2 root wheel 512 May 7 15:52 bin
drwxr-xr-x 3 root wheel 512 May 7 15:52 etc
drwxr-xr-x 3 root wheel 512 May 7 15:52 include
drwxr-xr-x 2 root wheel 512 May 7 15:52 info
drwxr-xr-x 4 root wheel 512 May 7 15:52 lib
drwxr-xr-x 2 root wheel 512 May 7 15:52 libexec
drwxr-xr-x 25 root wheel 512 May 7 15:52 man
drwxr-xr-x 7 root wheel 512 May 7 15:52 share
[16:01:02] agc@sys1 ...vpkg/packages/pth-1.4.1 >
```

Figure 2: the contents of a “depot” directory

6 Practical Aspects of Package Views

6.1 Views

A package’s files are always in one canonical location, the depot directory. On top of that, views can be constructed.

- There is a default view, which defaults to ${LOCALBASE}.
- Any number of views can be added.
- The traditional NetBSD pkg_install(1) tools are used, with the addition of the script to manage the symbolic link farms.

6.2 ${LOCALBASE} vs. ${X11BASE}

Traditionally, packages have installed into ${LOCALBASE}, or ${X11BASE}, depending upon a number of issues.

NetBSD’s pkgsrc has a utility called xpkgwedge which forces all packages which would normally install into ${X11BASE} into ${LOCALBASE}, thereby keeping the X11 tree "clean".

6.3 ${PREFIX}

With xpkgwedge installed on a computer, all packages now install into ${LOCALBASE}. The floating ${PREFIX} definition is now unnecessary. However, ${PREFIX} is used in most of the packages’ own Makefiles to represent the installation prefix.

We thus “move” the ${PREFIX} definition to refer to the depot directory, ${LOCALBASE}/packages/${PKGNAME}. This gives us a simple and easy way to refer to the depot directory from package Makefiles.

6.4 bsd.pkg.mk internals

At the present time, packages which use GNU configure scripts are passed the item –

prefix=${GNU_CONFIGURE_PREFIX} where ${GNU_CONFIGURE_PREFIX} defaults to ${PREFIX}. With package views, as mentioned above, PREFIX is modified to point to ${LOCALBASE}/packages/${PKGNAME}, and so no further internal manipulation of prefices needs to take place.

6.5 Upgrading packages

Previously, an upgrade or update to a package, especially one containing shared libraries and objects, could be an onerous task, especially (on ELF systems) if a shared library major number change was involved. With packages views, the new package is installed alongside the old one. There are now two possible circumstances (it is assumed that ELF platforms are being used, since almost all systems now use the ELF format):

6.5.1 The majority of cases

In the overwhelming majority of cases, the newer version of the package is installed in its own depot directory, the linkfarm in the default view to the older version is deleted, and a new linkfarm to the newer version is created in the default view. No further changes are necessary, and it is possible to try out other packages which use this package, even if shared libraries are involved. Note that, should the newer version of the package not function as intended, it is a simple matter to revert to the older version, by deleting the linkfarm in the default view to the newer version, and adding a linkfarm to the default view for the older version. As we optimise for the most common occurrence in all things, this approach brings huge benefits.

6.5.2 A shared library major number change

Using the existing "overwrite" meachanism, for a few specific and annoying cases, a major number change for a shared library has meant that those packages, and any other packages which re-use them, have to be re-linked. There have been two memorable occasions over the last year (libpng and libiconv) when this has necessitated a large amount of "make update" work. With package views, this situation does not cause any problems, since the old shared library is still around in its depot directory, and the symbolic link to it still exists from the default view; similarily, the new shared library exists in its depot directory, and a symbolic link to its major version exists in the default view, too:

libwibble.so -> /usr/pkg/packages/wibble-2.0/lib/libwibble.so.2.0
libwibble.so.1 -> /usr/pkg/packages/wibble-1.0/lib/libwibble.so.1.0
libwibble.so.2 -> /usr/pkg/packages/wibble-2.0/lib/libwibble.so.2.0

Whilst the symbolic link to the non-versioned shared library in the default view (libwibble.so) is overwritten, it makes no difference, since that symbolic link is only used for compilation.

7 A Worked Example

7.1 An illustration - the depot directory

The files which constitute the package's entries in the file system are shown in Figure 3.

7.2 An illustration - the default view

The files which constitute the package in the default view are shown in Figure 4.

