Design, Implementation and Operation of NetBSD Base System Packaging

Yuuki Enomoto* Ken'ichi Fukamachi†

Abstract

It is believed that UNIX operating system (OS) built on fine granular small parts is preferable to one built on the traditional large tarballs in order to support speedy security update, easy replacement and rollback of specific parts. In Linux distributions, the system are already divided into many small packages. On the other hand, BSD Unix variants are behind the curve on the base system packaging. To improve NetBSD base system granularity, we propose a framework for OS base system packaging. We have developed a software "`basepkg`" by making the best use of pkgsrc framework and operate an experimental base package distribution server to evaluate our software in realistic environment. It is shown that replacement of a few OS granular parts is clearly faster and can provide extra useful functions for NetBSD users and customers.

Key words: Unix, NetBSD, Open Source Software, System Management

1 Background

Historically operating system (OS) has been managed on one source tree and the source tree set has been distributed.

In this quarter century, either of a large or small archive or the combination is used for OS distribution.

*Chitose Institute of Science and Technology, e-mail: mail@e-yuuki.org

†Chitose Institute of Science and Technology, e-mail: k-fukama@photon.chitose.ac.jp

"tarball" (which extension is known as ".tgz") is one of typical large archive formats. BSD UNIX distributes the base system as a set of tarballs where the term "base system" implies a set of programs officially maintained and distributed by the project. In almost cases, OS base system distribution is divided by roles to a set of tarballs such as "base.tgz" (mandatory for the operating system), "comp.tgz" (compiler tools), "man.tgz" (manual) and so on.

There is another classification such as the base system or 3rd party software. Historically BSD UNIX considers that the system consists of the following two categories: (1) the base system built from the official source tree and (2) 3rd party software not contained on the source tree. Usually the latter 3rd party software are managed as a set of small archives called as "package" where this "package" implies a container which consists of software, documentation, configuration files and this package's meta data required to operate in installation and de-installation. Each package role, format and manager differs from one Unix OS to another. Table 1 shows a list of OS, package format and package manager.

Historically BSD Unix has been developed in its own source tree including kernel, general command-

name	format	manager
FreeBSD	txz	pkg
NetBSD	tgz	pkg_install
Debian	deb	apt
Red Hat	rpm	yum
openSUSE	rpm	zypper

Table 1: List of OS, package format and the manager

s, configuration files, and manuals. “Linux” distribution is the opposite. What we call “Linux” was released as just a kernel with a few core programs. Accidentally, Linux distributions needed to assemble a lot of system utilities in order to build a whole Unix clone system. For that reason, the base system management based on a lot of small packages was inevitable and a good idea for Linux distributions.

Major Linux distributions such as Debian and Red Hat Enterprise Linux are already divided into many small packages. These OS's can manage both its own base system and third-party software through its package manager.

On the other hand, BSD Unix such as FreeBSD and NetBSD have each package framework e.g. `ports`(7) and `pkgsrc`(7), but they have been used only for third-party software management.

However today, for users and customers, it is better that OS can be assembled on a lot of small parts easily added or removed. It is suitable especially for rapid security update, easy replacement and rollback of specific parts. In this paper we call this granular base system building “base system packaging”.

To implement a granular NetBSD base system with features mentioned above, we have developed a base system packaging utility “basepkg” for NetBSD.

The rest of this paper is organized as follows. In Chapter 2, we review idea and technique for software packaging on UNIX. In Chapter 3, we describe `basepkg` usage and the internals to show how to write a sustainable shell program by making the best use of pkgsrc framework. We operate a base package distribution server experimentally and estimate the processing speed. In Chapter 4, we discuss several issues to resolve in our system.

2 Packages in BSD UNIX

As mentioned above, BSD UNIX consists of the base system and optional 3rd party software not distributed within the base system. The 3rd party software are called `ports`(7) on FreeBSD and OpenBSD, and `pkgsrc`(7) on NetBSD. We can use `pkgsrc`(7) on a lot of platforms[1].

