FreeBSD package management system

Vsevolod Stakhov
vsevolod@FreeBSD.org

FreeBSD

pkgSrcCon July 01, 2017

What is pkg

Pkg (previously pkgng) is the binary package management system written for FreeBSD but ported for other BSD and Linux platforms.

- **Binary packages** management

What is pkg

Pkg (previously pkgng) is the binary package management system written for FreeBSD but ported for other BSD and Linux platforms.

- **Binary packages** management
- Replaces old pkg_⋆ tools

What is pkg

Pkg (previously pkgng) is the binary package management system written for FreeBSD but ported for other BSD and Linux platforms.

- **Binary packages** management
- Replaces old `pkg_*` tools
- Uses central *sqlite3* based storage

What is pkg

Pkg (previously pkgng) is the binary package management system written for FreeBSD but ported for other BSD and Linux platforms.

- **Binary packages** management
- Replaces old `pkg_*` tools
- Uses central *sqlite3* based storage
- Provides the comprehensive toolset for **binary packages** management

Pkg development goals

The main goal of pkg is to simplify system management tasks.

- Easy install, remove and upgrade of binary packages

Pkg development goals

The main goal of pkg is to simplify system management tasks.

- Easy install, remove and upgrade of binary packages
- Integration with the **ports** (easily adopted for other source management systems, e.g. pkgsrc)

Pkg development goals

The main goal of pkg is to simplify system management tasks.

- Easy install, remove and upgrade of binary packages
- Integration with the **ports** (easily adopted for other source management systems, e.g. pkgsrc)
- Automatic **resolving** of dependencies and conflicts

Pkg development goals

The main goal of pkg is to simplify system management tasks.

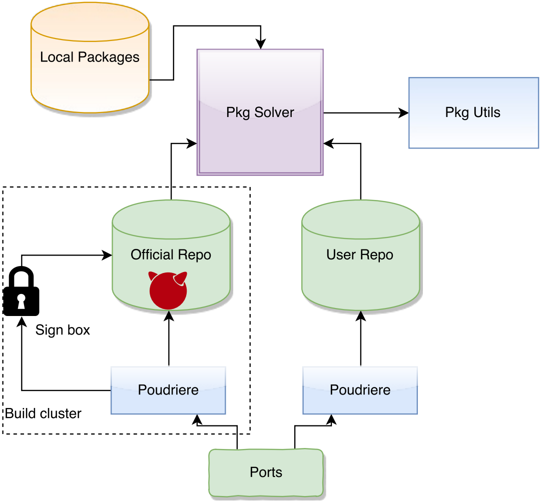
- Easy install, remove and upgrade of binary packages
- Integration with the **ports** (easily adopted for other source management systems, e.g. pkgsrc)
- Automatic **resolving** of dependencies and conflicts
- Encourage users to install software from **binary packages**

Pkg development goals

The main goal of pkg is to simplify system management tasks.

- Easy install, remove and upgrade of binary packages
- Integration with the **ports** (easily adopted for other source management systems, e.g. pkgsrc)
- Automatic **resolving** of dependencies and conflicts
- Encourage users to install software from **binary packages**
- ...but do not prevent users from building custom packages using the **ports**

Ports management using pkg

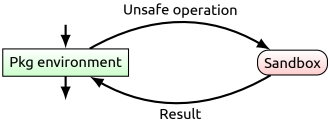


Unique pkg features

- Dependencies and conflicts solver that can automatically resolve complex upgrade or install scenarios

Unique pkg features

- Dependencies and conflicts solver that can automatically resolve complex upgrade or install scenarios
- Improved security by **sandboxing** (Capsicum) untrusted operations:

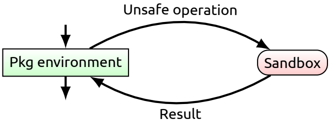


Sandboxing:

- **archives** extracting

Unique pkg features

- Dependencies and conflicts solver that can automatically resolve complex upgrade or install scenarios
- Improved security by **sandboxing** (Capsicum) untrusted operations:

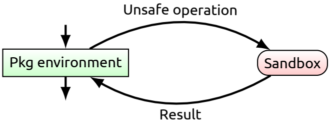


Sandboxing:

- **archives** extracting
- **vulnxml** parsing

Unique pkg features

- Dependencies and conflicts solver that can automatically resolve complex upgrade or install scenarios
- Improved security by **sandboxing** (Capsicum) untrusted operations:

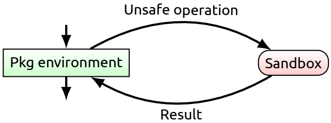


Sandboxing:

- **archives** extracting
- **vulnxml** parsing
- repositories **signatures** checking and public keys extracting

Unique pkg features

- Dependencies and conflicts solver that can automatically resolve complex upgrade or install scenarios
- Improved security by **sandboxing** (Capsicum) untrusted operations:

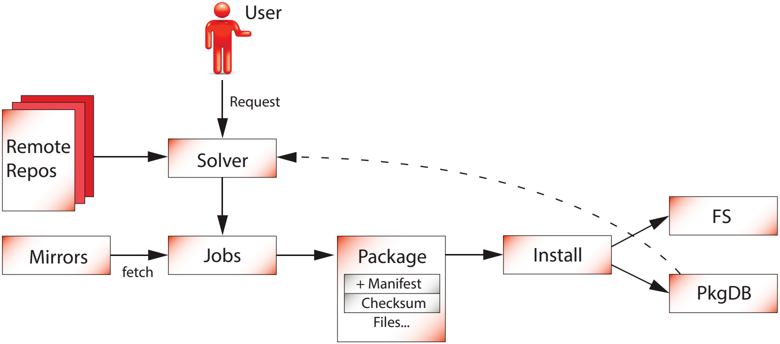


Sandboxing:

  - **archives** extracting
  - **vulnxml** parsing
  - repositories **signatures** checking and public keys extracting

- Concurrent **locking** system

Pkg architecture



The problems of the old solver in pkg

- Absence of conflicts resolving

The problems of the old solver in pkg

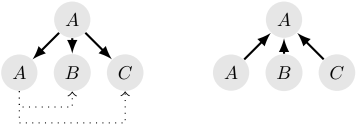
- Absence of conflicts resolving
- No **alternatives** support (plain dependencies only)

The problems of the old solver in pkg

- Absence of conflicts resolving
- No **alternatives** support (plain dependencies only)
- Can perform merely a **single task**: either install or upgrade or remove

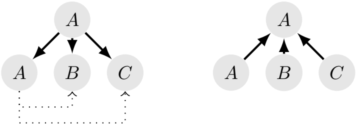
Tasks to solve

- Ports renaming:
  - simple: `racket-textual` → `racket-minimal`
  - splitting/merging:



Tasks to solve

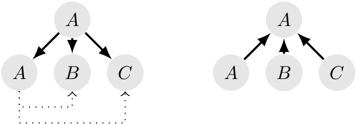
- Ports renaming:
  - simple: `racket-textual` → `racket-minimal`
  - splitting/merging:



- Ports reorganising:
  - files **moving**

Tasks to solve

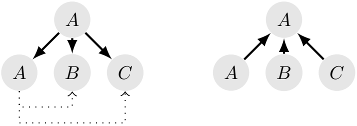
- Ports renaming:
  - simple: `racket-textual` → `racket-minimal`
  - splitting/merging:



- Ports reorganising:
  - files **moving**
  - **dependencies** change

Tasks to solve

- Ports renaming:
  - simple: `racket-textual` → `racket-minimal`
  - splitting/merging:



- Ports reorganising:
  - files **moving**
  - **dependencies** change
  - adding or removing new **conflicts**

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list
- Set jobs priorities using the following rules:

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list
- Set jobs priorities using the following rules:
  - install **dependencies** first

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list
- Set jobs priorities using the following rules:
  - install **dependencies** first
  - check for **reverse dependencies** and increase priority

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list
- Set jobs priorities using the following rules:
  - install **dependencies** first
  - check for **reverse dependencies** and increase priority
  - deal with **conflicts** using the same priority

Tasks to solve

There are another issues to be resolved:

- Find conflicts using files list
- Set jobs priorities using the following rules:
  - install **dependencies** first
  - check for **reverse dependencies** and increase priority
  - deal with **conflicts** using the same priority
  - packages removing **reverses** the priority order

Existing systems

There are many examples of solvers used in different package management systems, for example:

- Zypper/SUSE - uses libsolv as the base
- Yum/RedHat - migrating to libsolv
- OpenBSD/pkg_add - uses internal solver
- Apt/Debian - uses internal solver
- Pacman/Archlinux - uses internal solver

External solvers

To interact with an external solver we have chosen the CUDF format used in the Mancoosi research project `http://mancoosi.org`:

```
^^Ipackage: devel/libblah
^^Iversion: 1
^^Idepends: x11/libfoo

^^Ipackage: security/blah
^^Iversion: 2
^^Idepends: devel/libblah
^^Iconflicts: security/blah-devel

^^I
```

Interaction with external solver

There are some limitations and incompatibilities with CUDF.

