

# FreeBSD package management system

Vsevolod Stakhov  
vsevolod@FreeBSD.org



FreeBSD

BSDCan May 17, 2014

# What is pkg

Pkg (previously pkgng) is the binary package management system written for FreeBSD.

- ▶ 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.

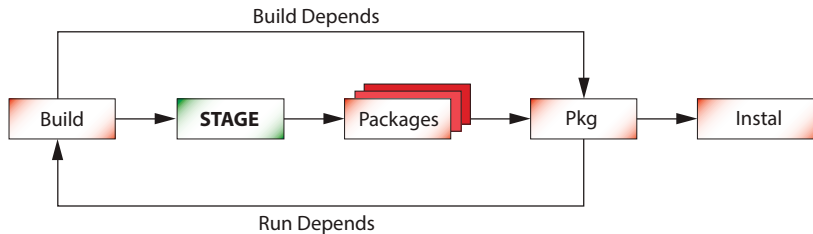
- ▶ Easy install, remove and upgrade of binary packages
- ▶ Integration with the ports
- ▶ Automatic resolving of dependencies and conflicts
- ▶ Provide secure package management tool
- ▶ Encourage users to install software from binary packages

# Pkg development goals

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

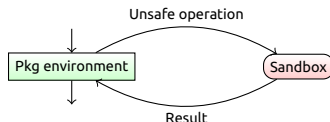
- ▶ Easy install, remove and upgrade of binary packages
- ▶ Integration with the ports
- ▶ Automatic resolving of dependencies and conflicts
- ▶ Provide secure package management tool
- ▶ Encourage users to install software from binary packages
- ▶ . . . but do not prevent users from building custom packages using the ports

# Planned ports and pkg interaction



# What is new in pkg 1.3

- ▶ New solver that can automatically resolve complex upgrade or install scenarios
- ▶ Improved security by sandboxing untrusted operations:

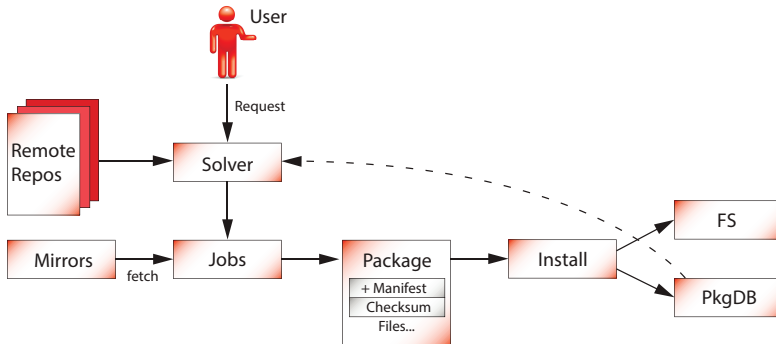


## Sandboxing:

- ▶ archives extracting
- ▶ vulnxml parsing
- ▶ repositories signatures checking and public keys extracting
- ▶ Concurrent locking system



# Pkg architecture



# The problems of the old solver in pkg

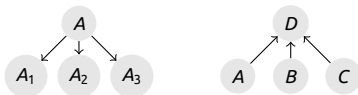
- ▶ Absence of conflicts resolving
- ▶ No alternatives support (plain dependencies only)
- ▶ Can perform merely a single task: either install or upgrade or remove



# Tasks to solve

- ▶ Ports renaming:

- ▶ simple: racket-textual  $\rightarrow$  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
- ▶ 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
- ▶  Apt/Debian - uses internal solver
- ▶  Pacman/Archlinux - uses naive 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>:

```
package: devel/libblah
```

```
version: 1
```

```
depends: x11/libfoo
```

```
package: security/blah
```

```
version: 2
```

```
depends: devel/libblah
```

```
conflicts: security/blah-devel
```

# 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
- ▶ CUDF transformation is expensive in terms of performance

# 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

$$\overbrace{(x_1 \parallel \neg x_2 \parallel x_3) \& (x_3 \parallel \neg x_1) \& (x_2)}^{\text{SAT expression}}$$

Clause

# 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  $\rightarrow (\neg b_1)$
- ▶ Convert dependencies and conflicts to disjunct 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:

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

# 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:

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

- ▶ 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:

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

# The solving of SAT problem

Some rules to follow to speed up SAT problem solving.

- ▶ Trivial propagation - solve unary clauses
- ▶ Unit propagation - solve clauses with only a single unsolved variable
- ▶ DPLL algorithm backtracking.
- ▶ Package specific assumptions.

