– pkgsrc –

The NetBSD Packages Collection

Jan Schaumann

`jschauma@netbsd.org`

`136D 027F DC29 8402 7B42  47D6 7C5B 64AF AF22 6A4C`

Topics covered

- Concept of Software Package Management
- Features of pkgsrc
- Using pkgsrc
  - Using binary packages
  - Building packages from source
  - Updating packages
  - Deploying large numbers of binary packages
- A look ahead

General Concepts: File System Hierarchy

Basic distinction between *shareable* and *unshareable* files.

Shareable content:

- essential components that remain the same across multiple hosts
- often possible to mount read-only
- may overlap with *local* data

Unshareable content:

- data that needs to be individual to each host
- data that changes often
- may overlap with *local* data

General Concepts: File System Hierarchy

Examples:

	shareable content	unshareable content
static		
variable		

General Concepts: File System Hierarchy

Examples:

	shareable content	unshareable content
static	`/usr` `/opt`	
variable		

General Concepts: File System Hierarchy

Examples:

	shareable content	unshareable content
static	`/usr` `/opt`	`/etc` `/boot`
variable		

General Concepts: File System Hierarchy

Examples:

	shareable content	unshareable content
static	`/usr` `/opt`	`/etc` `/boot`
variable	`/var/mail` `/var/spool/news`	

General Concepts: File System Hierarchy

Examples:

	shareable content	unshareable content
static	`/usr` `/opt`	`/etc` `/boot`
variable	`/var/mail` `/var/spool/news`	`/var/run` `/var/lock`

General Concepts: System Software vs. Third Party Software

Consider:

- OS upgrades vs. software upgrades
- configuration files get mixed up
- duplicates or conflicting versions in the base system vs. the add-ons
- startup scripts, dæmons
- location of third party software
- proprietary third party software

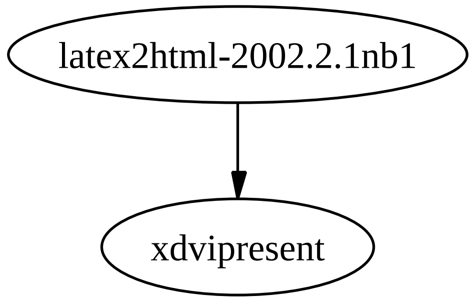
General Concepts: Why use a Package Management System?

Sure, you *could* install everything by hand. But ...

- thousands of applications and libraries are available
- binaries are not available for all platforms
- compilation from source is not always trivial
  - “90% of everything is crud”
  - platform and compiler issues may arise
  - different ways of satisfying dependencies / optional features
- how to maintain a database of installed applications?
- dependencies among applications and libraries grow rather complex

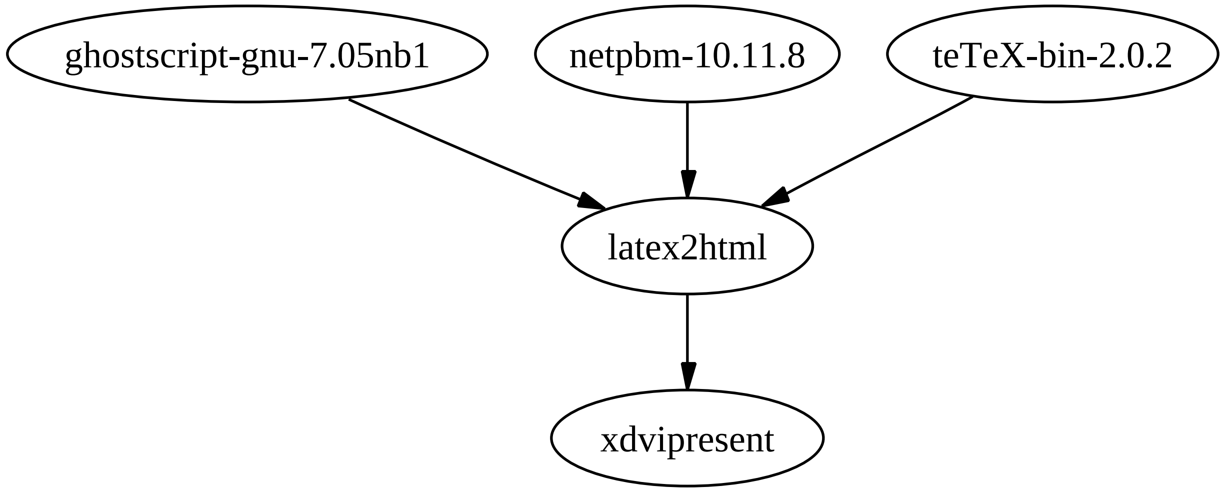
General Concepts: Complexity of Dependencies

