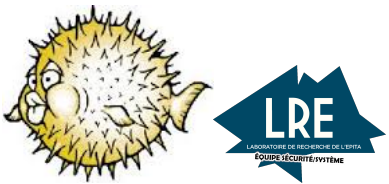
Recent advances in **pkg_tools**

Marc Espie <espie@openbsd.org>, <marc.espie@epita.fr>



September 14, 2026

WIP: three easy (?) pieces

I hoped to finish it this summer, but thanks to the heatwave: brain-melt.
Time willing

- TEX Live vs. `update_plist`,
- ocaml weird things,
- shared items across package updates...

WIP: three easy (?) pieces

I hoped to finish it this summer, but thanks to the heatwave: brain-melt.
Time willing

- TeX Live vs. `update_plist`,
- ocaml weird things,
- shared items across package updates...
- ?

Edd

- I've known Edd for about 10 years, he's a *very* nice guy.
- ... He's also a bit of an idiot: he's been tricked into becoming the maintainer of TEX Live and has been doing it all this time
- ... even though he no longer actively uses it !

TEX Live

- Roughly 40K files spread over 5 subpackages,
- Very annoying to update,
- Custom pipeline: TLPDB[a] $\rightarrow$ hints $\rightarrow$ packing-lists,
- Completely outside the scope of `update_plist` !

[a] TEX Live Plain text database

Why is this a problem ?

- Lots of work each time TEX live updates,
- creates slightly broken packing-lists (common directories are not right),
- lots of special cases.

pipeline details: the update_plist_hints.py script I

```
#!/usr/bin/env python3
from tlpdb import PkgPartSpec, FileKind, Parser

MANS_INFOS_RE = re.compile(r'(man\/man[0-9]\/.*[0-9]|info\/.*\.info)$')
MOVE_MANS_INFOS_RE = re.compile(r'^share/texmf-dist/doc/(man|info)/')

# Individual files that conflict with other ports.
CONFLICT_FILES = set([
    # Comes from print/ps2eps.
    # ps2eps is included in a larger texlive package called pstools, so it
    # cannot be excluded by package. We disable this in the base build at
    # configure time.
    "man/man1/bbox.1",
    "man/man1/disdvi.1",
    # We have a psutils port, but tex live's version includes some other
    # stuff (perl scripts). Best package those up.
```

```
17      # a bunch of perl scripts.
18      "man/man1/epsffit.1",
19      "man/man1/extractres.1",
20          ...
21  ])
22
23
24      buildset_pkgs = [
25          # Barebones of a working latex system
26          "scheme-basic",
27          # textproc/dblatex
28          "anysize", "appendix", "changebar",
29          "fancyvrb", "float", "footmisc",
30          "jknapltx", "multirow", "overpic",
31          "stmaryrd", "subfigure",
32          "fancybox", "listings", "pdfpages",
```

pipeline details: the update_plist_hints.py script III

```
33          "titlesec", "wasysym",
34          ...
35          ]
36  def should_comment_file(f, commented_files):
37      return (
38          # Stuff provided by other ports.
39          f in commented_files or
40          # Windows junk
41          re.match(r'.*\.([Ee][Xx][Ee]|[Bb][Aa][Tt])$', f) or
42          # no win32 stuff, but should probably keep win32 images in tl docs.
43          ("win32" in f and "doc/texlive" not in f) or
44          "mswin" in f or
45          f.endswith(".vbs") or
46          ...
47      )
```

The hints

```
1 bin/amstex        -main
2 # bin/asymptote
3 share/texmf-dist/doc/alepha/base/readme.txt        -docs
```

The plan

- add a `-h` hints option
- add a `-t` exclude option

The snag

Reconstituting full path has to be delayed

- `update_plist` runs
  `update-plist [global_options] [subpkg1] - [subpkg2] - ...` and `-p prefix` is specific to each package, also the full plist name linked to `SUBPACKAGE` like `-main`
  → so it has to postprocess the hints **after** parsing options.
- Avoiding other ports is simpler, as we already have the full framework in order to trim common subdirectories from dependencies → it's almost the same but we're trimming everything.
- a few odds and ends with privilege separation: hints file is created after staging, so need to be owned by _pbuild, and needs to be rebuilt after staging → two internal bsd.port.mk variables need to be exposed.

