What is a packaging system?

- A set of tools used to maintain a large number of open source programs
- Provides an inventory of all third-party files that are currently installed
- Provides a coherent way to handle program/library dependencies
- Provides binary (pre-built) packages in a centralised location with regular updates

Why use a packaging system?

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.

-- George Bernard Shaw

Why use a packaging system?

- Central repository of knowledge:
  - No need for site to discover/fix the same bug
  - Ready to run packages (minimal configuration required)
  - Get other, new packages "for free"
- Prompt, security-related bug fixes
- Easier updates for a large number of hosts
- No programming experience/development environment required
- Automatic handling of package dependencies

Packaging system features

- Tools completely self-contained
- Dependency and conflict handling
- Just-in-time su for installation
- Ability to build from source
- Third-party license handling
- Cryptographic signature verification of package
- Ability to build in sandbox
- Ability to install multiple versions of same package

What pkgsrc does

- Retrieve the software distribution and any official patches
- Verify its integrity
- Apply pkgsrc patches and any local patches
- Configure the software for the host operating system, build and install
- Track all installed files to permit easy removal of the software using the packaging utilities
- Optionally create a binary package which can be installed on other hosts

Any prerequisite software will automatically be built using the same procedure

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with NetBSD
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with Solaris
- It is easy to use, and quick, even over a dialup connection
- You can submit additional software to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with Linux
- It is easy to use, and quick, even over a dialup connection
- You can submit additional software to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with Mac OS X
- It is easy to use, and quick, even over a dialup connection
- You can submit additional software to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with FreeBSD
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with OpenBSD
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with IRIX
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with BSD/OS
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with AIX
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with Interix
- It is easy to use, and quick, even over a dialup connection
- Additional software can be submitted to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages

- 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
- The latest stable version of a program, and its patches are obtained for you, and sorted out so that the software works with Unixware
- It is easy to use, and quick, even over a dialup connection
- You can submit additional software to the packaging system, so that others can benefit from your porting work
- You can create scripts easily to install sets of packages and maintain the standard software for hosts on your network
- The same ease of use and maintenance applies to both binary and source based packages

Advantages (continued)

- All packages are installed in a consistent directory tree, including binaries, libraries, man pages, and other documentation
- Optional configuration parameters are controlled by a single central config file, including installation prefix, acceptable software licenses, and other configuration parameters
- The packages are sorted into categories, providing useful lists of tools to browse through, all guaranteed to work
- Pkgsrc knows about primary distribution and mirror sites for source packages, so you can install even when that URL you memorise doesn’t work

People who benefit

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

History

NetBSD’s pkgsrc grew out of the FreeBSD ports collection in early 1997. Firstly, the pkg_install tools were imported, then the ports infrastructure, and then each "port" piecemeal

Terminology

Some of the terminology had to be changed. In NetBSD, a "port" is a platform to which NetBSD has been ported. Another name was necessary...

Package

The word "package" was used by FreeBSD to represent a piece of ported software which was already compiled, and that word seemed appropriate. In NetBSD, base source is held under basesrc, X src under xsrc, GNU src under gnusrc, and so package sources came to live in a CVS module called pkgsrc. pkgsrc was born

Infrastructure

A lot of the infrastructure needed to be expanded to work with ports which are ELF-based, and, later, on different operating systems. We also wanted to modify the infrastructure in other ways, too

Changes to pkgsrc

A list of changes made to pkgsrc since its inception

Changes (1)

bsd.port.mk was moved to the pkgsrc hierarchy, and relative paths are used 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

Changes (2)

Real CONFLICT handling was added to packages

```
$ grep CONFLICTS mpg*/Makefile
mpg123-nas/Makefile:CONFLICTS+= mpg123-[0-9]*
mpg123/Makefile:CONFLICTS+=     mpg123-nas-[0-9]*
$
```

Changes (3)

Relative matching of package version numbers were added

csh(1)-style alternates were introduced

Changes (3)

Relative matching of package version numbers were added

csh(1)-style alternates were introduced

(OK, I admit it, it was because I was told it couldn’t be done)

Changes (4)

"just-in-time su(1)" functionality was added, so that people can run as unprivileged users for every operation, and be prompted by su(1), priv or sudo when necessary

Changes (5)

Manual pages are automatically catered for, whether or not they're gzipped, and the PLIST fixed up accordingly

Changes (6)

PLISTs are automatically modified for ELF vs a.out shared objects and shared libraries

Changes (7)

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 pkgsrc/bootstrap generic bootstrap kit

Changes (8)

ONLY_FOR_PLATFORM/NOT_FOR_PLATFORM definitions allow us to specify on which platforms a package will or will not work. This takes the form of a triple:

- OS-version-platform

e.g.