[1] `http://www.pkgsrc.org/#platforms`

+COMPACT_MANIFEST	meta-data, JSON format a subset of +MANIFEST
+MANIFEST	meta-data, JSON format includes the whole information
bin/hangman	binary

Table 2: Content of FreeBSD `ports`(7) package `hangman-0.9.2_12.txz`.

In this section, we briefly summarize both 3rd party and base package system on FreeBSD and NetBSD since the technical details are referenced in the latter section.

2.1 Packages for 3rd Party Software

2.1.1 FreeBSD `ports(7)`

We review FreeBSD `ports`(7) briefly since FreeBSD `ports`(7) system is the ancestor of NetBSD `pkgsrc`(7).

FreeBSD `ports`(7) is 3rd party software management framework for “installing from source, and packages, for installing from pre-built binaries”[2]. `make`(1) command is used to build a package. The package consists of meta-data, compiled binaries, configuration files and so on. Table 2 shows the content of “hangman” package. We can also use `pkg`(8) command to manage packages.

2.1.2 NetBSD `pkgsrc(7)`

NetBSD `pkgsrc`(7) is a “ framework for building and maintaining third-party software on NetBSD and other UNIX-like systems.”[3]. Initially `pkgsrc`(7) was a spin-off of FreeBSD `ports`(7). Hence, the fundamental usage of pkgsrc is similar to one of FreeBSD `ports`(7).

Consider an example of “openssh” (`pkgsrc/security/openssh`) installation. To build an “openssh” package, you run `make package` in `pkgsrc/security/openssh` directory to generate the package `openssh-7.5.1nb1.tgz` in `pkgsrc/packages/All` directory. The package consists of the meta-data and the program content. Figure 1 shows the content of `openssh-7.5.1nb1.tgz`. The

```
openssh-7.5.1nb1.tgz/
    +CONTENTS
    +COMMENT
    +DESC
    +INSTALL
    +DEINSTALL
    +DISPLAY
    +BUILD_VERSION
    +BUILD_INFO
    +SIZE_PKG
    +SIZE_ALL
    bin/
        scp
        sftp
        ssh
    ...snip...
```

Figure 1: Content of of NetBSD pkgsrc package `openssh-7.5.1nb1.tgz`. Unlike FreeBSD ports, pkgsrc meta-data consist of small separate files.

files beginning with the character "+" are meta-data as same as `ports`(7). However unlike `ports`(7), the meta-data consist of small separate files(Figure 1).

To manage packages, we can use programs prefixed by `pkg_`. We run `pkg_add`(1) to install, `pkg_delete`(1) to de-install and `pkg_info`(1) to display the information of the specified package. It is important for us that several meta data files are used as running hooks in installation and de-installation processes (Table 3).

+CONTENTS	list of contents and other information
	hooked within `pkg_add`(1) and `pkg_delete`(1)
	modify ownership, groupship and permission
+INSTALL	a shell script
	hooked within `pkg_add`(1)
+DEINSTALL	a shell script
	hooked within `pkg_delete`(1)
+DISPLAY	message shown after installation

Table 3: A part of meta data contained in NetBSD pkgsrc package. They can be used to run hooks in installation/de-installation processes. For more details, see the online manual of `pkg_create`(1)[4].

2.2 Package Details for Base System

2.2.1 FreeBSD PkgBase

It is traditional that FreeBSD uses `make`(1) to build the kernel and userlands.

FreeBSD 11 introduced a base packaging mechanism PkgBase (packaged base) and a new package manager called "pkg" to manage the packages for both base and 3rd party software. PkgBase is a "beta feature in the FreeBSD 11 branch with r298107"[5] to manage the packaged base system in using `pkg`(8). The packages are created by running `make packages` after `make buildworld` and `make buildkernel` operations. The format of these packages is same as one of `ports`(7)'s package. In our environment FreeBSD base system comprises around 795 packages in the case of amd64 architecture by default.