- CUDF supports plain integers as versions and we need to convert versions twice

Interaction with external solver

There are some limitations and incompatibilities with CUDF.

- CUDF supports plain integers as versions and we need to convert versions twice
- There is no support of options in CUDF packages formulas

Interaction with external solver

There are some limitations and incompatibilities with CUDF.

- CUDF supports plain integers as versions and we need to convert versions twice
- There is no support of options in CUDF packages formulas
- External solvers are often too complicated and large

We need an internal solver!

Alternatives:

- Write own logic of dependencies and conflicts resolution?

We need an internal solver!

Alternatives:

- Write own logic of dependencies and conflicts resolution?
- Use some existing solution?

We need an internal solver!

Alternatives:

- Write own logic of dependencies and conflicts resolution?
- Use some existing solution?
- Use some known algorithm?

We need an internal solver!

Alternatives:

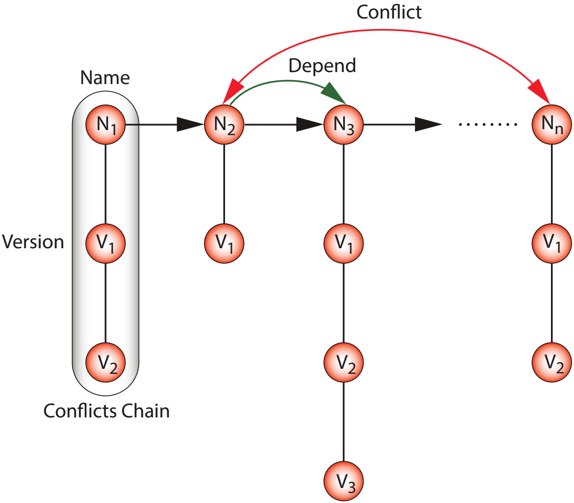
- Write own logic of dependencies and conflicts resolution?
- Use some existing solution?
- Use some known algorithm?

Use SAT solver for packages management

$$\overbrace{\underbrace{(x_1 \| \neg x_2 \| x_3)}_{\text{Clause}} \& (x_3 \| \neg x_1) \& (x_2)}^{\text{SAT expression}}$$

Packages universe

We convert all packages involved to a packages universe of the following structure:



Making a SAT problem

- Assign a **variable** to each package: package A $\rightarrow a_1$, package B $\rightarrow b_1$

Making a SAT problem

- Assign a **variable** to each package: package A $\rightarrow a_1$, package B $\rightarrow b_1$
- Interpret a request as a set of **unary clauses**:
  - Install/Upgrade package A $\rightarrow (a_1)$
  - Delete package B $\rightarrow (\neg b_1)$

Making a SAT problem

- Assign a **variable** to each package: package A $\to a_1$, package B $\to b_1$
- Interpret a request as a set of **unary clauses**:
  - Install/Upgrade package A $\to (a_1)$
  - Delete package B $\to (\neg b_1)$
- Convert dependencies and conflicts to **disjuncted clauses**

Converting dependencies and conflicts

- If package A depends on package B (versions B_1 and B_2), then we can either have package A not installed or any of B installed:

$$(\neg A \| B_1 \| B_2)$$

Converting dependencies and conflicts

- If package A depends on package B (versions B_1 and B_2), then we can either have package A not installed or any of B installed:

$$(\neg A \| B_1 \| B_2)$$

- If we have a conflict between versions of B (B_1, B_2 and B_3) then we ensure that merely one version is installed:

$$\underbrace{(\neg B_1 \| \neg B_2) \& (\neg B_1 \| \neg B_3) \& (\neg B_2 \| \neg B_3)}_{\text{Conflicts chain}}$$

The solving of SAT problem

Some rules to follow to speed up SAT problem solving.

- Trivial propagation - solve **unary** clauses

The solving of SAT problem

Some rules to follow to speed up SAT problem solving.

- Trivial propagation - solve **unary** clauses
- **Unit propagation** - solve clauses with only a single unsolved variable

The solving of SAT problem

Some rules to follow to speed up SAT problem solving.

- Trivial propagation - solve **unary** clauses
- **Unit propagation** - solve clauses with only a single unsolved variable
- **DPLL** algorithm backtracking (conflict driven resolution is used now)

The solving of SAT problem

Some rules to follow to speed up SAT problem solving.