- ONLY_FOR_PLATFORM= NetBSD-*-i386

Changes (9)

Build versions and build information were added to binary packages, which allow us to tell with what definitions a packages was built

Changes (9) continued

[12:43:37] agc@sys1 /usr/pkgsrc/audio 46 > pkg_info -B mpg123-esound

Information for mpg123-esound-0.59.18:

Build information:
USE_INET6= YES
PKG_SYSCONFDIR= /usr/pkg/etc
PKGPATH= audio/mpg123-esound
OPSYS= NetBSD
OS_VERSION= 1.6I
MACHINE_ARCH= i386
MACHINE_GNU_ARCH= i386
CPPFLAGS= -DINET6 -I/usr/pkg/include
CFLAGS= -O2 -I/usr/pkg/include
FFLAGS= -O
LDFLAGS=   -Wl,-R/usr/pkg/lib -L/usr/pkg/lib
CONFIGURE_ENV= MAKE="make" LDFLAGS=" -Wl,-R/usr/pkg/lib -L/usr/pkg/lib" M4="/usr/bin/m4" YACC="yacc"
PATH=/usr/obj/pkgsrc/audio/mpg123-esound/work.sys1/.buildlink/bin:/usr/pkg/bin:/usr/pkg/sbin:/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin:/usr/local/bin:/usr/games:/usr/pkg/java/bin:/sbin:/usr/sbin:/u
PKG_SYSCONFDIR="/usr/pkg/etc" BUILDLINK_DIR="/usr/obj/pkgsrc/audio/mpg123-esound/work.sys1/.buildlink"
BUILDLINK_X11PKG_DIR="/usr/obj/pkgsrc/audio/mpg123-esound/work.sys1/.buildlink-x11pkg" BUILDLINK_UPDATE_CACHE=no BUILDLINK_CPPFLAGS="-I/usr/pkg/include"
BUILDLINK_LDFLAGS="-L/usr/pkg/lib -Wl,-R/usr/pkg/lib" CC="cc" CXX="c++" LD="ld"
CONFIGURE_ARGS=
OBJECT_FMT= ELF
LICENSE=
RESTRICTED=
NO_SRC_ON_FTP=
NO_SRC_ON_CDROM=
NO_BIN_ON_FTP=
NO_BIN_ON_CDROM=
CC=   cc-2.95.3
_PKGTOOLS_VER=20020827

Changes (9) continued

[12:43:43] agc@sys1 /usr/pkgsrc/audio 47 > pkg_info -b mpg123-esound

Information for mpg123-esound-0.59.18:

Build version:
audio/mpg123-esound/Makefile:# $NetBSD: Makefile,v 1.3 2002/09/06 11:51:59 wiz Exp $
audio/mpg123-esound/Makefile:# $NetBSD: Makefile,v 1.3 2002/09/06 11:51:59 wiz Exp $
audio/mpg123-esound/PLIST:@comment $NetBSD: PLIST,v 1.1 2002/06/22 17:56:38 kent Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-aa:$NetBSD: patch-aa,v 1.22 2002/09/06 11:51:59 wiz Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ab:$NetBSD: patch-ab,v 1.3 1999/07/18 19:23:55 tron Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ac:$NetBSD: patch-ac,v 1.3 1999/09/27 08:27:46 agc Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ad:$NetBSD: patch-ad,v 1.3 1999/10/12 04:43:12 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ae:$NetBSD: patch-ae,v 1.5 1999/10/12 04:43:13 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-af:$NetBSD: patch-af,v 1.1 1999/04/08 07:35:56 tron Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ag:$NetBSD: patch-ag,v 1.3 1999/10/12 04:43:13 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ah:$NetBSD: patch-ah,v 1.3 1999/10/12 04:43:14 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ai:$NetBSD: patch-ai,v 1.3 2002/02/22 13:17:54 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-aj:$NetBSD: patch-aj,v 1.2 2002/06/23 08:45:09 kent Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ak:$NetBSD: patch-ak,v 1.1 1999/10/12 04:43:15 simonb Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-al:$NetBSD: patch-al,v 1.3 2001/05/12 20:21:37 mycroft Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-am:$NetBSD: patch-am,v 1.1 2002/02/27 21:37:40 martin Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-an:$NetBSD: patch-an,v 1.1 2002/02/27 21:37:41 martin Exp $
audio/mpg123-esound/../../audio/mpg123/patches/patch-ao:$NetBSD: patch-ao,v 1.1 2002/06/22 17:56:37 kent Exp $

Changes (10)

Buildlink functionality was introduced, which ensures that the correct files are used in the build and linking process.

Consider, for example, if someone is building a new version of a package, and there already exists a version of that package in the destination directory.

Because of a pre-requisite library and header files being in the destination directory, linking must take place with that destination directory, and so old header files may be picked up in the build process, or it may be linked with old libraries.

Changes (10) continued

Buildlink removes this problem by using symbolic links to point to the correct files in separate directories.

Buildlink2 functionality does this in a transparent way.

Buildlink3 functionality does this in an extensible, portable, transparent way.

Buildlink functionality also removes the problem of "does this package use ncurses or curses"

Changes (10) continued

```
.include "../../devel/gettext-lib/buildlink2.mk"
.include "../../graphics/gdk-pixbuf/buildlink2.mk"
.include "../../x11/gtk/buildlink2.mk"
.include "../../mk/bsd.pkg.mk"
```

Changes (10) continued

Buildlink2 has problems of its own

- it doesn’t sit well with package views
- other operating systems have different needs

Buildlink3 was born

Buildlink3 can be used to make all systems appear to have consistent utilities by using wrapper scripts

Changes (11)

One single file is used, which can be included by package Makefiles, to pick up standard defaults, and also any differences from the norm as specified in /etc/mk.conf - package Makefiles

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

before any make(1) .if ... conditionals. All possible Makefile definitions are documented in bsd.prefs.mk

Changes (12)

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

Changes (13)

Simple 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

In /etc/mk.conf:

...
PKGSRC_LOCKTYPE= sleep
...

OBJHOSTNAME must also be set.