In addition FreeBSD has another update utility `freebsd-update`(8) that is "used to fetch, install, and rollback binary updates to the FreeBSD base system" [6].

2.2.2 NetBSD syspkg

NetBSD also uses traditional `make` in actual building process of the kernel and userlands but NetBSD has a top level dispatcher `build.sh`[7] to build cross platform tool-chain, distributable tarballs and installation media and update the base system. It enables automatic cross build for all architectures NetBSD supports.

For base system packaging, NetBSD has a framework called "`syspkg`" introduced at January 8, 2002 by jwise @, `syspkg` is also merged into build.sh as a feature of the official building process. NetBSD wiki says "There has been a lot of work in this area already, but it has not yet been finalized"[8].

However `syspkg` is stagnant these years[2]. There has been several problems in `syspkg` for these years.

- `syspkg` database has been incomplete. See `syspkg` files such as `deps`, `comments`, and `attrs` under `src/distrib/sets/` for more details.

[2]For example, PR46937 (2012) is still open[9]. You can find `syspkg` has not been maintained since February 21, 2010 by judging log messages of distrib/syspkg directory in the source tree.

```
base-sys-usr-7.1.0.20170311.tgz/
    +CONTENTS
    +COMMENT
    +DESC
    +BUILD_INFO
    +PRESERVE
    usr/
        bin/
        lib/
    ...snip...
```

Figure 2: Example format of a syspkg package `base-sys-usr-7.1.0.20170311.tgz`. This format is old pkgsrc one. Compare this with Figure 1.

- `syspkg` package format is not effective today since it lacks several contents the current pkgsrc defines. Figure 2 shows the content of `base-sys-usr-7.1.0.20170311.tgz` created by running `build.sh syspkgs`.

- We can overwrite or remove important files by accident since `+PRESERVE` handling is incomplete.

  Accidental removal should be prohibited. `+PRESERVE` file implies "used to denote that the package should not be deleted"[4]. It is used to indicate that this package should not be removed. This property is important especially for some critical packages e.g. `etc-*.tgz` which contains `/etc/passwd`, `/etc/group` and so on.

  Also the overwrite should be prohibited. For example, when we can install packages e.g. `etc-*.tgz` using `pkg_add`(1), existing /etc files are overwritten by `pkg_add`(1).

It looks hard to directly fix `syspkg` framework which consists of a lot of makefiles, scripts and undocumented data. For this reason, we have developed another base packaging mechanism as a third party software by using only `syspkg` meta-data and making the best use of pkgsrc framework.

feature	`syspkg`	`basepkg`
language	Makefile and Bourne shell	Bourne shell
install script	none	available
kernel package	none	supporting GENERIC kernel
imported to	official source tree	pkgsrc-wip

Table 4: Comparison of features between `syspkg` and `basepkg`

3 Basepkg

3.1 What is Basepkg?

We have developed a new framework "`basepkg`" that can package NetBSD base system instead of `syspkg`. `basepkg` is an open source software distributed under BSD License. It published on `github.com/user340/basepkg` in Oct 26, 2016. It is imported to pkgsrc-wip on May 19, 2017. The feature comparison between `basepkg` and `syspkg` are shown at Table 4.

`basepkg` is a Bourne shell script. It analyzes metadata(s) and dispatches the corresponding `pkg_*` programs. `basepkg` is just a shell script up to about 1000+ lines, well coded and documented, so it is easy to read. `basepkg` makes the best use of pkgsrc framework as could as possible. In the case of `syspkg`, `make` and `shell` programming styles are mixed, and `syspkg` is at least 2 times larger than `basepkg`[3]. Hence we consider `basepkg` is simpler and can be maintained more easily than `syspkg`.

It is commonly seen that a program will be used longer than the author expected. To write a sustainable program, `basepkg` is written to be POSIX compliant and portable as could as possible. In fact the current coding is POSIX compliant except `hostname(1)`, `mktemp(1)` and `pkg_create`(1). We use ShellCheck[4] to validate and gain code quality and make the code warning-less as could as possible.

