Advanced ports toolkit: near-perfect packing-list generation

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

Epita Research and Development Laboratory

September 22, 2019

- llvm is quietly changing its software license for 9.0 onwards.
- it's going from BSD (perfect) to Apache V2 (waaaay less free)
- it's probably too late to do anything but at least you know.

why is this important

- We're supposed to be BSD
- A big reason (besides technical excellence) is that we believe in the license
- We already got scammed big-time by the FSF with the GPLv3 toolchain
- And now, we are ready to accept Apache V2, which is also a compromise
- let's not be pushovers and be more vocal about licensing
- it's too late for llvm.

A brief history of ports

Binary packages

- we inherited `bsd.port.mk` from FreeBSD
- we converted to "staged installation" (we call that `fake`) 20 years ago
- went from build $\longrightarrow$ install $\longrightarrow$ package to build $\longrightarrow$ package $\longrightarrow$ install

Consequences

- Reproducible installation because we have every file
- Need some kind of MANIFEST for stuff that goes into packages
- ... generated with ad-hoc tools

Two sides

make plist

The original `make_plist` predates staging areas:

- 150 lines of perl that looks under /usr/local for new files.
- Very simple and stupid.
- support for manual calls to ldconfig, install-info
- uses mtree to avoid recreating directory hierarchies
- gets much faster with staging area

pkg create

And `pkg_create` gets its plist from a sed script Horrible stuff like:

```
sed -e '/^!%%${_i}%%$$/d' -e '/^%%${_i}%%$$/r${PKGDIR}/PFRAG.${_i}'
```

More features I

The package tools were replaced with a perl ABI and perl tools

```
# This is the basic class, which is mostly abstract, except for
# create and register_with_factory.
# It does provide base methods for stuff under it, though.
package OpenBSD::PackingElement;

# concrete objects
package OpenBSD::PackingElement::Object;
our @ISA=qw(OpenBSD::PackingElement);

sub cwd
{
        return ${$_[0]->{cwd}};
}

sub absolute_okay() { 0 }
```

More features II

```perl
sub compute_fullname
{
	my ($self, $state) = @_;

	$self->{cwd} = $state->{cwd};
	$self->set_name(File::Spec->canonpath($self->name));
	if ($self->name =~ m|^/|) {
		unless ($self->absolute_okay) {
			die "Absolute name forbidden: ", $self->name;
		}
	}
}

sub make_full
{
	my ($self, $path) = @_;
	if ($path !~ m|^/|o && $self->cwd ne '.') {
```

More features III

```
                        $path = $self->cwd."/".$path;
                        $path =~ s,^//,/,;
                }
                return $path;
        }
        [...]
        # concrete objects with file-like behavior
        package OpenBSD::PackingElement::FileObject;
        our @ISA=qw(OpenBSD::PackingElement::Object);
        # exec/unexec and friends
        [...]
        package OpenBSD::PackingElement::Action;
        our @ISA=qw(OpenBSD::PackingElement::Object);
        [...]
        package OpenBSD::PackingElement::Lib;
        our @ISA=qw(OpenBSD::PackingElement::FileBase);

```

More features IV

```perl
sub keyword() { "lib" }

sub mark_ldconfig_directory
{
	my ($self, $state) = @_;
	$state->ldconfig->mark_directory($self->fullname);
}
```

So make-plist was updated

It became able to read and generate more information... and grew to 300 lines.
Copying annotations like `@mode`, `@owner`, and file types like `@lib`.

Enter FLAVORS and MULTI PACKAGES I

Motivation

- Flavors are very simple, they're just a way to record compilation options into packages.
- This however results in more fragments... and variables.
- So `pkg_create` learnt to use them directly, exit the sed stuff

Enter FLAVORS and MULTI PACKAGES I

```
@comment $OpenBSD: PLIST,v 1.2 2019/05/18 15:00:41 espie Exp $
@option no-default-conflict
@option is-branch
@conflict nethack->=${BASEV}
@newgroup ${GAMEGRP}:806
@comment bin/license
@comment bin/nethack
bin/nethack-${V}
bin/recover-${V}
@mode 0775
@group ${GAMEGRP}
${GAMEDIR}/
@mode
${GAMEDIR}/license
@mode 2555
@bin ${GAMEDIR}/nethack
@bin ${GAMEDIR}/recover
@mode
${GAMEDIR}/nhdat
@mode 0664
@comment no checksum
${GAMEDIR}/logfile
${GAMEDIR}/perm
@comment no checksum
${GAMEDIR}/record
@mode 0664
@comment no checksum
```