7.3 The Linkfarms

The symbolic links in the default view, and their targets under the depot directory, are shown in Figure 5.

8 Advantages

With package views, the immediate benefits are the same as the aims:

1. to allow any number of different versions of packages to co-exist at any one time
2. to allow the testing of different versions of packages on a single machine at any one time
3. to allow more dynamic conflict detection at install time

```
[16:42:15] agc@sys1 /usr/vpkg/packages 339 >  env PKG_DBDIR=/usr/vpkg/packages
pkg_info -L pth
Information for pth-1.4.1:
Files:
/usr/vpkg/packages/pth-1.4.1/bin/pth-config
/usr/vpkg/packages/pth-1.4.1/bin/pthread-config
/usr/vpkg/packages/pth-1.4.1/include/pth.h
/usr/vpkg/packages/pth-1.4.1/include/pthread.h
/usr/vpkg/packages/pth-1.4.1/lib/libpth.a
/usr/vpkg/packages/pth-1.4.1/lib/libpth.la
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14
/usr/vpkg/packages/pth-1.4.1/lib/libpth.so.14.21
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.a
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.la
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14
/usr/vpkg/packages/pth-1.4.1/lib/libpthread.so.14.21
/usr/vpkg/packages/pth-1.4.1/man/man1/pth-config.1
/usr/vpkg/packages/pth-1.4.1/man/man1/pthread-config.1
/usr/vpkg/packages/pth-1.4.1/man/man3/pth.3
/usr/vpkg/packages/pth-1.4.1/man/man3/pthread.3
/usr/vpkg/packages/pth-1.4.1/share/aclocal/pth.m4
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/ANNOUNCE
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/AUTHORS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/COPYING
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/HACKING
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/NEWS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/README
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/SUPPORT
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/TESTS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/THANKS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/USERS
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/pthread.ps
/usr/vpkg/packages/pth-1.4.1/share/doc/pth/rse-pmt.ps
[16:42:27] agc@sys1 /usr/vpkg/packages 340 >
```

Figure 3: the contents of the package in the “depot” directory

```
[16:42:27] agc@sys1 /usr/vpkg/packages 340 > pkg_info -L pth
Information for pth-1.4.1:
Files:
/usr/vpkg//bin/pth-config
/usr/vpkg//bin/pthread-config
/usr/vpkg//include/pth.h
/usr/vpkg//include/pthread.h
/usr/vpkg//lib/libpth.a
/usr/vpkg//lib/libpth.la
/usr/vpkg//lib/libpth.so
/usr/vpkg//lib/libpth.so.14
/usr/vpkg//lib/libpth.so.14.21
/usr/vpkg//lib/libpthread.a
/usr/vpkg//lib/libpthread.la
/usr/vpkg//lib/libpthread.so
/usr/vpkg//lib/libpthread.so.14
/usr/vpkg//lib/libpthread.so.14.21
/usr/vpkg//man/man1/pth-config.1
/usr/vpkg//man/man1/pthread-config.1
/usr/vpkg//man/man3/pth.3
/usr/vpkg//man/man3/pthread.3
/usr/vpkg//share/aclocal/pth.m4
/usr/vpkg//share/doc/pth/ANNOUNCE
/usr/vpkg//share/doc/pth/AUTHORS
/usr/vpkg//share/doc/pth/COPYING
/usr/vpkg//share/doc/pth/HACKING
/usr/vpkg//share/doc/pth/NEWS
/usr/vpkg//share/doc/pth/README
/usr/vpkg//share/doc/pth/SUPPORT
/usr/vpkg//share/doc/pth/TESTS
/usr/vpkg//share/doc/pth/THANKS
/usr/vpkg//share/doc/pth/USERS
/usr/vpkg//share/doc/pth/pthread.ps
/usr/vpkg//share/doc/pth/rse-pmt.ps
[16:42:41] agc@sys1 /usr/vpkg/packages 340 >
```

Figure 4: the contents of the package in the default view

```
[16:42:41] agc@sys1 /usr/vpkg/packages 341 > ls -al ‘pkg_info -qL pth‘
lrwxr-xr-x 1 root wheel 43 Apr 24 09:28 /usr/vpkg//bin/pth-config -> /usr/vpkg/packages/pth-1.4.1/bin/pth-
config
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//bin/pthread-config -> /usr/vpkg/packages/pth-
1.4.1/bin/pthread-config
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//include/pth.h -> /usr/vpkg/packages/pth-1.4.1/include/pth.h
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//include/pthread.h -> /usr/vpkg/packages/pth-
1.4.1/include/pthread.h
lrwxr-xr-x 1 root wheel 41 Apr 24 09:28 /usr/vpkg//lib/libpth.a -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.a
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//lib/libpth.la -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.la
