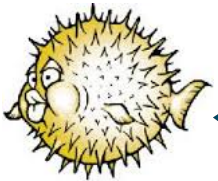
Double feature:
To cache or not to cache making pkg_add faster
and
How I learnt not to worry and use 5.36

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



September 19, 2023

The story so far (2022)

I wrote `pkg_add` a long time ago. Historically, we do "just in time" updates.

- open new package and peek at meta information
- decide whether we want to update
- if so, extract the new package, then delete the old one
- if not, just close the connection and move to the next one

However for complexity reasons, the connection stuff is handled through `ftp(1)` (*and has been privilege separated for a few years*)

The meta information

We got structured information (packing-lists) that looks like this:

```
1  @pkgpath x11/dbus
2  @newgroup _dbus:572
3  @newuser _dbus:572:_dbus::dbus user:/nonexistent:/sbin/nologin
4  @extra ${SYSCONFDIR}/machine-id
5  @rcscript ${RCDIR}/messagebus
6  @bin bin/dbus-cleanup-sockets
7  @bin bin/dbus-daemon
8  @bin bin/dbus-launch
9  @bin bin/dbus-monitor
10 @bin bin/dbus-run-session
11 @bin bin/dbus-send
12 @bin bin/dbus-test-tool
13 [... more files]
```

This is the source information for the `dbus` *package, telling us it requires some user/groups, has a service start-up script, and contains a bunch of files*

The whole story I

After going through `pkg_create`, the full *packing-list* looks more like

```
1  @name dbus-1.14.0v0
2  @version 8
3  @comment pkgpath=x11/dbus,-main ftp=yes
4  @arch amd64
5  +DESC
6  @sha TYbBC2oO7XXOXqnQOFU6qikEuiN+fqoN2azXrJA9jJg=
7  @size 448
8  @pkgpath x11/dbus
9  @wantlib X11.18.0
10 @wantlib c.96.1
11 @wantlib execinfo.3.0
12 @wantlib expat.14.0
13 @wantlib pthread.26.1
14 @wantlib xcb.4.1
15 @newgroup _dbus:572
16 @newuser _dbus:572:_dbus::dbus user:/nonexistent:/sbin/nologin
17 @cwd /usr/local
```

The whole story II

```
18 @extra /etc/machine-id
19 @rcscript /etc/rc.d/messagebus
20 @sha G8InGFO+lEOiPUMpXqicxPO1KEkofHOguRhxV9sMXHk=
21 @size 172
22 @ts 1653570364
23 @bin bin/dbus-cleanup-sockets
24 @sha lew9j03YckJ1VnMPtypbKh1k1eedAXgwCvYU3hE44jU=
25 @size 13318
26 @ts 1653570364
27 [... more files]
```

Structured information

- a *packing-list* is a structured object which has constructors. Most often it starts as
  ```
  my $plist = OpenBSD::PackingList->from_file("filename");
  ```
- objects (*packing elements*) can be added to it using the right method:
  ```
  OpenBSD::PackingElement::Wantlib->add($plist, $w);
  ```
- some complex objects can have a multiline representation, like files:
  - name
  - extra modes
  - checksum
  - timestamp
  - ownership

OO properties

- there's a whole hierarchy of objects: anything file-system related is a `FileObject`, annotations are `Meta`, anything depend-related is a `Depend`
- objects are emitted in a specific order: first all the meta information, then the actual objects (in order)
- most operations happen as visitors on the packing-list
- there are specialized scanners that take advantage of the text structure of the packing-list to avoid reading it all

```
1  sub DependOnly($fh, $cont)
2  {
3          while (<$fh>) {
4                  if (m/^\@(?:libset|depend|wantlib|define-tag)\b/o) {
5                          &$cont($_);
6                  # XXX optimization
7                  } elsif (m/^\@(?:newgroup|newuser|cwd)\b/o) {
8                          last;
9                  }
10         }
11 }
```

Speed

In order to decide whether to update a package

- we look up all packages that have the same name with a different version number
- we open every package to see whether it's a valid candidate
- we filter the ones we don't want
- pathological case: autoconf. We have a branch for each version, which means 17 packages to consider.
- ... and we decide to update