`basepkg` package format is same as pkgsrc one. Hence the packages can be managed by pkg_* utilities.

[3] The number of lines even in the two files (`src/distrib/syspkg/mk/bsd.syspkg.mk` and `distrib/sets/regpkg`) are about 1700 lines.

[4] http://www.shellcheck.net/

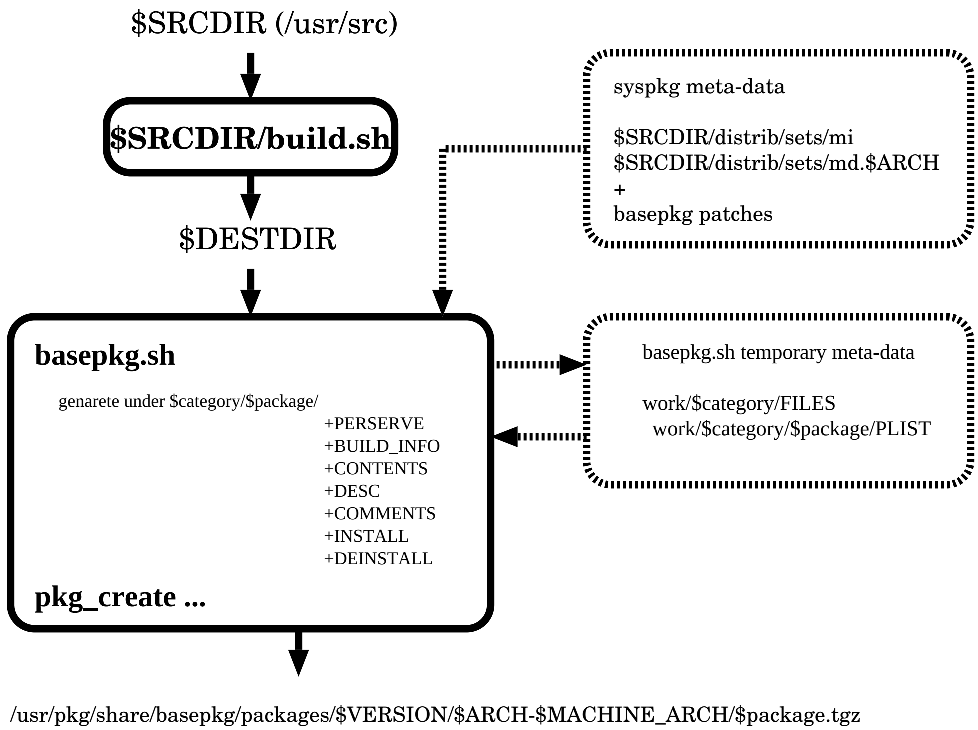

Figure 3: Basepkg Processing Internals: `basepkg` requires the built NetBSD base system on DESTDIR build.sh generates. It reads files at `/usr/src/distrib/sets/` with `basepkg` patches. It parses files to generate processed meta data temporarily and creates several pkgsrc style files under each package directory. Finally it runs `pkg_create`(1) to create the packages.

`basepkg` meta-data are derived from `syspkg` but corrected. For example, package utilities should not remove a package holding a "+PRESERVE" file. `syspkg` meta data are incomplete but `basepkg` has introduced a new `essential`[5] list to handle the preservation more correctly. Also `basepkg` checks package dependency more correctly.

3.2 Basepkg Processing Internals

We describe the `basepkg` processing details (Figure 3). It runs as follows:

[5] It seems that `syspkg` also tried it but it has been incomplete since `distrib/sets/attrs` contains a list with preserve flag.

1. `basepkg` gathers meta data from `syspkg` one and prepare the next step.