lrwxr-xr-x 1 root wheel 42 Apr 24 09:28 /usr/vpkg//lib/libpth.so -> /usr/vpkg/packages/pth-1.4.1/lib/libpth.so
lrwxr-xr-x 1 root wheel 45 Apr 24 09:28 /usr/vpkg//lib/libpth.so.14 -> /usr/vpkg/packages/pth-
1.4.1/lib/libpth.so.14
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//lib/libpth.so.14.21 -> /usr/vpkg/packages/pth-
1.4.1/lib/libpth.so.14.21
lrwxr-xr-x 1 root wheel 45 Apr 24 09:28 /usr/vpkg//lib/libpthread.a -> /usr/vpkg/packages/pth-
1.4.1/lib/libpthread.a
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//lib/libpthread.la -> /usr/vpkg/packages/pth-
1.4.1/lib/libpthread.la
lrwxr-xr-x 1 root wheel 46 Apr 24 09:28 /usr/vpkg//lib/libpthread.so -> /usr/vpkg/packages/pth-
1.4.1/lib/libpthread.so
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//lib/libpthread.so.14 -> /usr/vpkg/packages/pth-
1.4.1/lib/libpthread.so.14
lrwxr-xr-x 1 root wheel 52 Apr 24 09:28 /usr/vpkg//lib/libpthread.so.14.21 -> /usr/vpkg/packages/pth-
1.4.1/lib/libpthread.so.14.21
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//man/man1/pth-config.1 -> /usr/vpkg/packages/pth-
1.4.1/man/man1/pth-config.1
lrwxr-xr-x 1 root wheel 54 Apr 24 09:28 /usr/vpkg//man/man1/pthread-config.1 -> /usr/vpkg/packages/pth-
1.4.1/man/man1/pthread-config.1
lrwxr-xr-x 1 root wheel 43 Apr 24 09:28 /usr/vpkg//man/man3/pth.3 -> /usr/vpkg/packages/pth-
1.4.1/man/man3/pth.3
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//man/man3/pthread.3 -> /usr/vpkg/packages/pth-
1.4.1/man/man3/pthread.3
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/aclocal/pth.m4 -> /usr/vpkg/packages/pth-
1.4.1/share/aclocal/pth.m4
lrwxr-xr-x 1 root wheel 51 Apr 24 09:28 /usr/vpkg//share/doc/pth/ANNOUNCE -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/ANNOUNCE
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/AUTHORS -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/AUTHORS
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/COPYING -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/COPYING
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/HACKING -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/HACKING
lrwxr-xr-x 1 root wheel 47 Apr 24 09:28 /usr/vpkg//share/doc/pth/NEWS -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/NEWS
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/doc/pth/README -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/README
lrwxr-xr-x 1 root wheel 50 Apr 24 09:28 /usr/vpkg//share/doc/pth/SUPPORT -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/SUPPORT
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//share/doc/pth/TESTS -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/TESTS
lrwxr-xr-x 1 root wheel 49 Apr 24 09:28 /usr/vpkg//share/doc/pth/THANKS -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/THANKS
lrwxr-xr-x 1 root wheel 48 Apr 24 09:28 /usr/vpkg//share/doc/pth/USERS -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/USERS
lrwxr-xr-x 1 root wheel 53 Apr 24 09:28 /usr/vpkg//share/doc/pth/pthread.ps -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/pthread.ps
lrwxr-xr-x 1 root wheel 53 Apr 24 09:28 /usr/vpkg//share/doc/pth/rse-pmt.ps -> /usr/vpkg/packages/pth-
1.4.1/share/doc/pth/rse-pmt.ps
[16:43:05] agc@sys1 /usr/vpkg/packages 342 >
```

Figure 5: the target symbolic links of the package view

4. whilst continuing to use the existing pkg_install tools, and

5. to provide support for dynamic packing lists

and, quite unexpectedly, other advantages were gained:

1. it is immediately obvious to which package a file or directory belongs

2. many additional views can be built up - package views are scalable

3. pkg_delete(1) deletes links in the views as well as the package itself

4. multiple conflicting packages (not just multiple versions of one package) can be installed at the same time

5. development packages can be tested and evaluated on the same machine on which they will eventually run

This is portable to any system on which pkgsrc runs - NetBSD, Solaris, Darwin, and Linux. FreeBSD, Irix, Digital Unix and HP/UX are currently in the works, although the generic bootstrap kit should work on any POSIX-compliant system.