Slower and slower

- for a long time, the network was slow, bandwidth-wise, so opening lots of files was not a big issue
- actually properly closing was an issue with ftp: premature closing requires full telnet support (with "attention" commands")
- and so I had to fix ftp-proxy back in the day
- ... but recently, latency is more of an issue, most people have lots of bandwidth, and so does our current setup

CDN for the masses

We got a CDN

- ftp is dead, long live http(s).
- establishing connections might be a bit slow
- the cdn first gives you a redirect which means two connections
- we've parsed the redirect from the start, to make sure an update connects to exactly one mirror (*)
- bandwidth is not an issue, latency is

Shearing

- When the mirrors update, it's not instant: one snapshot is 64G.
- Safeguards (like library versions) will be verbose but protect us.
- Todo: make them less verbose and more useful

https ? not such a good idea

- http connection establishment is 1 $\frac{1}{2}$ RTT
- https is 2 $\frac{1}{2}$ RTT at best !
- we did implement session resumption (with fun results)
- TLS 1.3 should help a lot (not available at the time)
- ... so updating is slow, because we establish lots of connections
- also, signatures

Signatures ?

```
1  untrusted comment: verify with openbsd-69-pkg.pub
2  RWSG2ib5ZXSfQTrcxxj+A9b6oeFI/OiJVB49nvIs+UPIull+Mk/BclTXRuG4a+XbnyoiZffDILfP58BNelKOyMjZNEE
3  date=2021-02-26T23:06:32Z
4  key=/etc/signify/openbsd-69-pkg.sec
5  algorithm=SHA512/256
6  blocksize=65536
7
8  9d61ddfc76218e7c3745bd942a29725ff1bc651f64af27a450da33a73f292d69
9  8621c7932e29c838783177287fc5779186c854b35eaa541e787979f78288c2a6
10 d895cc173cb9058341bbcbe6abe3c018b915eb9218fd65c31f490f9af9c11041
11 9895735d7a109e497ef3f616f35938ae4d6e66f851f038ba50aa2a69808ef53a
12 ce23313490656aaeda9b21aa137a7e70fb268db9372cafeefe860e3fb98c4dfb
13 d34eedc74d714c7a5702b386d36ee422d614d0239cf45e3ae417dd5cd6a09f6f
14 55330726f9221f239c76d4809463ebc251a634360f7098cff98931f8948b7669
15 e84f66e180f1be0c5ef057ea2c4bc74106791b6b794e2de74dc56a9968fa8410
16 e2e4283c81ace8474a32dfc6e43fa3515f02e9bdc93daa86d84875cf9d4ac72d
17 aa1588b1ca21bf13dc132fd12e485cf0edebc787ee53a4cf6df6aa8d5e5e5611
18 9f6723f0419bc16b0a1230407ab3e25015dda27793c424bc50a6ace4f7de4a2e
```

chicken and egg

We'd like to store the update info somewhere but

- we don't have any db tools in the base system
- we need to generate and grab it securely from the cdn

no database

- But we have `locate` in the base system.
- it's been designed to store efficiently "similar" strings (by sorting and compressing according to prefix)
- already used for `pkglocatedb`
- this stores each path in packages prefixed by the pkgname/path location

pkglocatedb

```
nausicaa$ pkglocate /usr/local/bin/vim
graphviz-2.42.3p0:math/graphviz,-main:/usr/local/bin/vimdot
vim-8.2.5036-gtk3-lua:editors/vim,-main,gtk3,lua:/usr/local/bin/vim
vim-8.2.5036-gtk3-lua:editors/vim,-main,gtk3,lua:/usr/local/bin/vimdiff
vim-8.2.5036-gtk3-lua:editors/vim,-main,gtk3,lua:/usr/local/bin/vimtutor
vim-8.2.5036-gtk3-perl-python3-ruby:editors/vim,-main,gtk3,perl,python3,ruby:/usr/local/bin
vim-8.2.5036-gtk3-perl-python3-ruby:editors/vim,-main,gtk3,perl,python3,ruby:/usr/local/bin
vim-8.2.5036-gtk3-perl-python3-ruby:editors/vim,-main,gtk3,perl,python3,ruby:/usr/local/bin
vim-8.2.5036-gtk3-python3:editors/vim,-main,gtk3,python3:/usr/local/bin/vim
[...]
```