# SAT problem propagation

- ▶ Trivial propagation - direct install or delete rules

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

# SAT problem propagation

- ▶ Trivial propagation - direct install or delete rules

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

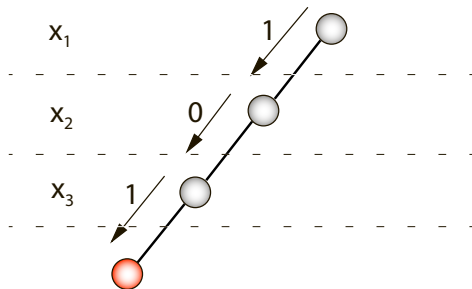
- ▶ Unit propagation - simple depends and conflicts

$$\begin{array}{ccccccc} \text{Dependency} & & true & & false & & \text{Conflict} \\ \overbrace{(\neg A \parallel B)} & \& & \underbrace{(A)} & \& & \underbrace{(\neg C)} & \& & \overbrace{(\neg A \parallel \neg D)} \\ B \rightarrow true & & & & & & & & & D \rightarrow false \end{array}$$



# DPLL algorithm

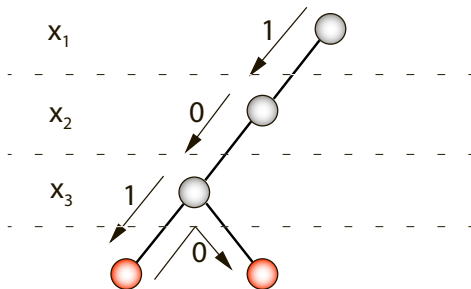
DPLL is proved to be one of the efficient algorithms to solve SAT problem (not the fastest but more simple than alternatives).





# 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

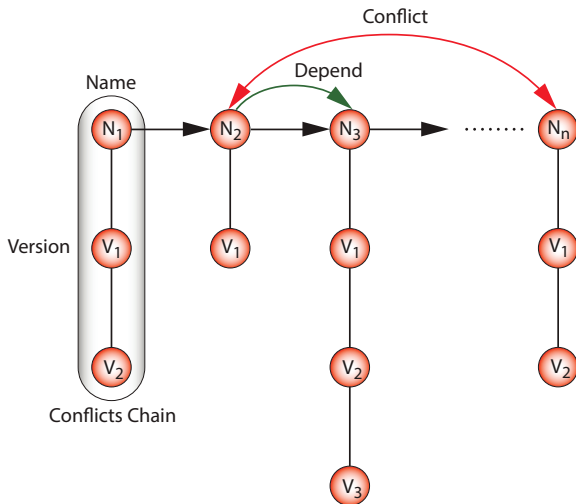
Pure SAT solvers cannot deal with package management as they do not consider several packages peculiarities:

- ▶ 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)

These options also improve SAT performance providing a good initial assignment.

# Packages universe

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



# Package management task

- ▶ A request is splitted to install/upgrade and delete requests which could be passed simultaneously to the solver
- ▶ A conflicts between packages are detected with a repository creation
- ▶ All depends, reverse and conflicts of the requested packages are analyzed and the package universe is created
- ▶ Each package is defined by its name and the digest of significant fields (version, options and so on)

# 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

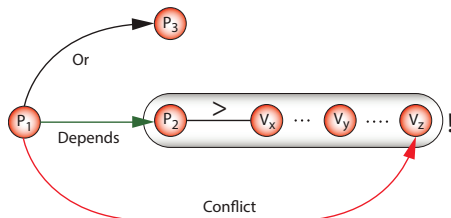
# Perspectives

- ▶ Using pkg solver for ports management
- ▶ Better support of multiple repositories
- ▶ Test different solvers algorithms using CUDF
- ▶ New dependencies and conflicts format
- ▶ Provides and alternatives

# New dependencies format

*libblah*  $\geq 1.0 + \text{option}_1, + \text{option}_2 || \text{libfoo}! = 1.1$

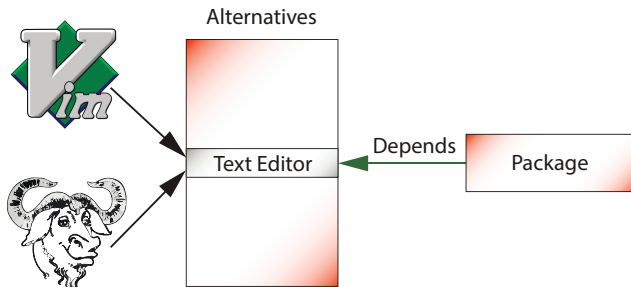
- ▶ Can depend on normal packages and virtual packages (provides)
- ▶ Easy to define the concrete dependency versions
- ▶ Alternative dependencies





# Alternatives

- ▶ Used to organize packages with the same functionality (e.g. web-browser)
- ▶ May be used to implement virtual dependencies (provides/requires)



# Existing issues to be solved prior to 1.3

- ▶ A solver cannot find install candidates for non-automatic top level packages (those without reverse depends)
- ▶ Package upgrade is performed improperly (need to rename, install and unlink)
- ▶ Minor issues and crashes



freeBSD

Thank you for your  
attention!

*Questions?*

[vsevolod@FreeBSD.org](mailto:vsevolod@FreeBSD.org)



freeBSD