Changes (14)

The ability to use digitally-signed packages was added - 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

```
$ sudo pkg_add -s gpg $PKGREPOSITORY/skill-4.0.tgz
gpg: Signature made Fri Sep 21 13:07:56 2001 BST using DSA key ID 26B1CB95
gpg: Good signature from "Alistair Crooks "TEST KEY" <agc@pkgsrc.org>"
Proceed with addition of /usr/packages/i386/skill-4.0.tgz: [y/n]? y
$
```

Changes (15)

The funny LIB_DEPENDS=regexp functionality was removed, and all the choices about installing a package or not are based on relational version number matching

From pkgsrc/audio/gqmpeg on my system

```
DEPENDS+=       mpg123-{,esound}>=0.59.18:../../audio/mpg123
DEPENDS+=       vorbis-tools>=1.0.0.6:../../audio/vorbis-tools
DEPENDS+=       xmp>=2.0.2:../../audio/xmp
```

Changes (16)

Debugging output was added to all the necessary targets, conditional upon PKG_DEBUG_LEVEL. e.g.

$ make

might print weird errors due to a shell quoting bug

$ make PKG_DEBUG_LEVEL=2

will show you what the shell quoting problem is (and much, much more)

Changes (17)

HOMEPAGE definitions were added to all of our packages, which are gathered together in the generated README.html files

Changes (18)

It’s possible to have a completely read-only pkgsrc, so that building from a pkgsrc hierarchy on a CD is possible (unusual, but possible)

Changes (19)

We have moved to a scheme of "one file per patch file"

More extravagant in terms of CVS usage, much easier to use in reality

Changes (20)

Message digests of all relevant patch files were added, 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)

Changes (21)

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

The user has to specify in advance which licences are acceptable to them. Packages with unacceptable licences will not be built

Changes (22)

We believe we were the first to introduce bulk building of packages

We use output from i386-platform bulk build runs as a release criterion

Changes (23)

The effective date of the pkg_install tools is calculated automatically. If the tools aren't old enough, the user will be told this, and how to fix it (typically, by installing the pkg_install package)

Changes (24)

A digest package was added

Still some controversy over this one

Changes (25)

The xpkgwedge package was added, which makes packages which would normally be installed in ${X11BASE} be installed in ${LOCALBASE}

Changes (26)

The mtree files were moved to the pkgsrc tree

Changes (27)

If any full-pathname symbolic links are encountered by pkg_create(1), adjust them to be relative to ${PREFIX}, if appropriate. This helps with binary packages

Changes (28)

All GNU awk-isms were eradicated from bsd.pkg.mk

Changes (29)

"Failover fetch" functionality was added when retrieving distfiles, so that the digests can be checked, and, if they don’t match, the distfile will be considered incorrect, and the next site will be tried

Changes (30)

The type of shared library is derived dynamically at 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

Changes (31)

A definition was added whereby we can sort the MASTER_SITES topologically

```
[13:09:02] agc@sys1 ...pkgsrc/audio/vorbis-tools 60 > more /etc/mk.conf
 # pkgsrc definitions
DISTDIR= /usr/distfiles
WRKOBJDIR= /usr/obj/pkgsrc
OBJHOSTNAME= true
PACKAGES= /usr/packages/i386

PAPERSIZE= A4
PKG_VERBOSE= yes
SMART_MESSAGES= yes
PKG_DEVELOPER= yes
OBJHOSTNAME= yes
SU_CMD= priv sh -c
PKGSRC_LOCKTYPE= sleep
PKGSRC_USE_REPLACE= yes

PKG_VIEWS=yes
INSTALLATION_TYPE= staged

_ACCEPTABLE= yes

MASTER_SORT=    .uk .fi .ie .de .ch .se .no .fr .be .ac.at .at
...
```

Changes (32)

We have a truly generic bsd.pkg.mk, whereby different Operating Systems have values defined in a defs.${OPSYS}.mk, and these abstractions are then used within bsd.pkg.mk. For example,

```
_DO_LIBINTL_CHECKS=  yes  # perform checks for valid libintl
_DO_SHLIB_CHECKS=  yes  # fixup PLIST for shared libs/run ldconfig
_IMAKE_MAKE=   ${MAKE}  # program which gets invoked by imake
_OPSYS_HAS_GMAKE=  no  # GNU make is not standard
_OPSYS_HAS_MANZ=  yes  # MANZ controls gzipping of man pages
_OPSYS_HAS_OSSAUDIO= yes       # libossaudio is available
_PATCH_BACKUP_ARG=  -V simple -b     # switch to patch(1) for backup suffix
_PREFORMATTED_MAN_DIR= cat       # directory where catman pages are
_USE_RPATH=   yes      # add rpath to LDFLAGS
```

Changes (33)

An audit-package package was added, which uses the relational matching of package names to match against a published list of known vulnerabilities

- the vulnerability list is maintained by the NetBSD security officer
- the list is published on ftp.netbsd.org
- there is a small script to download the "known vulnerabilities" file

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.