pkglocatedb 2

- It is very efficient: 300MB compress to 23MB
- It is fast
- It is in the base system

locate vs updateinfo

- generate data with `pkgname:update-info-line`
- this should compress correctly
- where to put it to make this accessible
- I did a script that worked. Compression is okay (compresses 23M to 3M)

chicken and egg too

- I gave the script to my fellow builders and asked for pkgindex.tgz to be on the mirrors
- they did it for a while, but I got distracted
- and then they no longer did it
- right when I got motivated again

so better integration

- it had to be on, all the time. Add glue at the end of dpb to generate it ?
- delivery system. Sign it specifically ? teach `pkg_add` how to read it ?
- scrape that, let's use quirks

quirks ?

- Quirks is the package that holds "Exceptions" to the rules (such as package renames, or packages that got dropped).
- First action of `pkg_add` ever is always to try to update quirks.
- So it's a natural location to drop update info

first experiment

- so I got the script that builds the db into quirks
- told my friends to always regenerate quirks at the end
- and waited for the new package to show up

Timings: whazzza

- (there was a small issue with "always-update" packages, let's avoid them)
- try to grab the updateinfo from the locate before going to the packages
- result **over twenty times speed-up**
- so worth making it work

details

- the db is linked to a given quirks, which means a given package repository.
- this is not a big issue because we got unique objects for repositories
- furthermore, quirks is an "always-update" package, so if we find we don't need to update it, it means the quirks we got contains update info for our packages
- we can actually put that in production !

more speed-ups I

- we run a separate locate for each updateinfo
- we can actually run a single locate upfront, because we got the list of pkgnames we want to handle

```
1  sub prime_update_info_cache($self, $state, $setlist)
2  {
3          my $progress = $state->progress;
4          my $found = {};
5
6          for my $set (@{$setlist}) {
7                  for my $h ($set->older, $set->hints) {
8                          next if $h->{update_found};
9                          my $name = $h->pkgname;
10                         my $stem = OpenBSD::PackageName::splitstem($name);
11                         next if $stem =~ m/^\.libs\d*\-/;
12                         next if $stem =~ m/^partial\-/;
13                         $stem =~ s/\%.*//; # zap branch info
14                         $stem =~ s/\-\-.*//; # and set flavors
```

more speed-ups II

```
15                      $self->add_stem($stem);
16              }
17      }
18      my @list = sort keys %{$self->{stems}};
19      return if @list == 0;
20
21      my $total = scalar @list;
22      $progress->set_header(
23          $state->f("Reading update info for installed packages",
24              $total));
25      my $done = 0;
26      my $oldname = "";
27
28      open my $fh, "-|", $self->pipe_locate(map { "$_-[0-9]*"} @list)
29          or $state->fatal("Can't run locate: #1", $!);
30      while (<$fh>) {
31              if (m/^(.*?)\:(.*)/) {
32                      my ($pkgname, $value) = ($1, $2);
```

more speed-ups III

```
33                          $found->{OpenBSD::PackageName::splitstem($pkgname)} = 1;
34                          $self->{raw_data}{$pkgname} //= '';
35                          $self->{raw_data}{$pkgname} .= "$value\n";
36                          if ($pkgname ne $oldname) {
37                                  $oldname = $pkgname;
38                                  $done++;
39                          }
40                          $progress->show($done, $total);
41                  }
42          }
43          close($fh);
44          return unless $state->defines("CACHING_VERBOSE");
45          for my $k (@list) {
46                  if (!defined $found->{$k}) {
47                          $state->say("No cache entry for #1", $k);
48                  }
49          }
50  }
```

always-update

- at first those were not handled at all
- because we handled the full packing-list, **which is ordered**
- it means a package that needs an update each time it changes
- after a few tries, I decided that storing a crypto hash would work
- so now it is `@option always-update <hash value>`
- and `pkg_create` generates it

