Reproducible Software Deployment with GNU Guix

Ludovic Courtès

Inria Rennes Bretagne Atlantique, November 2015
The difficulty of keeping software environments under control.
#1. Upgrades are hard.
Distribution Upgrade of all the files:

**WARNING**

Following the upgrade instructions found in the [release notes](https://wiki.debian.org/DebianPackageManagement) is the best way to ensure that your system upgrades from one major Debian release to another (e.g. from lenny to squeeze) without breakage!

These instructions will tell you to do a `dist-upgrade` (instead of `upgrade`) in the case of `apt-get` or `full-upgrade` (instead of `safe-upgrade` in the case of `aptitude`) at least once. So you would have to type something like

```sh
# aptitude full-upgrade
```

or

```sh
# apt-get dist-upgrade -dy
```
4.1. Preparing for the upgrade
   4.1.1. Back up any data or configuration information
   4.1.2. Inform users in advance
   4.1.3. Prepare for downtime on services
   4.1.4. Prepare for recovery
   4.1.5. Prepare a safe environment for the upgrade

4.2. Checking system status
   4.2.1. Review actions pending in package manager
   4.2.2. Disabling APT pinning
   4.2.3. Checking packages status
   4.2.4. The proposed-updates section
   4.2.5. Unofficial sources

4.3. Preparing sources for APT
   4.3.1. Adding APT Internet sources
   4.3.2. Adding APT sources for a local mirror
   4.3.3. Adding APT sources from optical media

4.4. Upgrading packages
   4.4.1. Recording the session
   4.4.2. Updating the package list
   4.4.3. Make sure you have sufficient space for the upgrade
   4.4.4. Minimal system upgrade
   4.4.5. Upgrading the system

4.5. Possible issues during upgrade
   4.5.1. Dist-upgrade fails with “Could not perform immediate configuration”
   4.5.2. Expected removals
   4.5.3. Conflicts or Pre-Depends loops
   4.5.4. File conflicts
   4.5.5. Configuration changes
   4.5.6. Change of session to console
#2. Stateful system management is intractable.
$DISTRO$

\[\downarrow \text{apt-get update} \]

state 1\(_a\)

$DISTRO$

\[\downarrow \text{apt-get update} \]

state 1\(_b\)
```bash
$DISTRO

apt-get update

state 1

apt-get install foo

state 2

$DISTRO

apt-get update

state 1

apt-get remove bar

state 2
```
$DISTRO$

apt-get update

state 1

apt-get install foo

state 2

apt-get remove bar

state 3

apt-get update

state 1b

apt-get remove bar

state 2b

apt-get install foo

state 3b
apt-get update
state 1a
apt-get install foo
state 2a
apt-get remove bar
state 3a

= ?

apt-get update
state 1b
apt-get remove bar
state 2b
apt-get install foo
state 3b
#3. It’s worse than this.
Application-level package managers

- **Anaconda** - a package manager for Python
- **Assembly** - a partially compiled code library for use in Common Language Infrastructure (CLI) deployment, versioning and security.
- **Biicode** - a file-focused dependency manager for C/C++ languages and platforms (PC, Raspberry Pi, Arduino).
- **Bower** - a package manager for the web.
- **UPT** - a fork of Bower that aims to be a universal package manager, for multiple environments and unlimited kind of package
- **Cabal** - a programming library and package manager for Haskell
- **Cargo** - a package manager for Rust (programming language)
- **CocoaPods** - Dependency Manager for Objective-C and RubyMotion projects
- **Composer** - Dependency Manager for PHP
- **CPAN** - a programming library and package manager for Perl
- **CRAN** - a programming library and package manager for R
- **CTAN** - a package manager for TeX
- **DUB** - a package manager for D
It’s worse, really.
“Let’s Package jQuery: A Javascript Packaging Dystopian Novella” by Chris Webber

http://dustycloud.org/blog/javascript-packaging-dystopia/
Giving up?
Giving up?

→ “app bundles” (Docker images)
Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities

Docker Hub is a central repository for Docker developers to pull and push container images. We performed a detailed study on Docker Hub images to understand how vulnerable they are to security threats. Surprisingly, we found that more than 30% of images in official repositories are highly susceptible to a variety of security attacks (e.g., Shellshock, Heartbleed, Poodle, etc.). For general images – images pushed by docker users, but not explicitly verified by any authority – this number jumps up to ~40% with a sampling error bound of 3%.
Functional package management.
openmpi = f(hwloc, gcc, make, coreutils)

where $f = ./configure && make && make install$
openmpi = \( f(hwloc, gcc, make, coreutils) \)

hwloc = \( g(pciaccess, gcc, make, coreutils) \)
openmpi = \( f(\text{hwloc}, \text{gcc}, \text{make}, \text{coreutils}) \)

\( \text{hwloc} = g(\text{pciaccess}, \text{gcc}, \text{make}, \text{coreutils}) \)

\( \text{gcc} = h(\text{make}, \text{coreutils}, \text{gcc}_0) \)

...
openmpi = f(hwloc, gcc, make, coreutils)
hwloc = g(pciaccess, gcc, make, coreutils)
gcc = h(make, coreutils, gcc)
... the complete DAG is captured
▶ **A Safe and Policy-Free System for Software Deployment**, Dolstra et al., 2003

**Nix**, http://nixos.org/nix/

▶ **Functional Package Management with Guix**, Courtès, 2013
(define hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
      (method url-fetch)
      (uri (string-append
            "mirror://gnu/.../hello-" version
            ".tar.gz"))
      (sha256 (base32 "0wqd...dz6"))))
  (build-system gnu-build-system)
  (synopsis "Hello, world!")
  (description "Produce a friendly greeting.")
  (home-page "http://www.gnu.org/software/hello/")
  (license gpl3+)))
build processes
chroot, separate UIDs

Guile Scheme
(guix packages)
(guix store)

build daemon
build processes
chroot, separate UIDs

Guile Scheme
(guix packages)
(guix store)

build daemon

RPCs
build processes
chroot, separate UIDs

Guile, make, etc.