Changes (33) continued

```
[13:09:13] agc@sys1 ...pkgsrc/audio/vorbis-tools 61 > audit-packages
Package mozilla-1.0nb2 has a remote-file-read vulnerability, see http://archives.neohapsis.com/archives/bugtraq/2002-07/0259.html
Package mozilla-1.0nb2 has a remote-file-read vulnerability, see http://www.geocities.co.jp/SiliconValley/1667/advisory03e.html
Package suse_base-7.3 has a remote-code-execution vulnerability, see http://www.suse.com/de/security/2002_031_glibc.html
[13:13:51] agc@sys1 ...pkgsrc/audio/vorbis-tools 62 >
```

Changes (34)

"system packages" were added to the base system, whereby all system utilities and kernels can be treated as packages, and deleted, added, matched, updated at will

Changes (35)

The size of packages is recorded in a separate metadata file in the ${PKG_DBDIR}

[22:02:25] agc@sys1 ...pkgsrc/editors/ssam 71 > pkg_info -S perl
Information for perl-5.6.1nb7:

Size in bytes including required pkgs: 18352730

[22:02:33] agc@sys1 ...pkgsrc/editors/ssam 72 >

Changes (36)

The object format of prerequisite packages is checked before attempting to build with them, which simplifies the move between a.out and ELF object formats. It’s possible to abort the build, or to continue blindly on, depending on an /etc/mk.conf definition

Changes (37)

All fuzz was removed from patches in pkgsrc

I suspect we may need to do this again

Changes (38)

In informational messages to the user, use ’=>’ in preference to ’>>’, so that cut-n-paste into send-pr will work correctly

Changes (39)

When SMART_MESSAGES is defined, when compiling packages, the make(1) target is displayed, and also the current stack of packages being built

```
[13:01:56] agc@sys1 ...pkgsrc/audio/vorbis-tools 57 > make
=> Checksum OK for vorbis-tools-1.0.tar.gz.
work.sys1 -> /usr/obj/pkgsrc/audio/vorbis-tools/work.sys1
=> Lock acquired on behalf of process 7500
===> extract-message [vorbis-tools-1.0.0.8nb1] ===> Extracting for vorbis-tools-1.0.0.8nb1
===> install-depends [vorbis-tools-1.0.0.8nb1] ===> Required package libao>=0.8.3: NOT found
===> install-depends [vorbis-tools-1.0.0.8nb1] ===> Verifying reinstall for ../../audio/libao
=> Checksum OK for libao-0.8.3.tar.gz.
work.sys1 -> /usr/obj/pkgsrc/audio/libao/work.sys1
=> Lock acquired on behalf of process 7627
===> extract-message [libao-0.8.3nb1, vorbis-tools-1.0.0.8nb1] ===> Extracting for libao-0.8.3nb1
===> install-depends [libao-0.8.3nb1, vorbis-tools-1.0.0.8nb1] ===> Required installed package gmake>=3.78:
gmake-3.79.1 found
===> install-depends [libao-0.8.3nb1, vorbis-tools-1.0.0.8nb1] ===> Required installed package
libtool-base>=1.4.20010614nb9: libtool-base-1.4.20010614nb9 found
=> Lock released on behalf of process 7627
=> Lock acquired on behalf of process 7627
===> patch-message [libao-0.8.3nb1, vorbis-tools-1.0.0.8nb1] ===> Patching for libao-0.8.3nb1
===> do-patch [libao-0.8.3nb1, vorbis-tools-1.0.0.8nb1] ===> Applying NetBSD patches for libao-0.8.3nb1
...
```

Changes (40)

All binaries and shared libraries are checked after installation to make sure that shared libraries are found correctly by said binaries and other shared libraries. PKG_DEVELOPER must be set to enable this

```
...
===> do-shlib-handling [libutf-2.10, ssam-1.9] ===> [Automatic ELF shared object handling]
===> fake-pkg [libutf-2.10, ssam-1.9] ===> Registering installation for libutf-2.10
/usr/bin/ldd /usr/pkg/lib/libutf.so
/usr/bin/ldd /usr/pkg/lib/libutf.so.2
/usr/bin/ldd /usr/pkg/lib/libutf.so.2.10
===> install-depends [ssam-1.9] ===> Returning to build of ssam-1.9
=> Lock released on behalf of process 8675
...
```

Changes (41)

A bin-install target was added, which will install a binary package if available, otherwise it will run a "make package"

Changes (42)

An EVAL_PREFIX= GTKDIR=gtk-[0-9]* definition was added, which will use pkg_info(1) to find out the installed prefix of a package, rather than guessing at ${X11BASE} or ${LOCALBASE}, depending on some current definition

Changes (42) continued

```
.if defined(EVAL_PREFIX)
.  for def in ${EVAL_PREFIX}
.    if !defined(${def:C/=.*//}_DEFAULT)
${def:C/=.*//}_DEFAULT= ${X11PREFIX}
.    endif
.    if !defined(${def:C/=.*//})
_depend_${def:C/=.*//} != ${PKG_INFO} -e ${def:C/.*=//} 2>/dev/null; ${ECHO}
.      if (${_depend_${def:C/=.*//}} == "")
${def:C/=.*//}=${${def:C/=.*//}_DEFAULT}
.      else
_dir_${def:C/=.*//} != (${PKG_INFO} -qp ${def:C/.*=//} 2>/dev/null) | ${AWK} '{ print $$2; exit }'
${def:C/=.*//}=${_dir_${def:C/=.*//}}
MAKEFLAGS+= ${def:C/=.*//}=${_dir_${def:C/=.*//}}
.      endif
.    endif
.  endfor
.endif
```

Changes (43)

The automatic Perl packages create the PLIST for you

Changes (44)

The basis of all PLISTs moved from being a.out-based to ELF-based, and modify the way the derived PLIST is generated, so that the correct files are noted for each object format

Changes (45)

A package was added to generate Solaris packages from an installed package on Solaris

Changes (46)

SVR4_PKGNAME definitions were added across pkgsrc, since Solaris package names can have at most 9 characters

Changes (47)

The contents of the COMMENT files were moved into the package Makefiles

Changes (48)

Support for message digests other than md5 for distfiles and patches was added, by using the digest package, and support was added for SHA256 and SHA512 to the digest package

Changes (49)

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)