   `basepkg` reads list of a set of (file name, package name and options) from `sets/lists/base/mi` (machine/architecture independent list) and `sets/lists/base/md.ARCH` (ARCHitecture dependent list) under `/usr/src/disrib/` directory (see `distrib/sets/README` for more details on file definitions under `sets/`). After excluding files defined as `obsolete` in the `mi` and `md.ARCH` files, `basepkg` parses the list to generate `FILES` for each category e.g. `base`, `etc` and so on. Each `FILES` has a mapping between package names and filenames.

2. `basepkg` generates temporary meta data.

(a) basepkg reads FILES to create directories for the corresponding base packages.

(b) basepkg creates PLIST files for each package. Each PLIST holds a list contained in the package.

3. basepkg emulates the generation of pkgsrc metadata.

   (a) basepkg reads sets/essential to generate proper +PRESERVE files in the corresponding directories. It indicates that this package should not be removed.

   (b) basepkg creates +BUILD_INFO file for each package. It holds environment information in package building.

   (c) basepkg creates +CONTENTS file for each package. It holds a list of files each package contains and commands for pkg_* tools.

   (d) basepkg creates +DESC and +COMMENT files for each package. These are brief descriptions for the package.

   (e) basepkg creates +INSTALL and +DEINSTALL files to be hooked in installation (pkg_add(1)) and de-installation (pkg_delete(1)) processes.

4. basepkg runs pkg_create(1) for all packages (up to about 800) to generate packages. In creating packages, basepkg gathers the package content under DESTDIR directory build.sh generated.

5. basepkg creates the checksum files (both MD5 and SHA512) over all packages.

For regression test, we have verified that the content of tarball category.tgz is same as the sum of category-*.tgz base packages e.g. base.tgz $==\sum$ base-*.tgz.

3.3 Basepkg Installation

The latest version of basepkg can be obtained at github.com/user340/basepkg/releases. It requires the latest pkgtools/pkg_install, so we recommend the use of pkgsrc-wip/basepkg to install basepkg. When you install basepkg using pkgsrc-wip, basepkg is installed to /usr/pkg/share/basepkg directory by default.

3.4 How to Build Base Packages

The basepkg requires the built NetBSD base system (the whole set under DESTDIR in the term of build.sh) and pkg_* tools.

Firstly, we prepare the NetBSD binary at DESTDIR. We recommend building it from the NetBSD source tree[6].

```
# cd /usr/src
# ./build.sh -O ../obj -T ../tools tools
# ./build.sh -O ../obj -T ../tools distribution
# ./build.sh -O ../obj -T ../tools kernel=GENERIC
```

In this example, we assume the source directory is /usr/src, the obj root directory is /usr/obj, the tools directory is /usr/tools and basepkg root directory is the default one /usr/pkg/share/basepkg.

Secondly, we change to the directory where basepkg is installed (/usr/pkg/share/basepkg by default in using pkgsrc-ip). we run basepkg.sh with “pkg” and “kern” options to build base packages. basepkg generates packages at packages/[NetBSD_version]/[MACHINE]-[MACHINE_ARCH] directory under /usr/pkg/share/basepkg directory by default. For example, if you run basepkg.sh on NetBSD-7.1/amd64, the corresponding packages are generated at /usr/pkg/share/basepkg/packages/7.1/amd64-x86_64 directory.

```
# cd /usr/pkg/share/basepkg
# ./basepkg.sh pkg
# ./basepkg.sh kern
```

Figure 4 shows the part of etc-sys-etc-7.1.tgz package content created by basepkg. The format is same as pkgsrc one described above (See Section 2.1.2), so the package can be handled by pkg_* tools used in pkgsrc.

[6] In fact, the latest basepkg works well except for the kernel package building when we fetch binaries from NetBSD daily build system (nycdn.netbsd.org) and extract them under DESTDIR.

```
etc-sys-etc-7.1.tgz/
    +CONTENTS
    +COMMENT
    +DESC
    +INSTALL
    +DEINSTALL
    +BUILD_INFO
    boot.cfg
    dev/
        MAKEDEV
    ...snip...
```

Figure 4: Content of base package etc-sys-etc-7.1.tgz. The format is aligned to the modern pkgsrc style.