While coming to the conference, I had a big problem: `minted` was broken.
Turns out the move to python 3.14 broke `latexminted` the extra glue that runs the python code from `pdflatex`.
It's been fixed upstream, we're late by one year, but Edd hasn't got around to it yet.

It's not over

- Thanks to a remark from Stuart Henderson, I discovered something **else** is not working.
- Normally `update_plist` should put files in the right packing-list according to the nearest file, but it didn't work.
- It's an OO issue: I got a `known_directory` method that ties approximate locations, but once I find an exact location, I delete the approximate location.
- and of course, that's done for all file-like objects, which include directories. So I'm wiping out my own tracks.
- Not noticed because it is a "soft failure".
- Easy to fix, costly to keep it working.
- Not the first time this happened (had a similar issue with the LRU cache).

ocaml vs. update_plist

- Depending on architecture, ocaml creates different binaries:
- $\rightarrow$ classify those files (easy),
- $\rightarrow$ creates fragments (should be easy).

The recognizer I

```
1  sub classes($)
2  {
3          return (qw(OpenBSD::FS::File::Directory OpenBSD::FS::File::Rc
4                  OpenBSD::FS::File::Desktop
5                  OpenBSD::FS::File::Glib2Schema
6                  OpenBSD::FS::File::IbusComponent
7                  OpenBSD::FS::File::PkgConfig
8                  OpenBSD::FS::File::MimeInfo
9                  OpenBSD::FS::File::LoginClass
10                 OpenBSD::FS::File::Icon
11                 OpenBSD::FS::File::IconTheme
12                 OpenBSD::FS::File::Ocaml::Cmx
13                 OpenBSD::FS::File::Ocaml::Cmxa
14                 OpenBSD::FS::File::Ocaml::Cmxs
15                 OpenBSD::FS::File::Subinfo OpenBSD::FS::File::Info
16                 OpenBSD::FS::File::Dirinfo OpenBSD::FS::File::Manpage
```

The recognizer II

```
17                  OpenBSD::FS::File::Library OpenBSD::FS::File::Plugin
18                  OpenBSD::FS::File::StaticLibrary
19                  OpenBSD::FS::File::Binary OpenBSD::FS::File::Font
20                  OpenBSD::FS::File));
21  }
22
23  # $class->recognize($filename, $fsobj, $data):
24  #          called for each class in order until the "default" file
25  #         some retrieved data such as file's contents are cached for
26  #          efficiency
27  sub recognize($, $, $, $)
28  {
29          return 1;
30  }
31
32  package OpenBSD::FS::File::Ocaml::Cmx;
```

The recognizer III

```
33  our @ISA = qw(OpenBSD::FS::File);
34
35  sub recognize($, $filename, $, $)
36  {
37          return $filename =~ m/\.cmx$/;
38  }
39
40  sub element_class($)
41  {
42          "OpenBSD::PackingElement::File::Ocaml::Cmx"
43  }
44
45  sub tweak_other_paths($self, $fs, $files)
46  {
47          my $o = $self->path;
48          $o =~ s/\.cmx$/.o/;
```

The recognizer IV

```
49          if (exists $files->{$o}) {
50                  bless $files->{$o}, "OpenBSD::FS::File::Ocaml::o";
51          }
52  }
```

Now we move to the hard part

Two reasons:

- Common code between base and ports,
- code that's too beautiful and badly broken at the same time.