From the pkgsrc/audio/trplayer/Makefile on my system:

```
BUILD_DEPENDS+= rpm2pkg-1.2:../../pkgtools/rpm2pkg
DEPENDS+= realplayer>=8.0.1:../../audio/realplayer
DEPENDS+= suse_base>=7.3:../../emulators/${SUSE_DIR_PREFIX}_base
DEPENDS+= suse_compat>=7.3:../../emulators/${SUSE_DIR_PREFIX}_compat
DEPENDS+= suse_libc5>=7.3:../../emulators/${SUSE_DIR_PREFIX}_libc5
DEPENDS+= suse_slang>=7.3:../../emulators/${SUSE_DIR_PREFIX}_slang
```

Changes (50)

The version of the package extracted is saved in the ${EXTRACT_COOKIE}, and checked at installation time that this version matches ${PKGNAME}

Changes (51)

A .USE macro was replaced by normal targets, to stop sub-makes being spawned for the pre-, do- and post-target stages, replacing them with standard make(1) targets

Timing information as follows (multiple runs performed, best results taken):

800 MHz Celeron, 128 MB, local pkgsrc, local obj

```
scripts/, pre,do,post-*:        0.731u 0.261s 0:02.04 48.5%     0+0k 29+168io 9pf+0w
no scripts/, pre,do,post-*:     0.678u 0.242s 0:01.30 70.0%     0+0k 0+169io 0pf+0w
no scripts/, no pre,do,post-*:  0.267u 0.089s 0:00.90 37.7%     0+0k 0+155io 0pf+0w
```

40 MHz Sparc, 36 MB, nfs pkgsrc, local obj

```
scripts/, pre,do,post-*:        22.590u 6.839s 0:33.31 88.3%    0+0k 121+254io 0pf+0w
no scripts/, pre,do,post-*:     22.481u 6.442s 0:33.30 86.8%    0+0k 120+251io 0pf+0w
no scripts/, no pre,do,post-*:   8.534u 4.189s 0:16.48 77.1%    0+0k 105+242io 0pf+0w
```

Changes (52)

Special handling was added for packages which need to install rc.d scripts, create users, and install example files

Changes (53)

We taught bsd.pkg.mk how to extract all files in ${EXTRACT_ONLY} that end in suffices listed in ${_EXTRACT_SUFFICES}. Currently,

```
_EXTRACT_SUFFICES=      .tar.gz .tgz .tar.bz2 .tbz .tar.Z .tar _tar.gz
_EXTRACT_SUFFICES+=     .shar.gz .shar.bz2 .shar.Z .shar
_EXTRACT_SUFFICES+=     .zip
_EXTRACT_SUFFICES+=     .lha .lzh
_EXTRACT_SUFFICES+=     .Z .bz2 .gz
```

Changes (54)

A new framework was introduced for handling info files generation and installation

Changes (55)

A replace target was introduced

This target first makes a binary package of the existing installed package, then a copy of the +REQUIRED_BY file is taken, if it exists, and then the existing package is deleted. The new package is installed, and the preserved +REQUIRED_BY file is copied back into place, using its contents to modify the +CONTENTS files of all the packages which require it. The undo-replace shares code with the replace target, and does the same operation, but in reverse

Changes (56)

The IS_INTERACTIVE definition was deprecated, and a finer-grained INTERACTIVE_STAGE definition was introduced.

INTERACTIVE_STAGE can take any of the values: fetch, configure, build and install. Multiple values are allowed: e.g. INTERACTIVE_STAGE= configure install

Changes (57)

Two knobs were added for packages:
CONFIG_GUESS_OVERRIDE and CONFIG_SUB_OVERRIDE.
Example:

```
CONFIG_GUESS_OVERRIDE=  ${WRKSRC}/config.guess
CONFIG_SUB_OVERRIDE=    ${WRKSRC}/config.sub
```

Just before the bulk of the "configure" phase, the named files will be replaced with symlinks to their canonical pkgsrc versions in pkgsrc/mk/gnu-config. We can add support for new ports (such as the SuperH5 port) to GNU_CONFIGUREd packages easily

Changes (58)

More flexibility in the handling of UNLIMIT_RESOURCES was introduced. Each word of UNLIMIT_RESOURCES is supposed to be a knob on ULIMIT_CMD_<word> variable which value if defined is added to _ULIMIT_CMD. The ULIMIT_CMD_* variables are set per $OPSYS in defs.*.mk and are overridable by the user

pkgsrc/lang/jikes/Makefile

```
# $NetBSD: Makefile,v 1.10 2002/09/29 07:36:49 jlam Exp $
#

DISTNAME= jikes-1.15
CATEGORIES= lang
MASTER_SITES= http://oss.software.ibm.com/pub/jikes/

MAINTAINER= packages@netbsd.org
HOMEPAGE= http://www10.software.ibm.com/developerworks/opensource/jikes/
COMMENT= Java source to byte-code compiler

ONLY_FOR_PLATFORM= NetBSD-*-* SunOS-*-*

USE_BUILDLINK2=  yes
GNU_CONFIGURE=  yes
USE_CXX=  yes
CXXFLAGS+=  ${CFLAGS}
UNLIMIT_RESOURCES= datasize

USE_GMAKE= # uses multi-line comments with \ (naughty hack!)

.include "../../lang/gcc/buildlink2.mk"
.include "../../mk/bsd.pkg.mk"
```

pkgsrc/lang/jikes/Makefile

```
# $NetBSD: Makefile,v 1.25 2004/04/27 23:23:03 recht Exp $
#

DISTNAME= jikes-1.20
CATEGORIES= lang java
MASTER_SITES= http://oss.software.ibm.com/pub/jikes/1.20/
EXTRACT_SUFX= .tar.bz2

MAINTAINER= tech-pkg@NetBSD.org
HOMEPAGE= http://www10.software.ibm.com/developerworks/opensource/jikes/
COMMENT= Java source to byte-code compiler

USE_BUILDLINK3=  yes
GNU_CONFIGURE=  yes
USE_LANGUAGES=  c c++
USE_GCC_SHLIB=  yes
UNLIMIT_RESOURCES=      datasize

USE_GNU_TOOLS+= make # uses multi-line comments with \ (naughty hack!)

.include "../../mk/bsd.pkg.mk"
```