Enter FLAVORS and MULTI PACKAGES II

```
${GAMEDIR}/xlogfile
@mode 0774
${GAMEDIR}/save/
@mode
${GAMEDIR}/symbols
${GAMEDIR}/sysconf
@man man/man6/nethack-${V}.6
@man man/man6/recover-${V}.6
!%%no_x11%%
```

Enter FLAVORS and MULTI PACKAGES

Motivation

- Compile once, package several times.
- No need to hack at makefiles/configure to just build part of some software.
- Create subpackages based on dependencies and/or sizes.
- ... but how do we sort files into the right subpackage

Enter FLAVORS and MULTI PACKAGES I

```
# $OpenBSD: Makefile,v 1.82 2019/07/12 20:47:14 sthen Exp $

COMMENT-main =                  Japanese input method
COMMENT-dict =                 dictionaries for Japanese Wnn
COMMENT-ko =                  Korean input method
COMMENT-kodict =         dictionaries for Korean Wnn
[...]

DISTNAME =          Wnn4.2

CATEGORIES =          japanese

MULTI_PACKAGES =          -main -dict -zh -zhdict -ko -kodict -xwnmo -data

PKGNAME-main =                 ja-Wnn-4.2
```

Enter FLAVORS and MULTI PACKAGES II

```
PKGNAME-dict =                 ja-Wnndict-4.2
PKGNAME-ko =                 ko-Wnn-4.2
PKGNAME-kodict =          ko-Wnndict-4.2

WNNDICBASE =         /var
WNNDICDIR =         ${WNNDICBASE}/dict/Wnn
WNNBASE =         ${LOCALBASE}
SUBST_VARS =         WNNBASE

PREFIX-dict =                ${WNNDICBASE}
PREFIX-kodict =                ${WNNDICBASE}

[...]

.include <bsd.port.mk>
```

So make-plist grew I

1100 lines in 2011! Horrible code:

```
#! /usr/bin/perl
# TODO
# - multi-packages with inter-dependencies still are not 100% correct with
# respect to common directories.

my %prefix;
my %plistname;
my %mtree;
my @subs;
my $baseprefix=$ENV{PREFIX};
my $shared_only;
my $make = $ENV{MAKE};
my $portsdir = $ENV{PORTSDIR};
```

So make-plist grew II

```
[...]

sub parse_arg
{
        my $_ = shift;
        if (m/^DEPPATHS(-.*?)\=/) {
                $mtree{$1} = build_mtree($1, $');
                return;
        }
        if (m/\=/) {
                $subst->parse_option($_);
        }
        if (m/^\^PREFIX(\-.*?)\=(.*)\/?$/) {
                $prefix{$1} = $2;
```

So make-plist grew III

```
		} elsif (m/^PLIST(\-.*?)\=/) {
			$plistname{$1} = $';
		}
}

# Fragments are new PackingElement unique to make-plist and pkg_create,
# to handle %%thingy%%.
# (and so, make-plist will use a special PLIST reader)


# Method summary:
# add_to_mtree: new directory in dependent package
# add_to_haystack: put stuff so that it can be found on the FS
# copy_extra: stuff that can't be easily deduced but should be copied
# tag_along: set of items that associate themselves to this item
```

So make-plist grew IV

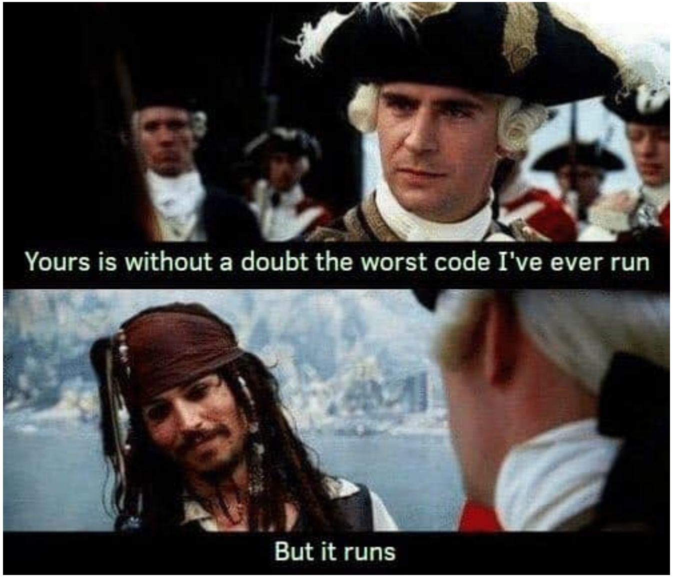
```
#   (e.g., @exec, @unexec, @sample...)
# clone_tags: copy tagged stuff over.
# deduce_fragment: find fragment file name from %%stuff%%

# note $plist->{nonempty}: set as soon as a plist holds anything
# but a custag.

[...]
```