Common code and distribution woes I

```
1  sub read_fragments($self, $state, $plist, $filename)
2  {
3          my $stack = [];
4          my $subst = $state->{subst};
5          my $main = $self->FileClass->new($filename);
6          return undef if !defined $main;
7          push(@$stack, $main);
8          my $fast = $subst->value("LIBS_ONLY");
9
10         return $plist->read($stack,
11             sub($stack, $cont) {
12                 while(my $file = pop @$stack) {
13                         while (my $l = $file->readline) {
14                                 if ($l =~ m/^(\!)?\%\%(.*)\%\%$/) {
15                                         $self->record_fragment($plist, $1, $2,
16                                             $file);
17                                         if (my $f2 = $self->handle_fragment($state,
                                           ↪  $file, $1, $2, $filename)) {
```

Common code and distribution woes II

```
18                                                      push(@$stack, $file);
19                                                      $file = $f2;
20                                              }
21                                              next;
22                                      }
23                                      my $o = &$cont($s);
24                                      if (defined $o) {
25                                              $self->annotate($o, $l, $file);
26                                      }
27                              }
28                      }
29              });
30  }
31
32  sub deduce_name($self, $frag, $not, $p, $state)
33  {
34          my $o = $self->name;
35          my $noto = $o;
```

Common code and distribution woes III

```
36          my $nofrag = "no-$frag";
37
38          $o =~ s/PFRAG\./PFRAG.$frag-/o or
39              $o =~ s/PLIST/PFRAG.$frag/o;
40
41          $noto =~ s/PFRAG\./PFRAG.no-$frag-/o or
42              $noto =~ s/PLIST/PFRAG.no-$frag/o;
43          unless (-e $o or -e $noto) {
44                  $p->missing_fragments($state, $frag, $o, $noto);
45                  return;
46          }
47          if ($not) {
48                  return $noto if -e $noto;
49              } else {
50                  return $o if -e $o;
51          }
52          return;
53  }
```

A change of API

- I need a better API for fragment names,
- but the code is shared between base and ports,
- ... and they evolve at different speeds !
- Bad idea: two apis in `PkgCreate.pm` ?
- Better: copy the code in `update_plist` then move it API1 $\rightarrow$ API2 (and get it committed),
- move base API1 $\rightarrow$ API2,
- ... wait for it to be distributed everywhere,
- scrape the duplicated code in ports !

Shared items

- the current state is hardcoded.
- Three kinds of items: directories, users, groups.
- directories have some special treatment (font directories and info directories).

The current code I

```
1  sub cleanup($recorder, $state)
2  {
3          my $remaining = find_items_in_installed_packages($state);
4
5          my $h = $recorder->{dirs};
6          my $u = $recorder->{users};
7          my $g = $recorder->{groups};
8
9          for my $d (sort {$b cmp $a} keys %$h) {
10                 $state->progress->show($done, $total);
11                 if (defined $remaining->{dirs}{$d}) {
12                         for my $i (@{$h->{$d}}) {
13                                 $i->reload($state);
14                         }
15                 } else {
16                         wipe_directory($state, $h, $d);
```

The current code II

```
17                  }
18          }
19          while (my ($user, $pkgname) = each %$u) {
20                  next if $remaining->{users}{$user};
21                  if ($state->{extra}) {
22                          $state->system(OpenBSD::Paths->userdel, '--',
23                              $user);
24                  } else {
25                          $state->log("You should also run /usr/sbin/userdel
                            ↪  #1", $user);
26                  }
27          }
28          while (my ($group, $pkgname) = each %$g) {
29                  next if $remaining->{groups}{$group};
30                  if ($state->{extra}) {
31                          $state->system(OpenBSD::Paths->groupdel, '--',
```

```
32                                  $group);
33                  } else {
34                          $state->log("You should also run /usr/sbin/groupdel
                            ↪  #1", $group);
35                  }
36          }
37  }
38
39  sub wipe_directory($state, $h, $d)
40  {
41          my $realname = $state->{destdir}.$d;
42
43          for my $i (@{$h->{$d}}) {
44                  $i->cleanup($state);
45          }
46          if (!rmdir $realname) {
```

