

# FreeBSD package management system

Vsevolod Stakhov  
vsevolod@FreeBSD.org



FreeBSD

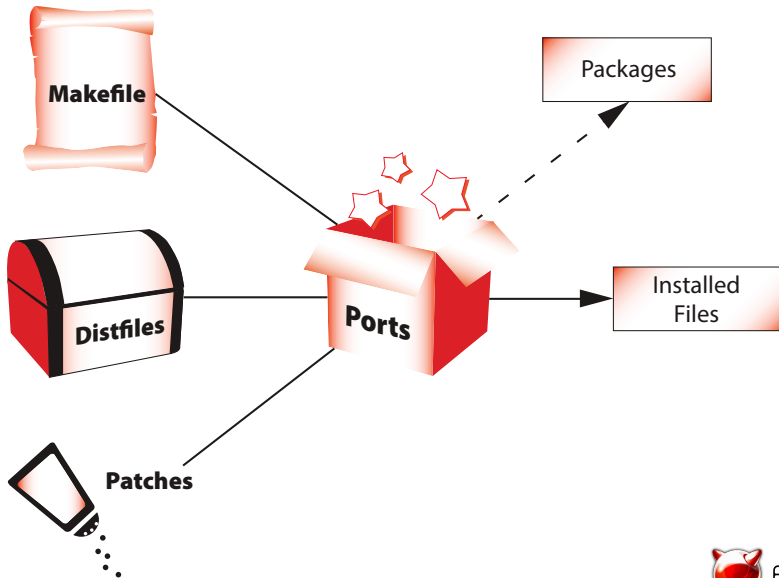
ruBSD conference December 14, 2013

# Ports and packages

Ports is the comprehensive system of source packages.

- ▶ Mature.
- ▶ Clear and well defined.
- ▶ Simple (sometimes not).
- ▶ Configurable.

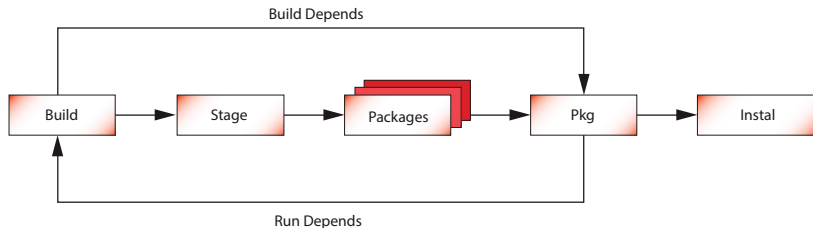
# Ports before pkg



# Disadvantages of the old architecture

- ▶ Make cannot handle complex packages relationships.
- ▶ Complicated upgrade procedure (hard to keep up-to-date).
- ▶ Hard to migrate between releases.
- ▶ Long build time.

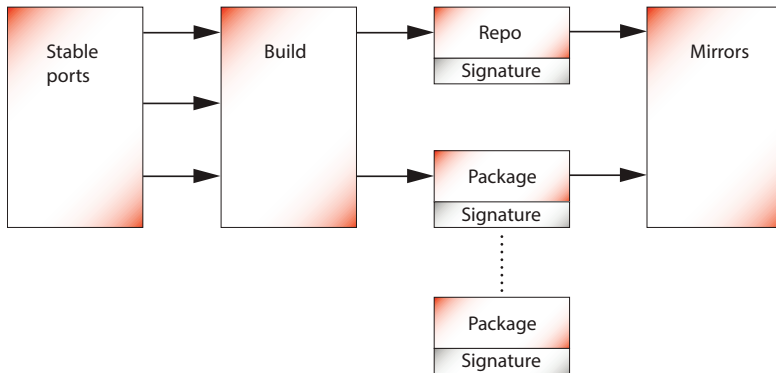
# Planned ports and pkg interaction



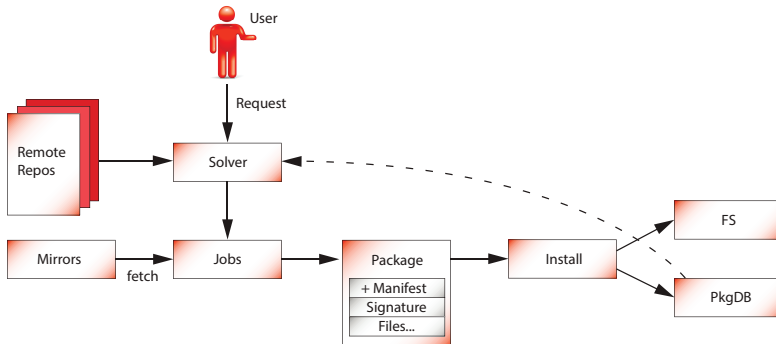
# Ports and packages

- ▶ Ports are used to build packages.
- ▶ Dependencies are resolved by pkg, not make.
- ▶ Stable branch of ports has an appropriate stable branch of packages.
- ▶ Encourage users to install software from binary packages.
- ▶ But do not prevent them from building custom packages from the ports.