- Trivial propagation - solve **unary** clauses
- **Unit propagation** - solve clauses with only a single unsolved variable
- **DPLL** algorithm backtracking (conflict driven resolution is used now)
- **Package specific** assumptions.

SAT problem propagation

- Trivial propagation - direct install or delete rules

$$(\neg A \| B) \& \underbrace{(A)}_{\textit{true}} \& \underbrace{(\neg C)}_{\textit{false}} \& (\neg A \| \neg D)$$

SAT problem propagation

- Trivial propagation - direct install or delete rules

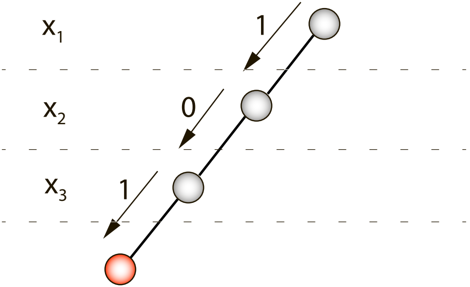
$$(\neg A \| B) \& \underbrace{(A)}_{\text{true}} \& \underbrace{(\neg C)}_{\text{false}} \& (\neg A \| \neg D)$$

- Unit propagation - simple depends and conflicts

$$\underbrace{\overbrace{(\neg A \| B)}^{\text{Dependency}}}_{B \to \text{true}} \& \overbrace{(A)}^{\text{true}} \& \overbrace{(\neg C)}^{\text{false}} \& \underbrace{\overbrace{(\neg A \| \neg D)}^{\text{Conflict}}}_{D \to \text{false}}$$

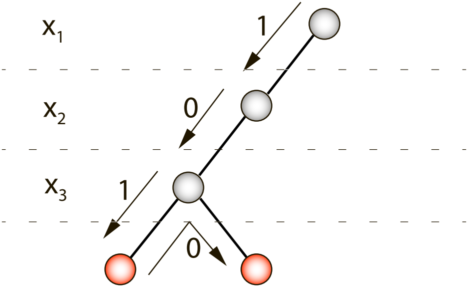
DPLL algorithm

DPLL is proved to be one of the efficient algorithms to solve SAT problem (not the fastest but more simple than alternatives).



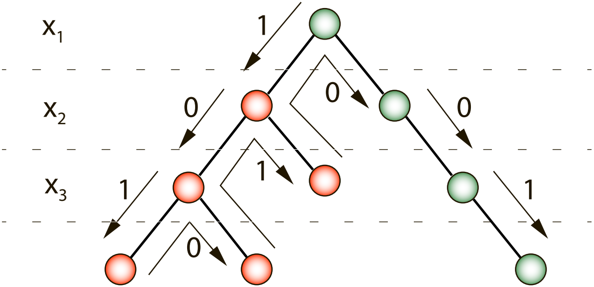
DPLL algorithm

DPLL is proved to be one of the efficient algorithms to solve SAT problem (not the fastest but more simple than alternatives).



DPLL algorithm

DPLL is proved to be one of the efficient algorithms to solve SAT problem (not the fastest but more simple than alternatives).



Package specific assumptions

Pure SAT solvers cannot deal with package management as they do not consider several packages peculiarities:

Package specific assumptions

Pure SAT solvers cannot deal with package management as they do not consider several packages peculiarities:

- Try to keep **installed** packages (if no direct conflicts)

Package specific assumptions

Pure SAT solvers cannot deal with package management as they do not consider several packages peculiarities:

- Try to keep **installed** packages (if no direct conflicts)
- Do not install packages if they are not **needed** (but try to upgrade if a user has requested upgrade)

Package specific assumptions

Pure SAT solvers cannot deal with package management as they do not consider several packages peculiarities:

- Try to keep **installed** packages (if no direct conflicts)
- Do not install packages if they are not **needed** (but try to upgrade if a user has requested upgrade)
- Additional logic when dealing with **multi-repo**

Solvers and Pkg

- Pkg may pass the formed universe to an external CUDF solver:

Solvers and Pkg

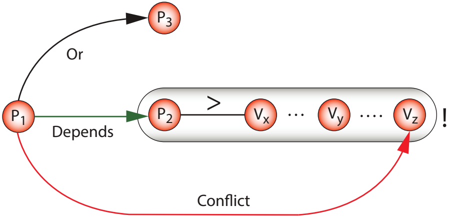
- Pkg may pass the formed universe to an external CUDF solver:
  - convert **versions**
  - format request
  - parse output