Changes (59)

A script was added to create a sandbox using null mounts, so that bulk builds or other isolated builds can take place without disturbing the currently-installed packages. This allows NetBSD packages to be built for different versions of the operating system from the one which is running

Changes (60)

if PKGSRC_RUN_TEST is yes, "make all" runs tests

Changes (61)

add OS and arch specific MESSAGE file handling

Changes (62)

Use PKG_FAIL_REASON and PKG_SKIP_REASON rather than IGNORE - allows builds to stop when a dependency is broken, yet continue builds when a dependency is merely skipped (usually because it duplicates functionality in the base system).

Changes (63)

redo the README.html target for increased speed. For packages with no dependencies the speedup is about 2x for ones like gnome with lots of dependencies, the speedup is aroud 400x.

Changes (64)

Added PKG_PRESERVE functionality.

A package which has PKG_PRESERVE definied in its Makefile will not be able to be deleted, and the capability is carried into binary packages.

Changes (65)

Add a check at fetch time to see if there are any known vulnerabilities in a package - should keep some admins' blood pressure a bit lower.

Changes (66)

Introduce a new framework to handle info files, install-info and makeinfo commands

- reduce the number of ’@exec’ and ’@unexec’ in PLIST by using INSTALL/DEINSTALL scripts to handle entries’ Info file addition and removal
- achieve lighter dependencies by avoiding unnecessary run-time dependency on the gtexinfo package

Changes (67)

As part of the build information, record the full pathnames of the shared object "provides" and "requires" information. This is only turned on just now if ${CHECK_SHLIBS} is set to "YES"

Example output:

```
$ pkg_info -B libutf | grep '^PROVIDES'
PROVIDES=/usr/pkg/lib/libutf.so.2
$ pkg_info -B ssam | grep '^REQUIRES'
REQUIRES=/usr/lib/libc.so.12
REQUIRES=/usr/pkg/lib/libutf.so.2
$
```

Changes (68)

Handle platforms with broken tools in the base system, such as sed and awk. As proposed on tech-pkg@, with some changes to set the appropriate tool variables and handle OSs which provide GNU tools in the base system (ie. do nothing)

This allows packages or users to force the use of pkgsrc GNU tools when they are not present in the base system by defining e.g. USE_GNU_TOOLS="awk sed".

Changes (69)

Introduce a PKGSRC_MESSAGE_RECIPIENTS, which takes the login names of users to whom the MESSAGE file should be mailed at package installation time, and mail the MESSAGE file at the "make install" stage (if PKGSRC_MESSAGE_RECIPIENTS is not empty).

Changes (70)

Merge pkgviews-mk branch into the HEAD

Changes (71)

In cases where we need the best match for a pkgpattern, use "${PKG_BEST_EXIST} pkgpattern" instead of "${PKG_INFO} -e pkgpattern". The latter can return multiple package names if there are multiple versions of a piece of software installed. PKG_BEST_EXIST is defined to be "${PKG_ADMIN} -b -d ${_PKG_DBDIR} -s "" lsbest", so it searches for the best installed package that matches the given pkgpattern or else returns the empty string.

Changes (72)

Deprecate Zoularis: remove any tests for ZOULARIS* and bomb if ${LOCALBASE}/bsd/share/mk/zoularis.mk exists.

Changes (73)

Support DEPENDS_TARGET="install clean"

Changes (74)

Add a new install macro INSTALL_LIB for use when installing libraries (mainly intended for shlib use, but for homeful use on all libraries so that currently static libs can be "provisioned" for future shlib use)

Analysis of Changes

All of the NetBSD users I know use pkgsrc - this is a huge benefit to us

- new users point out inconsistencies, imperfections, and places where we do not come up to scratch
- existing users say to us "it would be nice if..."
- people running -current and release branches of NetBSD test building and packaging
- people running foreign operating systems test building and packaging

In all, "we eat our own food - if it’s no good, we know"

Buildlink

Buildlink functionality has added a new dimension to pkgsrc, in that we can now be sure that we get packages built with the correct software, and the correct version of that software

Buildlink1 - what is it?