An example: Let's install some software to produce nice slides!



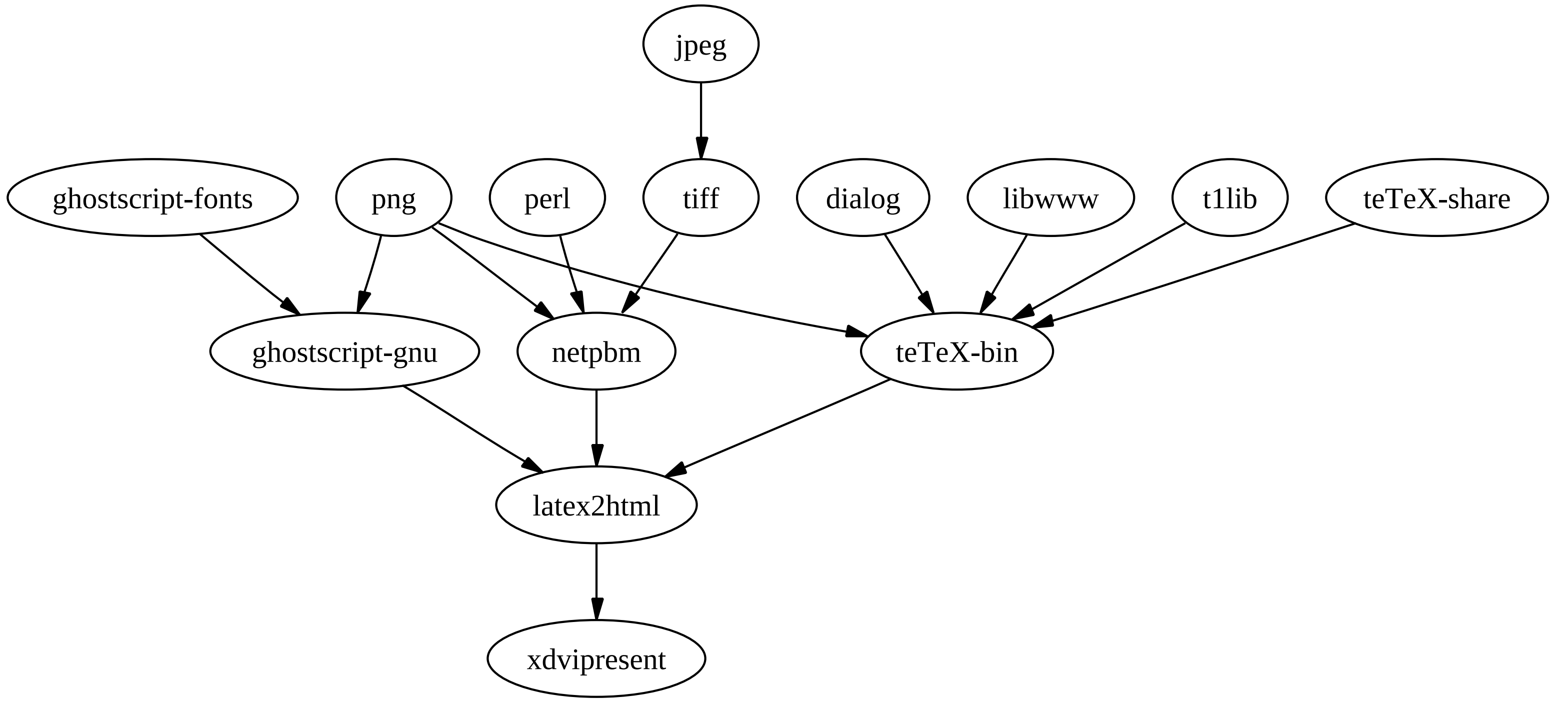
General Concepts: Complexity of Dependencies

