2014-07-11 – 13:00

PKG:

History:

- The old package system is 20 years old. There has been very little change since inception. The requirements at the time were minimal. The modern ports tree has much more complex requirements for a packaging system
- We need to be able to make any changes to the new package system as seamless as possible. This causes pkg to have a lot of hacks to be able to create a smooth transition.
- It is complicated to modernise the ports system to allow for new flexible packages.
- We currently have no way to re-build a single package where there is a security vulnerability, as we have no metric on what is statically compiled.
- One solution is to use more pkg inside the ports tree itself.

Developers	Normal	Ricers (gamers?)	Industry
Want as much customisability as possible -Headers -compilers	Don't care about building anything Just want it to work Sane default options	Want to tune everything Gamers? Combination of optimising specific games, and people optimising for fun	Similar to normal Might need specific options for customers

The major differences between these categories are to do with configurable options:

- Most of them are useless / have no meaning
- E.g. multiple methods for Unicode in python

Port maintainers provide default options, but maybe they're tailored more to developers than end users?

Problem currently is that unit testing is not done, but impossible for all combinations of options (e.g. nginx has 80+ options).

The objective is not to hide all of the options. Rather to provide sub packages.
e.g. with ports with multiple database backends, provide a different package for each database backend. (e.g. bogofilter).

Consensus was reached that maintainers should have more robust rules on default port options.

It was suggested that we try something similar to Debian's "policy manual", which provides guidance on the sorts of desired compile options / backends.

It is now much more automated to query the ports tree for which options are default thanks to Bapt's recent Makefile framework.

We have an issue currently with lack of statistics on popular ports / packages. We are looking to begin gathering aggregate download statistics with version 1.4. However, this doesn't extend to packages compiled from ports.

We need to try to remove old / broken by default packages. This would involve both automated testing and volunteers. This includes packages that compile but don't work properly at runtime.

Regression testing infrastructure is in development but unfinished. It's usually the ports without regression testing that are broken. Test by Figurign out where binaries are in staging area, and try running them and wait for a time-out.

Branching the ports tree. We could have a branch which has periods of stability. Only applying security and bug fixes. This would have issues with lack of maintainers. Having branching in the ports tree would allow the HEAD branch to move more quickly, as there would be less immediate concern about breakage.

We should only have build dependencies. We should be able to determine run-time dependencies from staging, and not have hard-coded version dependencies. Put everything into depends. This also ensures that ports can depend on packages in the base system, rather than requiring a specific version from ports.

We currently have no easy way of seeing which options are enabled for packages in the base system, this will become more manageable with packaging the base system.

We have ports trees for:

- HEAD
- 10
- 9
- 8
- MIPS
- PPC
- ARM
- X86
- Sparc64

Issues arise with different default compilers.

The old version of GCC shipped in base has an incompatible libstdc++ with both clang and modern GCC. FreeBSD 8 also has issues where even if you install a newer GCC, it has outdated libraries which cause it to miss c++11 functionality.

One suggestion is each ports tree having a specified / mandated compiler. The issue with this is that the entire base system API is still exposed. Either you can do what PKGSRC, or you can treat every pkg build as a cross-build. This has issues with ports that won't cross-build such as perl and python.

Suggested we could have a way of defining "core" packages. This could be based on the number of dependent ports.

We don't have enough manpower or motivation for stable branches for ports, based on the trial from quarterly branches.

We need to modify GCC to use libc++ instead of libstdc++ so we're only actively maintaining and supporting a single c++ lib.

Pkg 1.3:

Pkg is now able to install packages and handle all the names, splits and changing variables. This means that if a package is renamed in the repo, the local pkg will uninstall the old name and install the new name.

Currently, it will check for any packages with similar naming (such as moved -, . etc.).

It would be good if a port could be defined as "this port replaces port x". This is a planned feature coming to ports. This could also update any packages that depend on the newly upgraded package.

We have issues for package repository maintainers with conflicts, and how to resolve them.

Pkg 1.3 rc1 attempted to fetch everything it thought it might need, it resulted in multiple GB dependency fetching; not ideal.

Pkg 1.4:

Three main features:

- Flexible repos and dependencies
- Packaging base system
- Popularity contest

Flexible repos would be sourcing packages from multiple external repos. For example, "pkg install –t ports [package]" would install a package from the ports tree.

Allow finding suitable dependencies from several sources.

Base system. We need to be able to handle options for base system components. We would handle this with option sets, like in ports. We need statistics on the sorts of options that people need for sane default options.

We also need to work on how to merge configuration files. This would involve explicitly marking config files, and saving default config files. It would be a three-way merge between the default, the old and the new.

UNRELATED NOTE: We need to kill all interactivity with port installs (interactive install scripts). We can add this in properly later as a new feature, but it is currently incompatible with pkg.

We also need to decide where to store default configuration files, for use by pkg in 3-way merging.

"We won't be sane for FreeBSD 11" (courtesy of Bapt). (A discussion stemming from how to handle default configs in the base system for 11-RELEASE).

It is less simple to handle default configs for ports. For example, currently maintainers have used ".example" ".orig" ".diff" "-example" etc.

One solution could be storing defaults in /usr/local/share/examples, with a compressed backup in sqlite in case the user deletes it.

Popularity contest:

Want to anonymously gather statistics, with protection against eavesdropping and poisoning the database. Want to use HTTP, as HTTPS is filtered by many corporate files, and DNS is blocked in many places too.

Want to introduce crypto puzzle. Server generates string based on client

Discussion arose over whether poisoning can effectively be blocked. Even if crypto puzzles were used, they would just be limiting the largest poisoning to the people with the most CPU time.

We are seeking opinions on how popularity contest could be implemented.

One suggestion was that the popularity contest database could be initially seeded from internal sources within the FreeBSD community (such as all FreeBSD committers). This could be in the form of a survey of some description distributed (or at least announced) by email. Whilst the numbers would be significantly lower than any potential later data gathering, it would provide a reasonably reliable base for the statistics.

PKG in base:

How fine-grained should the packages be split?
What do we support, kernel without IPv6? For instance, do we tell every package that "You can't be installed because you only support IPv6".

We don't currently have a suitable system for declaring available kernel features (not sysctl).

There are issues with immutable flags not being removable in jails, meaning that you will be unable to upgrade immutable files in jails.

It is unlikely that manpages will be a separate package. It is more likely that packages will have a man page option on a per-package basis.

There is an existing script in the ports tree which will generate a list of files that were installed by buildworld.

Packages:

- Kernel
  - Kernel-debug
- Devel
  - Compiler
  - Linker
- Runtime

    - Libs
    - bin
    - share
- Debug
- Sendmail
- Rescue (maybe in runtime)

Security:

Security is important:

- Installed as root user

Three levels of security issues:

- Extracting (solved with sandboxing) (the actual installation isn't sandboxed)
- Parse VUXML (solved with sandboxing)

Extraction and verification are in separate sandboxes. This is partly for verification, but largely to protect against buffer overflow attack.

The aim is to remove all executable scripts from the package installation process, which could be led by example for the packaging of the base system.

We currently only have unsigned VUXML to publish security vulnerabilities (CVEs) in ports. The matching algorithm is also poor, producing false positives and negatives.

The CPE specification would provide a more robust matching algorithm.