Packaging for programming language implementors
Why you do want to care and what to do about it

Alexey Saushev

September 23, 2015

Why it matters

Why I don't use Gambit Scheme (don't even consider trying it):

Why it matters

Why I don't use Gambit Scheme (don't even consider trying it):

- gsc is installed by Gambit Scheme and by GhostScript.
- GhostScript is required by TeX and CUPS.
- There's no alternative neither to TeX nor to CUPS.
- I use GhostScript to read old papers that come in PostScript.
- Multiple Scheme implementations don't conflict with other software: CHICKEN, Racket, Guile...

Why it matters

Importance of cooperation:

Why it matters

Importance of cooperation:

- Gambit developers were told about this problem.
- ...Numerous times already.
- I have to maintain corrections myself.
- ...But I'm not going to.

Why it matters

Importance of cooperation:

- Gambit developers were told about this problem.
- ...Numerous times already.
- I have to maintain corrections myself.
- ...But I'm not going to.

I can use another Scheme implementation.

And I do!

Why it matters

Save others' time!

Why it matters

Complexity of the outer world:

- Less than 300 packages is simple system.
  E.g. pfSense contains around 100 packages.
  (Excluding base system, which is considered monolithic, custom software in web interface and few other places.)
- 10-20 systems with different software is typical.
  (Sometimes more!)

Why it matters

Complexity of the outer world:

- Rebuild-deploy approach is hard.
  Long interruptions, unreliable tools.
- Rebuild-migrate approach is even harder.
- Cloud computing and containers do not work for everyone.
  (Data access may be limiting.)
- Persistency approach makes everything unusual and hard.
  - It fills your storage up.
  - It complicates problem diagnosis very much.
  - It casts doubts at security.

  (Nix and Guix are nice only for enthusiasts.)

Why it matters

Package management is required!

Why it matters

Package management system (PMS) is used to:

- build binary packages of software.
  - Building in controlled environment.
  - Performing consistency checks.
  - Running bundled tests (where applies).
- deploy software, create new installations.
- update existing installations.

Ensuring consistency and reproducibility!
(Important!)

Why it matters

Using package management system helps
even tiny projects of pfSense scale.

The problem

The problem

Programming language implementors...

The problem

Programming language implementors...

- ...are well aware of these problems.

The problem

Programming language implementors...

- ...are well aware of these problems.
- ...do solve these problems.

The problem

Programming language implementors...

- ...are well aware of these problems.
- ...do solve these problems.
- ...only do that for their part of the whole environment.

The problem

Programming language implementors...

- ...are well aware of these problems.
- ...do solve these problems.
- ...only do that for their part of the whole environment.
- ...solve them incompletely even in their smaller world.

The problem

Numerous libraries in “conventional” languages
managed by “conventional” tools.

The problem

Programming language implementors suggest another set.

The problem

Two different package management systems in parallel.

The problem

???

The problem

???

The problem

Not a good idea, really!

The problem

"Sword and shield"

- Programming languages implementors misunderstand what engineers actually need.
- Packagers try hard to correct implementors.

The problem

"Sword and shield"

- Programming languages implementors misunderstand what engineers actually need.
- Packagers try hard to correct implementors.
- System engineers try hard to stay sane while switching back and forth between implementors and packagers.

The problem

Typical understanding:

> *I think we can address most of the problems with have using tarballs, a good separation between fetch, compile, install and test phases. We could have that with .setup scripts.*

The problem

Typical understanding:

> *I think we can address most of the problems with have using tarballs, a good separation between fetch, compile, install and test phases. We could have that with .setup scripts.*

This is exact quote picked on #chicken.

The problem

Separation of stages is enough:

- Fetch.
- Build.
- Test.
- Install.

Done.

The problem

Separation of stages is enough:

- Fetch.
- Build.
- Test.
- Install.

Done.

NOT AT ALL!

Fetch stage

Distribution bundle.

- Caching.
- Redistribution.
- Backup.
- Integrity check.

Fetch stage

Distribution bundle.

- Distinct names.
- Semantic version numbers.
  Because we need to handle compatible and incompatible changes.
- Stability.
  - Reproducibility.
  - Security.

  Never, never ever replace published bundle!
  Someone has fetched it already.
  Even if you did everything within 15 min!
  (Remember time zones!)

Fetch stage

Distribution bundle.

- (Not obvious.) Raw access to files.
  - Detecting new releases.
  - Mirroring.
  - Resuming fetches.
- (Not obvious.) Distribution bundle URL.
  Packager needs a way to construct URL of your bundle.
  Don't make him to implement custom fetch stage!
  (Even if he can do that easily!)
  Ideally, URL should look like:

  http://site/path/package-0.0.tar.gz

(These two are actually independent.)

Fetch stage

This is not news.

The problem is ensuring it.

Dependencies

Dependencies
(You forgot about them, but I didn't.)
The most important part, usually done wrong.

Dependencies

Dependencies
(You forgot about them, but I didn't.)
The most important part, usually done wrong.

- Don't recurse!
  At least let packager handle them his way.
  (You don't build binary packages anyway.)
- Check prerequisites and report failures.
  Report failures in machine processable and human ways.
- Provide a way to list dependencies in some human way so that packager can identify them.
- Avoid bundling common software or installing it yourself.
  Packagers and system engineers get annoyed when you install your personal copy of libjpeg.

Dependencies

```
$ chicken-install -show-depends http-client
...
        (synch 2.1.0)
md5:
        (message-digest 3.1.0)
intarweb:
        defstruct
        (uri-common 0.2)
        (base64 3.0)
http-client:
        (intarweb 1.5)
        (uri-common 0.7)
        (message-digest 3.0.0)
        (md5 3.0.0)
        string-utils
        (sendfile 1.7.4)
```

Dependencies

Listing dependencies:

- Non-recursive mode needed.
- Does "(P 1.0.0)" mean P $\geq$ 1.0.0? P $=$ 1.0.0? Anything else?

Build stage

Usually more or less correct, yet...

- Don't search everywhere you can reach.
  Perhaps packager's intentions differ.
  /usr/local is not necessarily what you're allowed to use.
- Let packager override.
  At least prepare to discuss such requests.
- Don't capture build time information.
  Build time cc is

  /tmp/lang/chicken/work/.wrapper/bin/cc

  which doesn't exist after installation.
  Not clear how to deal with it. (Case by case?)

Installation

- Staged installation is a hard requirement these days.
  If you don't do that, expect strong negative feedback.
- Registration of package without installation.
  Your software is going to be installed by unpacking a binary package.
- Deregistration of package.
  Your package is going to be removed by external tool.
- Configuration files to be handled specially.
- Installation time commands (e.g. building indices) to be handled specially.

Miscellanea

Other cooperation issues.

- Consider PMS interoperability as build time option. Installing or uninstalling your packages makes PMS information inconsistent.
- Protect operator from messing installation up.

Save packager's time and your software will be promoted by people you have never heard of!

Questions?