Unforeseen consequences: new bottlenecks

- LRU bugs
- not moving files if possible (extract/delete/install pattern)
- User interface (processing large packing lists)

the generation of quirks: McGyver

Rougly ten lines in dpb:

```
if ($state->{all}) {
        my $core = DPB::Core->get;
        my $w = DPB::PkgPath->new('devel/quirks');
        if ($state->{engine}{built_packages}) {
                $state->grabber->clean_packages($core, $w->fullpkgpath);
        }
        my $subdirlist = {};
        $w->add_to_subdirlist($subdirlist);
        $state->grabber->grab_subdirs($core, $subdirlist, undef);
        $state->engine->check_buildable;
        $core->mark_ready;
        main_loop();
}
```

the generation of quirks: v2

```
1  my @later = $state->grabber->later;
2  if (@later != 0) {
3          my $core = DPB::Core->get;
4          my $subdirlist = {};
5          for my $w (@later) {
6                  if ($state->{engine}{built_packages}) {
7                          $state->grabber->clean_packages($core,
8                              $w->fullpkgpath);
9                  }
10                 $w->add_to_subdirlist($subdirlist);
11         }
12         $state->grabber->grab_subdirs($core, $subdirlist, undef);
13         $state->engine->check_buildable;
14         $core->mark_ready;
15         main_loop();
16 }
```

Making things better

- Agressively review old code and kill it
- Better comments about stuff that's now stabilized
- Document "best practices"

perl v5.36

From some time now, perl has got "experimental features", which is a good and a bad thing

good

- Perl is still evolving
- it gets all the good stuff from perl6 over the years
- For instance yada yada

bad

- but they are experimental
- so you can't use them in production
- see switch operator

"Yada yada"

- Acknowledges "fill in the blanks Rapid Application Developement"
- "Yada Yada" is just ...

```
1 if ($verbose && ...) {
2         do_something();
3 }
4
5 for my $i (@list) {
6         if ($i =~ m/shouldn't happen/) {
7                 ...;
8         }
9 }
```

That's actually very useful

given/when

There was some kind of "pattern matching" related to keywords `given` and `when` that was available in perl, very similar to what pattern matching looks like in ocaml.

But for various reasons, the experiment didn't pan out! so anyone who's been using these is up shit creek!

semantic versioning

For a long time, `use` has been used (lol) to indicate semantic variations on perl (like
`use strict;`,
`use warnings;`
or more complicated forms, e.g.,
`use feature qw(say);`
The most interesting variation is
`use v5.something;`
which does enable various thingies.

fight or flight

For production code, it makes sense to use a recent version and its improvements...
as long as they're not *experimental* !!!

Prototypes

- It's totally different from other languages.
- Prototypes are used to "create syntax" or rather reproduce built-in behaviors.
- Completely esoteric syntax

Example I

```
1  sub mypush (\@@)
2  {
3  }
4  ...
5  mypush @l, 1, 2, 3;
6
```

Example II

```
1  sub try(&@)
2  {
3          my ($try, $catch) = @_;
4          eval { &$try };
5          if ($@) {
6                  &$catch;
7          }
8  }
9
10 sub catch(&)
11 {
12         return $_[0];
13 }
14
15 try {
16         ...
17 } catch {
18         ...
19 }
```

Signatures

- so what's called "prototypes" in other languages is called "signatures" in perl !
- it's ambiguous wrt prototypes
- so accordingly prototypes require annotation e.g.,

```
1  sub try :prototype(&@)($try, $catch)
2  {
3          eval { &$try() };
4          if ($@) {
5                  &$catch;
6          }
7          ...
8  }
9
10 sub catch :prototype(&)($code)
11 {
12         return $code;
13 }
14
```

Converting to 5.36: other stuff

- Every prototype needs to be made unambiguous so, `sub foo(...)` → `sub foo :prototype(...)`.
- Code calls may need explicit parentheses: stuff like

```
1 &$code;
```

  does call code *in the same context* as the parent with the same parameters. Use

```
1 &$code();
```

  instead.
- Object calls through indirect syntax has been deprecated.

Sometimes useful

```
1  package OpenBSD::PackingElement::Cwd;
2  sub find_extractible    # forwarder
3  {
4          &OpenBSD::PackingElement::Meta::find_extractible;
5  }
```

Indirect syntax ?

- Again linked to built-ins.
- Printing to a file looks like

```
print $fh "result is ", $i, "\n";
```

- as opposed to the less fancy

```
$fh->print("result is ", $i, "\n");
```

Converting to 5.36: signatures

Modelled after C++

- named scalar parameters like `$x`
- unnamed parameters if unused
- default values for parameters with `$x = value`

But perl!