An example: Let’s install some software to produce nice slides!



General Concepts: Complexity of Dependencies

An example: Let’s install some software to produce nice slides!



General Concepts: Package Management

Important aspects of package management:

- available for virtually all OS
- may include building from source as well as binary package management
- easily determine what is installed
- keep track of *where* things are installed
- make *consistent* use of the package manager
- consider providing separate *“depots”* or *“views”* for different purposes
- teach users to use the tools

Enter pkgsrc...

```
$ wtf pkgsrc
Gee...  I don't know what pkgsrc means...
$
```

Enter pkgsrc...

```
$ wtf pkgsrc
Gee...  I don't know what pkgsrc means...
$
```

“The NetBSD Packages Collection (pkgsrc) is a framework for building third-party software on NetBSD and other UNIX-like systems, currently containing nearly 5300 packages. It is used to enable freely available software to be configured and built easily on supported platforms.”

Platforms supported by pkgsrc

- NetBSD (1997)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)
- BSD/OS (2003)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)
- BSD/OS (2003)
- AIX (2003)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)
- BSD/OS (2003)
- AIX (2003)
- Interix – Microsoft Windows Services for Unix (2004)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)
- BSD/OS (2003)
- AIX (2003)
- Interix – Microsoft Windows Services for Unix (2004)
- DragonFlyBSD (2004)

Platforms supported by pkgsrc

- NetBSD (1997)
- Solaris – first non-NetBSD platform supported, aka “Zoularis” (1999)
- Linux (1999)
- Darwin / Mac OS X (2001)
- FreeBSD (2002)
- OpenBSD (2002)
- IRIX (2002)
- BSD/OS (2003)
- AIX (2003)
- Interix – Microsoft Windows Services for Unix (2004)
- DragonFlyBSD (2004)
- OSF/1 (2004)

pkgsrc Terminology

“pkgsrc” – the NetBSD Packages Collection (ie the package management system)

“port” – In NetBSD terminology, *“port”* refers to a different architecture; FreeBSD and OpenBSD refer to what we call a *“package”* with the word *“port”*.

“package” – A set of files and building instructions that describe what’s necessary to build a certain piece of software using pkgsrc. Packages are traditionally stored under `/usr/pkgsrc`.

“binary package” – A set of binaries built with pkgsrc from a distfile and stuffed together in a single `.tgz` file so it can be installed on machines of the same machine architecture without the need to recompile.

“distfile” – The file(s) that are provided by the author of the piece of software to distribute his or her work. All the changes necessary to build on NetBSD are reflected in the corresponding *package*. Distfiles are usually fetched into /usr/pkgsrc/distfiles.

Features of pkgsrc

Some of the more obvious features:

- fetching of *distfiles*
- dependency checking
- installation of the *package* and all dependencies
- maintenance of package database, including checksums of all files
- upgrade procedures:
  - automatically upgrade all installed packages
  - automatically upgrade a single package and all dependents
  - upgrade a package in place (careful!)
- all of the above applies equally to *binary packages* as well as building from source

Features of pkgsrc

Some more subtle features:

- integrity checking via recorded checksum
- automatic patching of sources where necessary
- verification of acceptable licenses
- central configuration file to control pkgsrc behavior (`/etc/mk.conf` – lots of knobs! See `pkgsrc/mk/defaults/mk.conf`)
- checking of installed packages against known vulnerabilities
- pkgsrc release engineering – `HEAD` and quarterly stable branches