Oh crap

When the going gets tough

Bad smells

- mismash of `DESTDIR`, `PREFIX` ...
- did not cope well with options
- ad-hoc code to make it work better
- each option took longer to implement


Yours is without a doubt the worst code I've ever run
But it runs

Pull the plug

So around 2015, I decided to stop implementing new stuff
And started thinking about rewriting things
... People had started writing scripts to "help" `make_plist`

The long wait

It took me a few years to get it into shape.
The basic structure was much easier, since I knew better.
But getting it to do correctly everything that the old tool did (incorrectly) took a long time!
Should I have done better ?

Parsing options

BOOrn again

- `make_plist` had an ad-hoc parser that did all of `pkg_create` work again.
- ... but `pkg_create` was now OO perl
- ... so the new `update_plist` could reuse `Pkg_Create.pm`
- synopsis would be `update_plist <options> -- pkg_create_args`
- and since *each* `pkg_create` synopsis was `pkg_create <options> pkg_name` it would work for multi-packages: just let the `Pkg_Create.pm` parser stop after the first "real" argument

Parsing options I

To summarize

- One single `Pkg_Create.pm` parser
- that one has a `$state` that contains all the info
- for `update_plist`, we have a global `$state` with the `update_plist` options
- ... and one `$state` for each package in the multi-package set
- e.g., create a `Pkg_Create` object that just does this:

Parsing options I

```perl
sub process_next_subpackage
{
	my $self = shift;
	my $r = PlistReader->new;

	my $s = PlistReader::State->new('update-plist', $self->{state});
	$r->{state} = $s;
	$s->handle_options;
	$s->{opt}{q} = 1;
	$r->{base_plists} = $s->{contents};
	my $pkg = shift @ARGV;

	$r->olist->set_pkgname($pkg);
	$r->nlist->set_pkgname($pkg);
```

Parsing options II

```
        $self->{state}->say("Reading existing plist for #1", $pkg)
            unless $self->{state}{quiet};
        $r->read_all_fragments($s, $r->olist);
        # add the cwd to new list as well!!!
        OpenBSD::PackingElement::Cwd->add($r->nlist, $s->{prefix});
        $r->add_extra_info($r->olist, $s);
        push(@{$self->{lists}}, $r);
}
```

Global structure

Checklist

- `parse_args` get the old plist(s)
- `known_objects` mark old files and directory locations
- `scan_fake_dir` just the old find with pattern recognition
- `handle_annotations` try to tie things to paths
- `copy_objects` actually create the new plists in correct order
- `add_missing_tags` global stuff needed for some files
- `add_delayed_objects` all about cwd
- `strip_dependency_directories`
- `adjust_final` whatever we didn't do

Variable substitutions

- Cool thing about OO, you can specialize things.
- There's an empty `annotate` sub in `Pkg_Create.pm`
- In `update_plist` it can match original lines with subst versions.

How to unsubst

- there are 400 lines in `ReverseSubst.pm`
- basic algorithm is to sort a subpackage variable by reverse length, and try to reverse substitute the longest strings first...
- ... and there are 5 options related to specific usage
  - variables that only occur at start of path (`/etc`)
  - variables that only occur at end of paths (`.pyc`)
  - variables that shouldn't be added (only allowed if already there)
  - variables that expand to nothing
  - the python weird stuff

A typical python plist I

```
@comment $OpenBSD: PLIST,v 1.1.1.1 2017/05/12 10:08:32 edd Exp $
lib/python${MODPY_VERSION}/site-packages/pyuv/
lib/python${MODPY_VERSION}/site-packages/pyuv/__init__.py
${MODPY_COMMENT}lib/python${MODPY_VERSION}/site-packages/pyuv/${MODPY_PYCACHE}/
lib/python${MODPY_VERSION}/site-packages/pyuv/${MODPY_PYCACHE}__init__.${MODPY_PYC_MAGIC_TAG}pyc
lib/python${MODPY_VERSION}/site-packages/pyuv/${MODPY_PYCACHE}_version.${MODPY_PYC_MAGIC_TAG}pyc
lib/python${MODPY_VERSION}/site-packages/pyuv/_cpyuv.so
lib/python${MODPY_VERSION}/site-packages/pyuv/_version.py
```