3.5 How to Handle Base Packages

In the previous section basepkg processing is mentioned from the point of administration or base package provider view. In this section, we describe how users and customers handle their system by using the base packages.

Firstly, it is easy to add or delete the specific base package by using pkg_* tools since the package format is same as pkgsrc one. pkg_add (1) can be used to install the package. To remove it, we use pkg_delete(1).

To avoid confliction between pkgsrc and basepkg packages, we should specify the other database path such as /var/db/basepkg by "-K" option in using pkg_* tools,

```
# cd ./packages/7.1/amd64-x86_64
# pkg_add    -K /var/db/basepkg games-games-bin
# pkg_delete -K /var/db/basepkg games-games-bin
```

Currently in using raw pkg_* tools to manipulate base packages, we need to be very careful to handle etc-* base packages such as etc-sys-etc-7.1.tgz since it overwrites files under the /etc directory. To avoid this disaster, once we extract the contents in another directory and determine to apply the content or not to /etc explicitly by hand.

```
# pkg_add -K /var/db/basepkg -p tmp/basepkg \
             etc-sys-etc-7.1.tgz
 ... apply it or not to /etc ...
```

To avoid these critical operations, we should prepare a wrapper for users and customers not to handle raw pkg_* tools.

Test	real time (s)	user time (s)	system time (s)
1	7.2374	0.2267	0.8443
2	19.2955	0.9457	1.1725
3	3.4656	0.0838	0.0924

Table 5: Comparison between processing time average among old and new installation methods.

3.6 Estimation of Basepkg Overhead

Packaging implies that an OS is built on a lot of small packages. Hence the OS update process to add or delete small parts must be faster. However the package size to add or delete is not proportional to the update processing speed since packaging introduces several new overheads e.g. resolution of dependencies among packages, execution of install scripts and so on.

We have compared the installation time between the traditional (tarball extraction) and our new method (basepkg based). We processed the following updates 100 times on NetBSD-7.1/amd64. We used time(1) command to measure the processing speed. The target category we used is "game" since "game" category is not mission critical.

1. Fetch a tarball "games.tgz" from ftp.jp.netbsd.org/pub/NetBSD/NetBSD-7.1/amd64/binary/sets/, then extract it at $HOME/tmp directory.
2. Install all packages beginning with "games-" to system from basepkg.netbsd.fml.org/pub/NetBSD/basepkg/7.1/amd64-x86_64
3. Install one "games-games-bin" package to system from basepkg.netbsd.fml.org/pub/NetBSD/basepkg/7.1/amd64-x86_64

where basepkg.netbsd.fml.org is an experimental base package distribution server we build and operate (See Appendix A for the server details). Table 5 shows the average time of the processing speed.

Table 5 verifies that our new installation using basepkg is faster than the traditional one. However it is not faster than we expected because of overheads mentioned above. Only when we update a few packages in the system, the process is comparable to the traditional one. In almost cases under normal operation, we replace only a few small parts for rapid security update. In addition it is good we explicitly know which parts we replace, not a large archive `base.tgz`. Hence we consider base packaging is meaningful for users and customers.

4 Discussion

Firstly, we summarize changes and improvements from AsiaBSDCon2017[1].

- import to pkgsrc-wip repository.
- `syspkg` meta-data handling fixes:
  - not generate obsolete packages.
  - enhance +PRESERVE handling to cover base, etc and shlib.
- hook support running +INSTALL and +DEINSTALL.
- cross build support.
- multi platform support.

  We have verified `basepkg.sh` can run on Ubuntu 17.04.

There are a lot of technical issues to resolve as follows:

- `basepkg` processing speed.

  We need to profile `basepkg` to improve the processing speed. `basepkg` runs slower than `syspkg`. Table 5 shows that `basepkg` user mode processing is about 4 times larger than `syspkg` one. It is not clear but the low speed may come from that `basepkg` creates a lot of directories.