Using pkgsrc: Getting Started

There are three ways to get pkgsrc:

- via *sup*: `release=pkgsrc`
- via *cvs*: `cvs -d anoncvs@anoncvs.nebsd.org:/cvsroot co -P pkgsrc`
- via *ftp*:
  `ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-current/tar_files/pkgsrc.tar.gz`

On non-NetBSD platforms, you also need the necessary tools:

- via binary bootstrap kits available from
  `ftp://ftp.netbsd.org/pub/NetBSD/packages/bootstrap-pkgsrc/`
- via pkgsrc itself:

```
# cd /usr/pkgsrc/bootstrap
# more README
# ./bootstrap
```

Using pkgsrc: Using binary packages

Using `pkg_add(1)`:

Precompiled packages are stored on ftp.netbsd.org and its mirrors in the directory `/pub/NetBSD/packages`.

Installation is as easy as

```
# pkg_add \\
ftp://ftp.netbsd.org/pub/NetBSD/packages/`uname -r`/`uname -p`/All/package.tgz
```

Influenced by (among others):

- `LOCALBASE` – place where all packages are installed; defaults to `/usr/pkg`
- `PKG_DBDIR` – place where information about all packages is stored; defaults to `/var/db/pkg`
- `PKG_PATH` – list of directories in which to look for binary packages; defaults to `/usr/pkgsrc/packages/All`.

Using pkgsrc: Building packages from source

Package components:

- `Makefile` – instructions on how to download and build the software
- `distinfo` – checksum of the *distfiles*
- `patches/*` – any patches necessary to make the software build
- `DESCR` – a description of the package
- `PLIST` – the list of files that are installed
- a number of other optional files

Using pkgsrc: Building packages from source

A simple package: *net/tor*

```
DISTNAME=           tor-0.0.9.3
CATEGORIES=         net security
MASTER_SITES=       http://tor.eff.org/dist/

MAINTAINER=         jschauma@NetBSD.org
HOMEPAGE=           http://tor.eff.org
COMMENT=            Anonymizing overlay network for TCP

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

.if !empty(PKGSRC_COMPILER:Mmipspro)
CFLAGS+=            -c99
.endif

do-install:
    ${INSTALL_PROGRAM} ${WRKSRC}/src/or/tor ${PREFIX}/bin/tor
    ${INSTALL_SCRIPT} ${WRKSRC}/contrib/torify ${PREFIX}/bin/torify
    ${INSTALL_MAN} ${WRKSRC}/doc/tor.1 ${PREFIX}/man/man1/tor.1
    ${INSTALL_MAN} ${WRKSRC}/contrib/torify.1 ${PREFIX}/man/man1/torify.1

.include "../../security/openssl/buildlink3.mk"
.include "../../mk/bsd.pkg.mk"
```

Using pkgsrc: Building packages from source

```
# cd /usr/pkgsrc/net/tor
# make install
```

Easy! But what exactly happens here?

Using pkgsrc: Building packages from source

```
# cd /usr/pkgsrc/net/tor
# make install
```

Easy! But what exactly happens here?

`make` performs the following steps:

1. fetch *distfile*
2. make sure checksum matches
3. install dependencies (repeating these steps for each)
4. extract *distfile*
5. apply any patches (if necessary)
6. configure software
7. build software
8. install software

Using pkgsrc: Creating binary packages

```
# cd /usr/pkgsrc/net/tor
# make package
```

Easy! But what exactly happens here?

Using pkgsrc: Creating binary packages

```
# cd /usr/pkgsrc/net/tor
# make package
```

Easy! But what exactly happens here?