The buildlink1 functionality in pkgsrc has two purposes:

(1) Cause all headers and libraries used by a particular package to be found in a known location during the configure and build process. These packages are said to be "weakly-buildlinked"

(2) Cause ONLY those headers and libraries used by a particular package to be found during the configure and build process. These packages are said to be "strongly-buildlinked"

Firstly, let’s look at "buildlink1"

How Buildlink1 Works (1)

Goal (1) is accomplished by simply including the buildlink.mk file of a dependency in the package's Makefile, which

- Adds a DEPENDS or BUILD_DEPENDS line for the package
- Creates a directory ${BUILDLINK_DIR}, by default set to a subdirectory of ${WRKDIR}
- Links all the headers and libraries for that dependency into ${BUILDLINK_DIR}/include and ${BUILDLINK_DIR}/lib, respectively

How Buildlink1 Works (2)

- Prepends -I${BUILDLINK_DIR}/include to CPPFLAGS, CFLAGS, CXXFLAGS, and -L${BUILDLINK_DIR}/lib to LDFLAGS

- Creates a wrapper script for GTK+-style config scripts, often found in GNOME software, that translates

  -I${LOCALBASE}/include and -L${LOCALBASE}/lib into

  references into ${BUILDLINK_DIR}

How Buildlink1 Works (3)

- Some packages are for software libraries whose functionality is a part of recent released versions of the host operating system, e.g. readline, OpenSSL, and ncurses
- For those packages, the buildlink.mk files link the appropriate system headers and libraries into ${BUILDLINK_DIR} so that goal (1) is still met
- Where possible, the system headers and libraries are renamed when linked into ${BUILDLINK_DIR} to match the names of their pkgsrc counterparts so that the files may be referenced under a consistent name

How Buildlink1 Works (4)

- Goal (2) requires some work on the part of the package creator
- As all headers and libraries used by a package may be found in ${BUILDLINK_DIR}, and -I${BUILDLINK_DIR}/include and -L${BUILDLINK_DIR}/lib are already passed to the compiler, it is no longer necessary to pass -I${LOCALBASE}/include or -L${LOCALBASE}/lib to the compiler
- Those lines should be removed from package Makefiles, and where necessary, the package sources should be patched to do the same

Buildlink - Problems

The buildlink framework tries to do its work "up-front" before the configure process, and "fix things up" after the build process

Over time, the buildlink framework grew overly complex to deal with software that stored build-time information in the installed files, e.g. GNOME packages

buildlink does not scale well - the command line for buildlinked packages can grow to huge sizes

Buildlink2 - How it works

The buildlink2 framework is a departure from the original buildlink framework, which tries to do its work up-front before the configure process, and fix things up after the build process

The new framework actually does its work as the sotfware is being configured and built through a collection of wrapper scripts that are used in place of the normal compiler tools

We still symlink libraries and headers into ${BUILDLINK_DIR} to normalize the environment in which the software is built, but now we tell the configure process the actual installed locations of the libraries and headers we are using, and the compiler wrappers will munge them into references into ${BUILDLINK_DIR}

Benefits of buildlink2

- The new framework makes it simpler to buildinkify a package because we just convert dependencies into including the equivalent buildlink2.mk files and define USE_BUILDLINK2_ONLY. We don’t need to lie about where libraries or headers we use are installed
- All packages using the new framework are strongly buildlinked; it is not possible to create weakly buildlinked packages. This deprecates the need for x11.buildlink.mk
- We no longer care if the configure or build process adds -I${PREFIX}/include or -L${PREFIX}/lib to the compiler or link lines. We WANT them to do so (and we actually add them ourselves) since they are munged into references to ${BUILDLINK_DIR) by the wrapper scripts

Benefits of buildlink2 (2)

- We no longer need to create and use config script wrappers
- buildlink2.mk files now simply create the <pkg>-buildlink target and can discard the REPLACE_BUILDLINK and *CONFIG_WRAPPER* lines
- We no longer mess around with configure scripts or Makefiles before the build process, so we don’t accidentally trigger rebuilds of those files if the software uses GNU autoconf/automake

Buildlink and buildlink2's co-existence

The buildlink and buildlink2 frameworks can coexist within pkgsrc, but packages that use the new framework must use it exclusively, i.e. a package Makefile can't include both buildlink.mk and buildlink2.mk files. Packages that use the old framework can continue to do so, but it is encouraged that they convert to the new buildlink2 framework for the benefits listed earlier

Buildlink and imake

Packages that use imake to drive the configuration and build processes can now be buildlink2-ed as well

Buildlink and compilers

Compilers other than the system-supplied cc, such as CC=/my/special/c-compiler in /etc/mk.conf, should DTRT

The wrapper scripts automatically handle this situation. The software is told to use CC=cc, which points to the special compiler wrapper script in ${BUILDLINK_DIR}/bin/cc, but the wrapper itself will call the CC that you explicitly set

Buildlink2 execution timing

The full build with buildlink2 now takes longer than it used to. Since we are using wrapper scripts in place of the compilers, we bear the cost of the extra shell processes invoked as a result. The increased build times on the two platforms on which I was able to test are roughly:

NetBSD-1.5ZC/i386	+3% (non-USE_LIBTOOL)
NetBSD-1.5ZC/i386	+5% more (USE_LIBTOOL)
NetBSD-1.5.1/mac68k	+9% more (USE_LIBTOOL)

The i386 box is an Intel PIII 850MHz + UDMA IDE HD + 512MB SDRAM
The mac68k box is a Quadra 650 (68040) + SCSI2 HD + 48MB RAM

Onto Package Views

Now that we know how pkgsrc differs from the other packaging systems in use in *BSD, we can move onto Package Views

The story up until now

The conventional *BSD ways of installing software (NetBSD's pkgsrc, FreeBSD/OpenBSD ports system) install directly into ${LOCALBASE}, possibly overwriting existing files

- There can only be one version of a piece of software installed at any one time
- 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
- Problems can arise when some 3rd party software is upgraded, and a lot of other software depends upon it (libpng, jpeg, zlib)

Various attempts have been made to work around this, but none of these address the fundamental problem - overwriting files

Different approaches

It is desirable to have a means whereby two packages with the same file system entries can co-exist