  We must need to try better shell coding technique. For example, we should not use `for` nor `while` loop as could as possible, instead use internal loops such as `find` and `grep`. Matsuura et.al. says "Processing speed can be improved when we use POSIX command chain through pipes with least bifurcations and loops." [10].

- `basepkg` database maintenance.

  `basepkg` can run hooks within the processing. We need to maintain hooks for such as PRE-INSTALL, POST-INSTALL et.al. within `basepkg` own meta data in addition to the current patches. It is by nature better to merge it back to `/usr/src/distrib/`.

- `syspkg` database maintenance.

  `basepkg` uses `syspkg` meta data under `src/distrib/sets/`. It is not clear who ensures the consistency under `src/distrib/sets/` files. For example, it looks `src/distrib/sets/descrs` and `src/distrib/sets/comments` has been incomplete.

- more user friendly naming convention.

  `syspkg` database naming convention is not clear for users and customers. It should be changed to more plain naming convention. For example, a base package name `base-postfix-bin` for `postfix` is obvious. However `base-secsh-bin` for `openssh` is far from `openssh` we expect. It is more difficult to find `openssl` than examples mentioned above. The shared library `libssl.so` is contained in `base-crypto-shlib`. The library `libssl.a` used in compilation is contained in `comp-c-lib`. In this example, the granularity should be too re-considered since `comp-c-lib` includes several kinds of libraries. To resolve this difficulty, as a workaround, it is better to provide a wrapper with naming mapping service.

- a wrapper convenient for users and customers.

  To resolve the issue mentioned at Section 3.5, we should provide a wrapper utility to manipulate base packages. This utility hides raw use of `pkg_*` tools and the database location `/var/db/basepkg`. It is useful to provide the following functions.

  - It warns or asks the user instructions step by step if `etc-*` is specified as the argument to avoid unexpected overwrite of `/etc`.

- It is better to provide alias mapping for ambiguous package names. For example, `wrapper update openssh` actually runs `pkg_delete base-secsh-bin.tgz` and `pkg_add base-secsh-bin.tgz`.
- It caches the fetched packages under `/var/cache/baspekg` for later use. The cache remains unless you run `wrapper clean`.
- It can rollback the specified base package cached above.

- integrated system management support.

  For users and customers, it must be useful to provide automatic management function for the base system like `apt` (Debian/Linux Advanced Package Tool). For example, `wrapper update` fetches the latest package database and `wrapper upgrade` upgrades (deletes and adds) base packages automatically. This function implies support of automatic vulnerability check for base packages.

  `basepkg` is built on `pkg_*` tools, so integration with pkgsrc framework must be easy.

  Currently pkgsrc vulnerability can be checked automatically but the base system check depends on your eyes[7]. The automatic vulnerability check for base system is useful for users and customers.

  The vulnerability database of base packages can be managed under pkgsrc audit-packages framework. The database is same as `/var/db/pkg/pkg-vulnerabilities` like this:

  ```
  sys-secsh-bin<20171220  reason...  url...
  ```

  It must be better that `basepkg` works with `pkgin`(`pkgsrc/pkgtools/pkgin`) to cover both base and pkgsrc packages totally.

- base package distribution support.

  It is not useful unless latest base packages are not provided. It is required to support automatic updates, rollbacks et.al. described above.

[7] You need to action based on NetBSD security advisory release.

Currently we build and operate an experimental base package distribution server (See Appendix A for the server details) but our machine power can generate base packages for at most 30 architectures on only latest NetBSD stable branch within one day. Appendix A discusses the cost evaluation to operate more rapid up-to-date system.

5 Conclusion

We have developed another framework “basepkg” to package NetBSD base system. It is shown that this framework provides more granular and faster update of NetBSD base system and useful functions for users and customers. However we have a lot of issues to resolve for realistic system operations, so we need to continue dogfooding and development.