Not all that complicated:

- `make install`
- create binary package by including all the files and the meta-information in a tarball

Meta-information includes:

- build information, such as `OPSYS`, `OS_VERSION`, `CFLAGS`, etc.
- a one-line comment
- contents of the package, including checksums of all files; list of dependencies
- a description of the package

Using pkgsrc: Updating packages

Holy Grail: During the upgrade process – which doesn't affect the running system – all dependents are upgraded as well (*iff* necessary).

In reality, this can get hairy!

Possibilities in pkgsrc:

- delete package, rebuild and reinstall
- replace package without rebuilding dependencies
- delete package, remembering all dependents, then rebuild them all

Using pkgsrc: Updating packages

Delete package, rebuild and reinstall:

```
# make deinstall clean install
```

or

```
# pkg_delete <pkg-old> && pkg_add <pkg-new>
```

- most simple approach
- works well for *leaf* packages
- well, works **only** for *leaf* packages

Using pkgsrc: Updating packages

Replace package without rebuilding dependencies:

```
# make replace
```

or

```
# pkg_add -u <pkg>
```

- register all dependents
- (force) delete existing package
- install new package
- register / update all dependents

- Problem: dependents may not work (correctly) with new version

Using pkgsrc: Updating packages

Delete package, remembering all dependents, then rebuild them all:

```
# make update
```

Note that this is the *only* way to guarantee that all packages will work with each other!

Problems:

- during rebuild process, the system is in unstable condition
- what if rebuild of one of the packages fails?
- highly “destructive” for upgrades of essential packages (such as libpng)

Solution: use a sandbox/chroot or another machine to build binary packages.

Using pkgsrc: Updating packages

Delete package, remembering all dependents, then rebuild them all.

Using binary packages:

```
# pkg_info > /tmp/before
# pkg_delete -r <pkg>
# pkg_info > /tmp/after
# diff /tmp/before /tmp/after
# pkg_add <each> <deleted> <pkg>
```

But this requires you to have binary packages available!
How do we do that?

Using pkgsrc: Updating packages

Delete package, remembering all dependents, then rebuild them all.

Using binary packages:

```
# pkg_info > /tmp/before
# pkg_delete -r <pkg>
# pkg_info > /tmp/after
# diff /tmp/before /tmp/after
# pkg_add <each> <deleted> <pkg>
```

But this requires you to have binary packages available!
How do we do that?

- download all required packages from ftp.netbsd.org
- (regularly) bulk-build all packages yourself
- (regularly) build only the packages you actually need yourself

Using pkgsrc: bulk-building all packages

See `/usr/pkgsrc/mk/bulk/*`

- create `build.conf` and point `$BULK_BUILD_CONF` to it
- `# cd /usr/pkgsrc`
- `# sh mk/bulk/build`
- read a good book

Warning: During the bulk build, *all packages will be removed!*

Using pkgsrc: bulk-building all packages

What you need:

- lots of RAM and CPU power
- ~ 10 GB diskspace for *distfiles*
- ~ 8 GB diskspace for all resulting binary packages
- ~ 5 GB diskspace to extract and build everything

If you do not have a constant internet connection, then you can fetch all required distfiles at once:

```
# cd /usr/pkgsrc
# make fetch-list | sh
```

Using pkgsrc: bulk-building all packages

The bulk builds consist of three steps:

1. pre-build:
   The script updates your pkgsrc tree via (anon)cvs, then cleans out any broken distfiles, and removes all packages installed.
2. the bulk build:
   This is basically `make bulk-package` with an optimized order in which packages will be built. Packages that don't require other packages will be built first, and packages with many dependencies will be built later.
3. post-build:
   Generates a report that's placed in the directory specified in the build.conf file named broken.html, a short version of that report will also be mailed to the build's admin.

Instructions on how to do this in a sandbox are available online.