Can slurp renaming parameters with either

```
1 sub f($x, $y, @l)
```

or

```
1 sub f($x, $y, %h)
```

What about @_

```
1  Implicit use of @_ in subroutine entry with signatured subroutine
2      is experimental at a line 5.
```

Why use signatures I

It makes perl code looks almost normal! before

```
1   sub set
2   {
3           my ($self, $set) = @_;
4           delete $self->{object};
5           $self->{set} = $set;
6           return $self;
7   }
8
9   sub object
10  {
11          my ($self, $object) = @_;
12          delete $self->{set};
13          $self->{object} = $object;
14          return $self;
15  }
16
17  sub what
18  {
19          my ($self, $what) = @_;
20          $self->{what} = $what;
21          return $self;
22  }
23
24  sub new
25  {
26          my $class = shift;
27
```

Why use signatures II

```
28          bless {}, $class;
29  }
30
```

after

```
1  sub set($self, $set)
2  {
3          delete $self->{object};
4          $self->{set} = $set;
5          return $self;
6  }
7
8  sub object($self, $object)
9  {
10         delete $self->{set};
11         $self->{object} = $object;
12         return $self;
13 }
14
15 sub what($self, $what = undef)
16 {
17         $self->{what} = $what;
18         return $self;
19 }
20
21 sub new($class)
22 {
23         bless {}, $class;
24 }
25
```

What about documentation

- because of OO
- sometimes a base method does nothing

```
1 package OpenBSD::PackingElement;
2  # $self->find_dependencies($state, $l, $checker, $pkgname)
3  sub find_dependencies($, $, $, $, $)
4  {
5  }
6
```

Problems

- Errors happen at runtime, so difficult to catch them all

```
Too few arguments for subroutine 'main::f'
    (got 2; expected 3) at b line 11.
```

- Much harder on lambdas

```
Too few arguments for subroutine 'main::__ANON__'
    (got 2; expected 3) at b line 11.

```

(it would be great to annotate the name of the anonymous routine with line number and filename)

Paradigm shift

- perl has long been fuzzy on parameter numbers
- some default interfaces use this to "tack on" parameters (e.g. signals)
- some OO code has been designed to take advantage of this

Signals

- like in C, normal signal handlers get the signal number (but it's not guaranteed)
- There are extra signal handlers for `__DIE__` or `__WARN__` and these take optional messages.

Easy solutions

- in the worst case, you can slurp stuff with @
- but it's better to track the error and get the right number of parameters
- sometimes variations, because of time (a parameter is no longer used)
- or variation in parameters used for a constructor subclass

real example

```
1  package OpenBSD::ProgressMeter;
2  sub new($class, $state)
3  {
4          # now saves state
5  }
6
7  sub for_list($self, $msg, $l, $code) # + $state
```

We can use @ to get parameters through until we have a default value

real example

```
1
2  package OpenBSD::PackingList;
3
4  sub read($a, $u, $code = \&defaultCode)
5
6  sub fromfile($a, $fname, $code = \&defaultCode)
7
8  package OpenBSD::PackageLocation;
9  sub grabPlist($self, $code = \&OpenBSD::PackingList::defaultCode)
10
11 sub grabPlist($self, @code)
```

Grepping

```
1  package OpenBSD::PackingElement::Cwd;
2  sub find_extractible    # forwarder
3  {
4          &OpenBSD::PackingElement::Meta::find_extractible;
5  }
1
2          my $handler = sub {         # SIGHANDLER
3                  $state->{received} = shift;
4                  $state->errsay("Interrupted");
5                  if ($state->{hardkill}) {
6                          delete $state->{hardkill};
7                          return;
8                  }
9                  $state->{interrupted}++;
10         };
11
12         local $SIG{'INT'} = $handler;
13         local $SIG{'QUIT'} = $handler;
14         local $SIG{'HUP'} = $handler;
15         local $SIG{'KILL'} = $handler;
```

coda

- insufficient coverage of the code base
- should look at `Devel::Cover` and subclasses
- static validation is lagging (`perlcritic`).
- future optimizations.