References

[1] Yuuki Enomoto and Ken'ichi Fukamachi, 2017, *Maintain the NetBSD Base System Using pkg_**, `www.netbsd.org/gallery/presentations/yuuki/2017_AsiaBSDCon/basepkg.pdf`

[2] The FreeBSD Documentation Project, Revision: 51193, *Chapter 4. Installing Applications: Packages and Ports*, FreeBSD Handbook, `www.freebsd.org/doc/handbook/ports.html`

[3] NetBSD, 2007, *pkgsrc*, Miscellaneous Information Manual.

[4] Jordan Hubbard, John Kohl and Hubert Feyrer, 2010, *pkg_create*, General Commands Manual

[5] `wiki.freebsd.org/PkgBase`

[6] Colin Percival, 2017, *freebsd-update*, FreeBSD System Manager's Manual.

[7] Luke Mewburn and Matthew Green, 2003, *build.sh: Cross-building NetBSD*, BSDCon '03.

[8] The NetBSD Project, 2014, *syspkgs*, `wiki.netbsd.org/projects/project/syspkgs/`

[9] Lloyd Parkes, 2012, *Lots of broken syspkgs*, NetBSD Problem Report #46937, `gnats.netbsd.org/cgi-bin/query-pr-single.pl?number=46937`

[10] Tomoyuki Matsuura, Hiroyuki Ohno and Nobuaki Tonaka, 2017, *POSIX Centric Programming to Make Software More Compatible and Sustainable*, Digital Practice 8(4), 352-360, `https://www.ipsj.or.jp/dp/contents/publication/32/S0804-R1601.html`.

A Estimation of Base Package Distribution

A.1 Build By Ourself

A.1.1 Current VPS Case

Experimentally we operate a base package distribution server **basepkg.netbsd.fml.org** which runs on SAKURA Internet VPS (SAKURA VPS(v3) 2G plan: 3 CORE CPU, 200GB storage, about 150 USD (16,745 JPY) per year)[8] . Currently we provide 30 architectures in NetBSD 7 STABLE.

We need to run a set of `build.sh` and `basepkg.sh` for all target architectures since `basepkg.sh` requires compiled objects at DESTDIR `build.sh` generates (Section 3). Fortunately this process can run parallelly, so we can run one set for one architecture on one CPU CORE. If we can prepare enough large storage (5GB per architecture per version), we can run `build.sh` with `-u` option (do not run "make cleandir"). The upper limit of 30 architectures are restricted by this storage limit (200GB) to run `build.sh -u`.

The base package building process costs about 1 hour per target where `build.sh -u` requires about 1000 sec. and `basepkg.sh` requires about 2000 sec. Hence we need 10 hours to prepare 30 architectures even if we can run processes parallelly per CPU CORE. When we clean up the working directories (i.e. `build.sh` runs normally without -u option), the building process requires 6 times processing time per target.

A.1.2 Cloud Case

The evaluation is underway.

Cloud service is more suitable for intermittent work like this. The updates for stable branches are rare, so we do not need to build base packages daily. If we run this building process only when a NetBSD security advisory is released and the target can be restricted to stable branches, modern cloud service is more proper than the current VPS service. We need to estimate the cloud speed and cost whether the cost

[8] https://vps.sakura.ad.jp/

may be lower since cloud is CPU meter rate charging. In the case of cloud service, we assume the following usage:

- Normally the build process does not run. The low cost cloud archive holds the built data (the previous build result).
- On demand, we wake up the cloud service, extract the built data from the archive, build base packages (by running `build.sh -u` and `basepkg`), update web servers, re-archive the built data and make the cloud sleep again.

A.2 Using NetBSD Daily Build System

Today it looks NetBSD daily build system can prepare daily binaries for some branches e.g. NetBSD 7.1 stable branch. Hence `basepkg` distribution server can fetch the tarballs and build base packages based on them.

The processing time seems almost comparable to the VPS case running `build.sh -u` described above but with less storage consumption (1GB per architecture per version). Hence we hope to operate base package distribution server at a low cost but only for latest branches. The details of evaluation will be reported at AsiaBSDCon 2018.