Using pkgsrc: bulk-building select packages

Using `pkg_comp(1)`:

- automatically sets up and operates inside a chroot
- allows easy built of packages for other system versions
- specify which packages to build by setting `BUILD_PACKAGES`

Using `pkg_chk(1)`:

- automatically updates packages to latest pkgsrc version
- easily build sets of packages for different hosts

Using pkgsrc: Security Audits

`security/audit-packages` – show vulnerabilities in installed packages

`pkg-vulnerabilities`, maintained by NetBSD Packages Team:

```
# package                      type of exploit         URL
[...]
mysql-client-4.1.[0-8]{,nb*}   local-file-write        http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0004
ethereal<0.10.9                remote-code-execution   http://www.ethereal.com/news/item_20050120_01.html
koffice<1.3.5nb4               remote-code-execution   http://www.kde.org/info/security/advisory-20050120-1.txt
squid<2.5.7nb6                 buffer-overrun          http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0094
unarj<2.65nb1                  local-file-write        http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1027
suse_libtiff<9.1nb1            remote-code-execution   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1308
suse_x11<9.1nb1                remote-code-execution   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0914
suse_gtk2<9.1nb3               denial-of-service       http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0788
webmin<1.160                   local-file-write        http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0559
sun-jre14<2.6                  remote-code-execution   http://sunsolve.sun.com/search/document.do?assetkey=1-26-57708-1
sun-jdk14<2.6                  remote-code-execution   http://sunsolve.sun.com/search/document.do?assetkey=1-26-57708-1
sun-jre13<1.0.13               remote-code-execution   http://sunsolve.sun.com/search/document.do?assetkey=1-26-57708-1
sun-jdk13<1.0.13               remote-code-execution   http://sunsolve.sun.com/search/document.do?assetkey=1-26-57708-1
evolution14<1.4.6nb3           remote-code-execution   http://www.gentoo.org/security/en/glsa/glsa-200501-35.xml
evolution<2.0.3nb1             remote-code-execution   http://www.gentoo.org/security/en/glsa/glsa-200501-35.xml
enscript<1.6.5                 remote-code-execution   http://www.securityfocus.org/advisories/7879
bind-9.3.0                     denial-of-service       http://www.kb.cert.org/vuls/id/938617
p5-DBI<1.46nb2                 local-file-write        http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0077
f2c<20001205nb8                local-file-write        http://www.debian.org/security/2005/dsa-661
[...]
```

Using pkgsrc: Other nifty tools

Useful packages:

- `pkgtools/pkgdepgraph` – visual representation of installed packages
- `pkgtools/pkgfind` – find packages by package name in pkgsrc
- `pkgtools/pkglint` – check a pkgsrc entry for correctness (`pkglint(1)`); check for and remove out-of-date distfiles and binary packages (`lintpkgsrc(1)`)
- `pkgtools/pkg_tarup` – create a binary package from an already-installed package.

Useful variables:

- `IGNORE_RECOMMENDED=YES`
- `DEPENDS_TARGET=package`
- `MASTER_SITE_OVERRIDE=ftp://your.mirror/pub/NetBSD/packages/distfiles/`

Useful targets:

- `make show-all-depends-dirs`
- `make show-var VARNAME=<VAR>`

pkgsrc – what's next?

Areas in which work is being done / ideas that have been suggested:

- using and bulk-building as non-root
- automated PLIST handling
- cross-building packages for slower architectures
- `pkg_select` – a curses based pkgsrc browser / manager
- integration with `sysinst`/`sushi(8)`
- parallelizing bulk-builds
- extend `audit-packages` to include a `pkg_ids`

How you can help

- join `tech-pkg@netbsd.org`
- join *pkgsrc-wip* on SourceForge
- create new packages
- update existing packages
- look at PR database and submit patches
- port pkgsrc to other operating systems
- `/usr/pkgsrc/doc/TODO`

References

http://www.pkgsrc.org/
http://www.netbsd.org/
http://www.netbsd.org/Documentation/pkgsrc/
http://pkgsrc-wip.sourceforge.net/
http://www.netbsd.org/Gnats/category/pkg.html

pkg_add(1), pkg_info(1), pkg_delete(1), packages(7)