Users can migrate to package views simply by setting an /etc/mk.conf variable definition. For cleanliness, it would be better to move to a complete package views system at one time, and so a pkgsrc flag day is on the cards. In reality, the current "overwrite" functionality and "pkgviews" functionality can coexist until such time as migration to package views has taken place.

In all, the package views approach is scalable in practice (see similar papers on the infrastructure.org web site `http://www.infrastructure.org/`, and from experience of other highly-experienced system administrators).

9 Disadvantages

Of course, there are disadvantages to this approach:

1. Some people think that the linkfarms are unruly, unsightly and ugly.

2. A minimal amount of extra space is used to provide the linkfarm. The early versions of package views had code to use "hard" links rather than symbolic links to achieve the same effect. This was possible, since it is highly likely that a file and its link will reside on the same file system. Where this approach failed was in configuration files, which may be edited by people using popular editors which create a new file rather than a "hard" link to a file when the editing session is saved. In all, however, some extra space is used to store the symbolic link information, but, in the whole scheme of things, with falling disk costs and increasing disk capacities, it is no more than a fraction of a percentage of the total disk space used, and so can be discounted for all practical purposes.

In all, with different versions of packages to be installed side by side, more disk space in general will be needed (this is more of a consequence than a disadvantage), which may not always be appropriate (NetBSD still runs on a number of systems, like the VAX and acorn26, where directly attached disk space is at a premium). One suggestion for this is to use NFS or cheaper, mass-produced IDE discs (where possible).

10 Conclusions

The advantages of being able to have two different versions of a package installed at any one time are immense. It is now possible to try out new versions of packages without compromising the existing version. The move to dynamic packing lists will simplify pkgsrc entry creation for everyone, and reduce the amount of maintenance which has to be performed on the current packages, including all the special cases for different operating systems and object formats. In addition, package views allow us to detect conflicts at package install time, rather than by specifying this as a static definition in a package Makefile, and resolve the conflicts in a non-destructive way. The existing package tools can continue to be used, and the symbolic link farms, whilst ugly, give an immediate idea of the package to which a file entry belongs. Whilst using package views, a direct increase in the amount of disk space which will be used is only to be expected. The utility value of the advantages far outweigh the disadvan-

tages.

11 Future directions

At present, package views are implemented on a CVS branch within the NetBSD CVS repository. We intend to take the following steps within pkgsrc:

1. Make xpkgwedge the default for all packages, having first made sure via a bulk build that all packages are xpkgwedge-friendly
2. Introduce package views by merging the **pkgviews** branch in the NetBSD pkgsrc CVS repository with the trunk. At the current time the default is not to use package views - they are only used if the definition **PKG_INSTALLATION_TYPE** is set to **pkgviews.** The default value for **PKG_INSTALLATION_TYPE** is **overwrite**.
3. When that has been done, we will switch over to dynamic PLISTs in pkgsrc. This will be done by using the **PLIST_TYPE** definition to **dynamic**. The default value for **PLIST_TYPE** is **static**.
4. Monitor reaction to package views and dynamic PLISTs, and to improve upon it where possible

References

[FreeBSD] http://www.freebsd.org/ports/index.html - FreeBSD Ports

[Pkgsrc] http://www.pkgsrc.org/ - A shortcut to the pkgsrc area on the NetBSD website.

[Feyrer] http://www.netbsd.org/Documentation/software/pkg-growth.html - The Growth of the packages Collection

[NetBSD] http://cvsweb.netbsd.org/bsdweb.cgi/ - a web interface to the NetBSD CVS Repository

[CMU] http://andrew2.andrew.cmu.edu/depot/ - The Depot Configuration Management Project

[GNU] http://www.gnu.ai.mit.edu/software/stow/stow.html - GNU Stow

[Encap] http://www.encap.org/ - The Encap Archive

[Infrastructures] http://www.infrastructures.org/ Infrastructures.org - Best Practices in Automated Systems Administration and Infrastructure Architecture

[Vulnerabilities] ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/vu - A List of Known Vulnerabilities in Packages