# Repositories creation



# Pkg architecture





# The current problems with pkg

- ▶ Legacy ports support (with no staging, for example).
- ▶ Plain dependencies style.
- ▶ Naive solver.

# The problems of the solver in pkg

- ▶ Absence of conflicts resolving/handling;
- ▶ No alternatives support;
- ▶ Can perform merely a single task: install, upgrade or remove, so install task cannot remove packages for example.

# The existing work

There are many examples of solvers, for example:

- ▶ `libsolv` - the complete solver and package management library;
- ▶ Apt solvers interface;
- ▶ Mancoosi - a European research project that compares and study different solvers;

# External solvers

To interact with an external solver we have chosen CUDF format used in the Mancoosi project:

```
package: devel/libblah  
version: 1  
depends: x11/libfoo
```

```
package: security/blah  
version: 2  
depends: devel/libblah  
conflicts: security/blah-devel
```

# 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

$$(x_1 \parallel \neg x_2 \parallel x_3) \& (x_3 \parallel \neg x_1) \& (x_2)$$



# 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 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:

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

## 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:

$$(\neg B_1 \parallel \neg B_2) \& (\neg B_1 \parallel \neg B_3) \& (\neg B_2 \parallel \neg B_3)$$

# 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;
- ▶ Conflicts learning - if we assign some free variable and detect a conflict during unit propagation, we can fallback and learn that this variable must be negated;
- ▶ Package specific assumptions.

# SAT problem propagation

- ▶ Trivial propagation - direct install or delete rules.

$$(\neg A \parallel B_1 \parallel B_2) \& (A) \& (\neg C) \& (\neg A \parallel \neg D)$$

Assume  $A$  as *true* and  $C$  as *false*.

# SAT problem propagation

- ▶ Trivial propagation - direct install or delete rules.

$$(\neg A \parallel B_1 \parallel B_2) \& (A) \& (\neg C) \& (\neg A \parallel \neg D)$$

Assume *A* as *true* and *C* as *false*.

- ▶ Unit propagation - simple depends and conflicts.

$$(\neg A \parallel B) \& (A) \& (\neg C) \& (\neg A \parallel \neg D)$$

Assume *B* as *true* and *D* as *false*, as *A* is true and it depends on *B* and conflicts with *D*.

# Conflicts driven learning

To handle alternatives it is required to test all variables unassigned:

1. full depth-first enumeration of possible values;
2. fallback if a conflict found;
3. remember which assignment caused conflict;
4. make negative assignment for the learned variable and go to the first step.

# Package specific assumptions

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

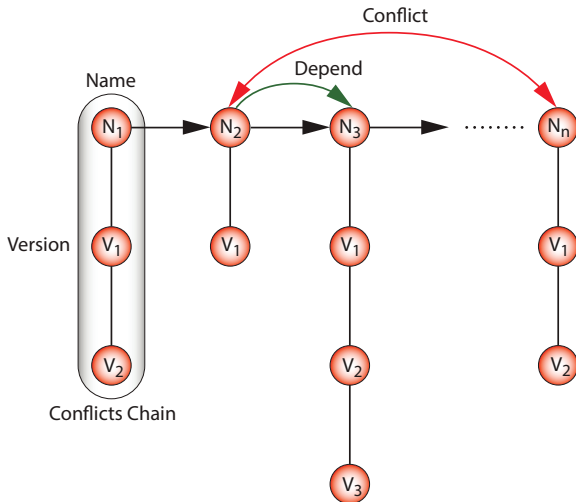
- ▶ try to keep installed packages (if no direct conflicts);
- ▶ do not install packages if they are not needed;
- ▶ prefer high priority packages and repositories over low priority ones.

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

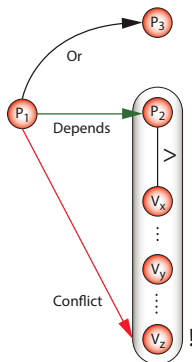
- ▶ Pkg may pass the formed universe to an external CUDF solver:
  - ▶ convert versions;
  - ▶ format request;
  - ▶ parse output.
- ▶ Alternatively the internal SAT solver may be used:
  - ▶ convert the universe to SAT problem;
  - ▶ formulate request;
  - ▶ ???
  - ▶ PROFIT

# 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.

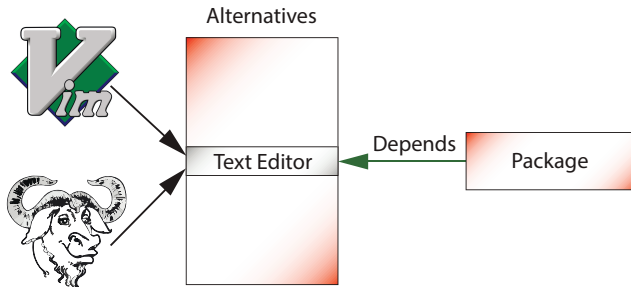
# New dependencies format

- ▶ 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).





FreeBSD

Thank you for your attention!  
*Questions?*