Solvers and Pkg

- Pkg may pass the formed universe to an external CUDF solver:
  - convert **versions**
  - format request
  - parse output
- Alternatively the internal SAT solver may be used:
  - convert the universe to **SAT** problem
  - formulate request
  - ???
  - **PROFIT**

New dependencies formulae

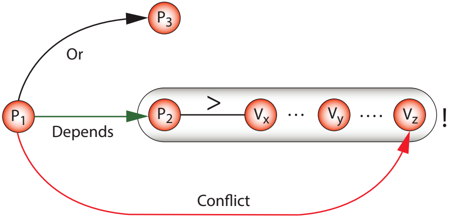
$$libblah >= 1.0 + option_1, +option_2 || libfoo! = 1.1$$



New dependencies formulae

$$libblah >= 1.0 + option_1, +option_2 || libfoo! = 1.1$$

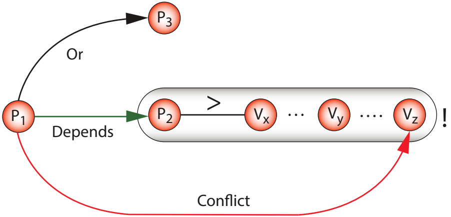
- Can depend on normal packages and **virtual packages** (provides)



New dependencies formulae

$$libblah >= 1.0 + option_1, +option_2 || libfoo! = 1.1$$

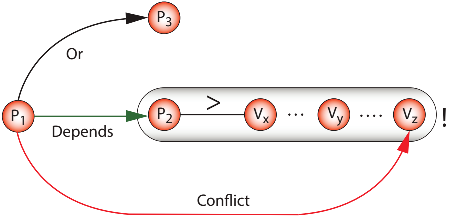
- Can depend on normal packages and **virtual packages** (provides)
- Easy to define the **concrete** dependency versions



New dependencies formulae

$$libblah >= 1.0 + option_1, +option_2 || libfoo! = 1.1$$

- Can depend on normal packages and **virtual packages** (provides)
- Easy to define the **concrete** dependency versions
- **Alternative** dependencies



Packages format

- Uses **xz** format:

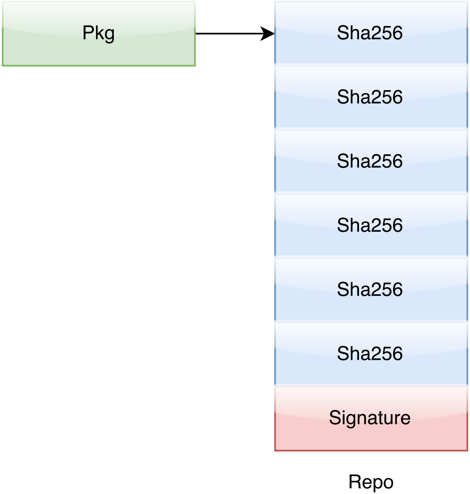
Packages format

- Uses **xz** format:
-
    - **Terribly** slow
    - XZ format is too **complicated**
    - No place to put **signature**

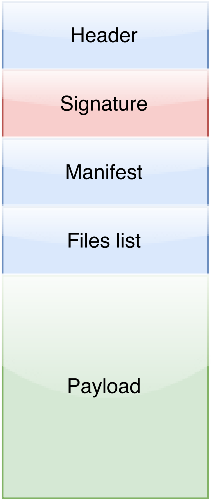
Packages format

- Uses **xz** format:
-
    - **Terribly** slow
    - XZ format is too **complicated**
    - No place to put **signature**
- Metadata is compressed together with payload (**slow index**)

Signatures now



New format proposal



Ed25519 via asignify

For faster index

Plain tar + zstd compression

New format proposal

- Uses **zstd** compression: very fast and good compressin rate (faster and better than gz)

New format proposal

- Uses **zstd** compression: very fast and good compressin rate (faster and better than gz)
- Fast **index**, allow to build repos rapidly

New format proposal

- Uses **zstd** compression: very fast and good compressin rate (faster and better than gz)
- Fast **index**, allow to build repos rapidly
- Has place for dedicated signature

New format proposal

- Uses **zstd** compression: very fast and good compressin rate (faster and better than gz)
- Fast **index**, allow to build repos rapidly
- Has place for dedicated signature
- Extendable

New format proposal

- Uses **zstd** compression: very fast and good compressin rate (faster and better than gz)
- Fast **index**, allow to build repos rapidly
- Has place for dedicated signature
- Extendable
- No direct support of libarchive (yet)

Questions?

vsevolod@FreeBSD.org