Details everywhere

- if several variables expand to the same value, we don't know, unless some variables are ignored. For instance, we prefer LOCALBASE to PREFIX
- there is a `MODPY_COMMENT` that expands to nothing in a python2 port and creates a python cache directory in python3.
- ... it is a comment because `MODPY_CACHE` vanishes, and we have duplicate dirs, and `pkg_create` does not like that.

```perl
# default backsubstitution and writing.
sub write_backsubst
{
	my ($o, $file, $p) = @_;

	if (defined (my $s = $o->{candidate_for_comment})) {
		if ($p->{stash}{$s} > 1) {
			$o->{prepared} =
			    $p->subst->{maybe_comment}.$o->{prepared};
		}
	}
	print {$file->fh} $o->{prepared}, "\n";
}
```

The FS parser

Basics

Really simple code, we just have a set of recognizers based on

- names
- contents

that create their own classes, which later get mapped to plist types.
... this can even be used for the subst handler (libraries...) and tags.

The fineprint

There is specific code for resolving symlinks, thanks to DESTDIR.
Recognizers have access to an extra stash (`data`) so they can leave useful info for later recognizers (for example, running `objdump`)
In theory, this could be located in individual files.

Order please I

How do we sort everything

- I decided to sort on actual filenames...
- ... so that adding a variable would not change everything
- this does change manually generated lists
- there are annotations that "tag along" with the nearest file (`@sample`, `@comment no checksum`)

Order please I

```
@comment $OpenBSD: PLIST-main,v 1.2 2019/06/07 20:24:04 sthen Exp $
@option no-default-conflict
@option is-branch
@conflict php->=7.3,<7.4
@extraunexec rm -f ${SYSCONFDIR}/php-${PV}.sample/*
@extraunexec rm -f ${SYSCONFDIR}/php-fpm.d/*
@mode 1700
@owner www
@group www
@sample ${CHROOT_DIR}/tmp/
@mode
@owner
@group
@rcscript ${RCDIR}/php${SV}_fpm
bin/phar-${PV}
@bin bin/php-${PV}
bin/php-config-${PV}
bin/phpize-${PV}
lib/php-${PV}/
lib/php-${PV}/modules/
lib/php-${PV}/modules/opcache.so
@man man/man1/phar-${PV}.1
@man man/man1/php-${PV}.1
@man man/man1/php-config-${PV}.1
@man man/man1/phpize-${PV}.1
@man man/man8/php-fpm-${PV}.8
@bin sbin/php-fpm-${PV}
share/doc/pkg-readmes/${PKGSTEM}
```

Order please II

```
share/examples/php-${PV}/
@sample ${SYSCONFDIR}/php-${PV}.sample/
@sample ${SYSCONFDIR}/php-fpm.d/
share/examples/php-${PV}/opcache.ini
@sample ${SYSCONFDIR}/php-${PV}.sample/opcache.ini
share/examples/php-${PV}/php-fpm.conf
@sample ${SYSCONFDIR}/php-fpm.conf
share/examples/php-${PV}/php.ini-development
share/examples/php-${PV}/php.ini-production
@sample ${SYSCONFDIR}/php-${PV}.ini
share/php-${PV}/
share/php-${PV}/build/
share/php-${PV}/build/Makefile.global
share/php-${PV}/build/acinclude.m4
[...]
share/php-${PV}/include/sapi/cli/
share/php-${PV}/include/sapi/cli/cli.h
@sample ${SYSCONFDIR}/php-${PV}/
```

Order please

Sorting through mail

- comments are a bit tricky
- if they correspond to real files they're sorted with them
- some comments are not really comments (cvs tags, no checksum)...
- otherwise, they just tag along with the nearest item in the old plist
- this is the most frustrating part: it will move things a bit sometimes

Order please

State comes last

- There are `@mode`, `@owner` annotations
- ... it comes last
- might create back and forth
- but the file system “stays” sorted

Implementing new stuff

- we now have `@define-tag` and `tag` for file-specific handling
- the file system scanner creates subclasses of file with the right type.
- they do not get specific annotations
- ... but `update_plist` visits and every item and notices special files.
- ... so we can add the `@tag` to the generated list

PRIVSEP and pkglocate

PRIVSEP

- `update_plist` starts its life as root, with two users to go to
- the file system scanner runs as `_pbuild`
- the plist writer runs as `user`

pkglocate

- File systems entries are run against `pkglocate` to check for collisions.
- If those collisions aren't registered, we warn.
- This is rather slow for large plists, so this runs in parallel by default.

End results

- Thanks to yandex, I have big machines to test things on
- so I was able to verify that plists didn't change
- also a huge improvement for multi-packages
- ended up having per-port options (`UPDATE_PLIST_ARGS`, well mostly for python) and user options (`UPDATE_PLIST_OPTS`).

Side note

`https://github.com/marcespie/openbsd-backgrounds`

Thank you for your time

Questions ?