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

Other approaches

"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 unchanged

OpenBSD’s staged installation approach, similar to Debian’s, will only allow one version of a package to be installed at any one time

CMU’s depot software, GNU’s stow program, and various other packaging efforts (http://www.encap.org/) use a tiered approach to the installation of software

"make replace" works well, except when a shared library major number is bumped

It was never intended to be put into production use, for example

Early efforts

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

The "destdir" approach

- where 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

The "wrapper script" approach

- 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

Internally within bsd.pkg.mk

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

Analysis of early efforts

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

The Aims of Package Views

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:

- to allow any number of different versions of packages to co-exist at any one time
- to allow the testing of different versions of packages on a single machine at any one time
- to allow more dynamic conflict detection at install time
- whilst continuing to use the existing pkg_install tools

Package Views

The basic idea of package views is that a tiered approach is used, which is 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

Tenet

"Every problem in Computing Science can be solved with another layer of indirection"

Hierarchies

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

Dynamic PLISTs

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

PLIST manipulations

- 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

Benefits of Dynamic PLISTs

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

Practical Aspects of Package Views

- A package’s files are always in one, canonical location, the depot directory
- There is a default view, which is the null view, and defaults to ${LOCALBASE}
- Any package which wants to link a shared object from another package should use the default view
- Any number of views can be added
- The traditional pkg_install(1) tools are used, with the addition of the script to manage the symbolic link farms

${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"

${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 use the ${PREFIX} definition to refer to the depot directory, ${LOCALBASE}/packages/${PKGNAME}

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, PREFIX is modified to point to

${LOCALBASE}/packages/${PKGNAME}

and no further internal manipulation of prefices needs to take place

Upgrading packages

Previously, an upgrade or update to a package, especially one containing shared libraries and objects, could be an onerous task, made worse (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)

Shared library minor version number change

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

Reverting to older versions

If the newer version of the package does 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

Shared library major version number change

Using the existing "overwrite" mechanism, 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 "use" them as a pre-requisite, 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:

Overwriting symbolic links

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

An illustration - the depot directory

```
[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 >
```

An Illustration - the default view

```
[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 >
```

The Linkfarms

```
[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 >
```

Metadata files in the depot directory

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

The linkfarm

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

Manipulating the linkfarm

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

The +VIEWS file

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

The default view

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

Other views

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

Selecting the view

The user (not necessarily the administrator) chooses which packages within which views are to be used by selecting a path accordingly. The MANPATH should also be selected accordingly

No other manipulation or scripts are necessary

xpkgwedge

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

Existing views

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

Co-existence

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

Building and Linking

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

We could use "hard" links

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

We can’t use "hard" links

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

Disadvantages

- Some people think that the linkfarms are unruly, unsightly and ugly
- A minimal amount of extra space is used to provide the linkfarms

Disadvantages

- Some people think that the linkfarms are unruly, unsightly and ugly ("So do I. So what?")
- A minimal amount of extra space is used to provide the linkfarms

Disk space

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)

Aims

- to allow any number of different versions of packages to co-exist at any one time
- to allow the testing of different versions of packages on a single machine at any one time
- to allow more dynamic conflict detection at install time
- whilst continuing to use the existing pkg_install tools, and
- to provide support for dynamic packing lists

Unexpected Advantages

- immediately obvious to which package a file or directory belongs
- many additional views can be built up
- pkg_delete(1) deletes links in the views as well as the package itself
- multiple conflicting packages can be installed at the same time

More Advantages

- development packages can be tested and evaluated on the same machine on which they will eventually run
- portable to any system on which pkgsrc runs - NetBSD, FreeBSD, OpenBSD, Solaris (2.6, 2.7, 2.8 and 2.9), Darwin, Linux. Irix, Digital Unix and HP/UX are currently in the works
- scalable in practice (see papers on http://www.infrastructure.org/), and from experience of other administrators using the same packaging system

Migrating to Package Views

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

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 availability of dynamic packing lists will simplify pkgsrc entry creation for everyone
- Conflict resolution is not fatal
- The existing package tools can continue to be used
- The symbolic link farms, whilst ugly, give an immediate idea of the package to which a file entry belongs
- A small increase in disk real-estate

The utility value of the advantages far outweigh the disadvantages

Future work

- At present, package views are implemented in pkgsrc in the trunk of the NetBSD CVS repository
- xpkgwedge has been made the default for pkgsrc on all platforms
- definitions have been added to pkgsrc, and individual packages are being converted to be pkgviews friendly
- pkgviews has a goal of being implemented in the pkgsrc-2004Q3 branch

Future work (continued)

- Move to dynamic PLISTs in pkgsrc, by setting the PLIST_TYPE definition to dynamic. The default value for PLIST_TYPE is static
- Monitor reaction to package views and dynamic PLISTs, and to improve upon it where possible

libc and package views

NetBSD still uses a major version of 12 for its libc. This is mandated, really, by ELF shared object constraints, and the need to keep backwards compatibility with pre-compiled third party binaries. We are looking at system packages, combined with package views to be able to bump libc’s major number

The End

Alistair G. Crooks
agc@pkgsrc.org
agc@netbsd.org
The NetBSD Project

self@alistaircrooks.com
Personal

agc@wasabisystems.com
Work

Fri May 7 20:42:18 BST 2004

Any Questions?

Alistair G. Crooks
agc@pkgsrc.org
agc@netbsd.org
The NetBSD Project

self@alistaircrooks.com
Personal

agc@wasabisystems.com
Work

Fri May 7 20:42:18 BST 2004