The current code IV

```
47                  $state->log("Error deleting directory #1: #2",
48                      $realname, $!)
49                          unless $state->{dirs_okay}{$d};
50                  return 0;
51          }
52          return 1;
53  }
54
55  package OpenBSD::PackingElement::Mandir;
56  sub cleanup($self, $state)
57  {
58          my $fullname = $state->{destdir}.$self->fullname;
59          $state->log->set_context('-'.$self->{pkgname});
60          $state->log("You may wish to remove #1 from man.conf", $fullname);
61          for my $f (OpenBSD::Paths->man_cruft) {
62                  unlink("$fullname/$f");
```

```
63          }
64  }
65
66  package OpenBSD::PackingElement::Fontdir;
67  sub cleanup($self, $state)
68  {
69          my $fullname = $state->{destdir}.$self->fullname;
70          $state->log->set_context('-'.$self->{pkgname});
71          $state->log("You may wish to remove #1 from your font path",
            ↪  $fullname);
72          for my $f (OpenBSD::Paths->font_cruft) {
73                  unlink("$fullname/$f");
74          }
75  }
76
77  package OpenBSD::PackingElement::Infodir;
```

```
78  sub cleanup($self, $state)
79  {
80          my $fullname = $state->{destdir}.$self->fullname;
81          for my $f (OpenBSD::Paths->info_cruft) {
82                  unlink("$fullname/$f");
83          }
84  }
85
86  1;
```

Making that more generic

The challenge

- Sorting items into several bins (type for object plus key for object),
- choosing an order to handle things,
- classifying hierarchy: group directories or not,
- type mishaps: what to do when a directory gets more specialized,
- can't grow too much since it's global.

The benefits

- Can be applied to more items: shells, configuration files...
- "shared" items can be anything that may move between packages, or even be "common" annotations to several packages,
- should allow easier group/user synchronization with dynamic ids.

To do

- Document the new interface,
- make sure it works,
- integrate new objects,
- more tests,
- more code at the end $\rightarrow$ breakage has more consequences.

Why so slow ?

```
1 #! /usr/bin/perl
2 use v5.36;
3 say "!" x 50;
```

- Shared items happen globally at the end (duh!),
- so any breakage there has big consequences.

Why so slow ?

```
1 #! /usr/bin/perl
2 use v5.36;
3 say "!" x 50;
```

- Shared items happen globally at the end (duh!),
- so any breakage there has big consequences.
- This has been in development hell for a year or so.

The mystery section: some context

- The current iteration of these tools is over 20 years and still evolving.
- Some artifacts cause **traction** in the weirdest places.
- Fixing issues often require precise logs and abreviated testcases.

It was the best of languages, it was the worst of languages

- Perl is interpreted, which simplifies deployment.
- Unfortunately it has an abstraction cost, as every added line or indirection costs a bit more !
- Perl has major introspection tools including `Devel::NYTProf`.

The debugging approach I

For testing code, I had to implement a way to break tools:

```
1  #! /usr/bin/perl
2  package Wrapper;
3  require Exporter;
4  our @ISA = qw(Exporter);
5  our @EXPORT = qw(wrap);
6
7  sub wrap($name, $sub)
8  {
9          my $typeglob = caller()."::$name";
10         my $original;
11         {
12                 no strict qw(refs);
13                 $original = *$typeglob{CODE};
14         }
15         my $imposter = sub(@p) {
```

```
16                      return &$sub($original, @p);
17              };
18
19              {
20                      no strict qw(refs);
21                      no warnings qw(redefine);
22                      *{$typeglob} = $imposter;
23              }
24      }
25      1;
```

working like this

The debugging approach III

```
1  package OpenBSD::digest;
2  use Wrapper;
3
4  wrap('digest_file',
5      sub { # Forwarder
6          my $original = shift;
7              if ($_[1] =~ m,/a$,) {
8                  my $self = shift;
9                  my $d = $self->_algo;
10                 $d->addfile("/dev/null");
11                 return $d->digest;
12         } else {
13                 goto &$original;
14         }
15     });
16 1;
```

- This "wrapper" approach is limited to named subs → there must be an entry point for it to be effective.
- There are also source filters in perl (see filter(1)), but they are too slow to be useful.

Thank you

Questions ?