Guile, make, etc.

Guile, make, etc.

Guile Scheme

(guix packages)

(guix store)

build daemon

RPCs
$ guix build hello

**isolated build**: chroot, separate name spaces, etc.
$ guix build hello
/gnu/store/ h2g4sf72... -hello-2.10

hash of all the dependencies
$ guix build hello
/gnu/store/ h2g4sf72... -hello-2.10

$ guix gc --references /gnu/store/...-hello-2.10
/gnu/store/...-glibc-2.22
/gnu/store/...-gcc-4.9.3-lib
/gnu/store/...-hello-2.10
$ guix build hello
.gnu/store/ h2g4sf72... -hello-2.10

$ guix gc --references /gnu/store/...-hello-2.10
 /gnu/store/...-glibc-2.22
 /gnu/store/...-gcc-4.9.3-lib
 /gnu/store/...-hello-2.10

(nearly) bit-identical for everyone
$ guix package -i gcc-toolchain coreutils sed grep
...

demo

$ eval `guix package --search-paths`
...

$ guix package --manifest=my-software.scm
...
Want your PhD student to hack on GNUnet?
Want your PhD student to hack on GNUnet?

A simple matter of installing the deps, right?
$ guix environment --container gnunet
...

$ guix environment --ad-hoc python-ipython python-numpy \ -E ipython
...

<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
<th>Name</th>
<th>Last Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☁️ ⚡️</td>
<td>default-config</td>
<td>6 mo 8 days - #204</td>
</tr>
<tr>
<td></td>
<td>☁️ ⚡️</td>
<td>guix-environment</td>
<td>2 days 3 hr - #67</td>
</tr>
<tr>
<td></td>
<td>☁️ ⚡️</td>
<td>guix-environment-clang</td>
<td>26 days - #21</td>
</tr>
<tr>
<td></td>
<td>☁️ ⚡️</td>
<td>guix-environment-gcc-5</td>
<td>26 days - #15</td>
</tr>
<tr>
<td></td>
<td>☁️ ⚡️</td>
<td>guix-environment-minimal</td>
<td>5 days 21 hr - #13</td>
</tr>
</tbody>
</table>
Whole-system deployment.
Linux-libre

initial RAM disk
Linux-libre

initial RAM disk

PID 1: GNU dmd
services...
Linux-libre

initial RAM disk

PID 1: GNU dmd

services...
Linux-libre

initial RAM disk

PID 1: GNU dmd
services...

applications

Guile

Guile
Trustworthiness.
Debian’s dirtiest secret: Binary packages built by developers are used in the archive

— Lucas Nussbaum, FOSDEM 2015
Transparent binary/source deployment

alice@foo$ guix package --install=emacs
The following package will be installed:
  emacs-24.5 /gnu/store/...-emacs-24.5

The following files will be downloaded:
  /gnu/store/...-emacs-24.5
  /gnu/store/...-libxpm-3.5.10
  /gnu/store/...-libxext-1.3.1
  /gnu/store/...-libxaw-1.0.11
Transparent binary/source deployment

```
alice@foo$ guix package --install=emacs
The following package will be installed:
  emacs-24.5 /gnu/store/...-emacs-24.5

The following files will be downloaded:
  /gnu/store/...-libxext-1.3.1
  /gnu/store/...-libxaw-1.0.11
The following derivations will be built:
  /gnu/store/...-emacs-24.5.drv
  /gnu/store/...-libxpm-3.5.10.drv
```
(define foo (package ...))
(define foo (package ...))

test

guix build foo
/gnu/store/...-foo-1.0
(define foo (package ...))

test

user

guix build foo
/gnu/store/...-foo-1.0

git push

git.sv.gnu.org
(define foo (package ...))

test

guix build foo /gnu/store/...-foo-1.0

git push

user

hydra.gnu.org

build farm

pull

pull

git.sv.gnu.org
(define foo (package ...))

test

user

get binary

hydra.gnu.org

build farm

pull

git.push

git.sv.gnu.org
(define foo (package ...))

guix build foo /gnu/store/...-foo-1.0

git push

git.push

user

git.sv.gnu.org

pull
(define foo (package ...))

guix build foo
/gnu/store/...-foo-1.0

git push

user

no “maintainer uploads”

no single point of trust

git.sv.gnu.org
(define emacs (package ...)) /gnu/store/...-emacs-24.5
The path to greater user control

1. Bit-reproducible builds

2. No single binary provider

3. Tools for users to challenge binaries
The path to greater user control

1. Bit-reproducible builds
   • we have *isolated build environments*!
   • ... but we need builds to be *deterministic*
   • http://reproducible-builds.org

2. No single binary provider

3. Tools for users to challenge binaries
The path to greater user control

1. **Bit-reproducible builds**
   - we have *isolated build environments*!
   - ... but we need builds to be *deterministic*
   - [http://reproducible-builds.org](http://reproducible-builds.org)

2. **No single binary provider**
   - `guix publish`
   - publish over GNUNet? (GSoC 2015)

3. **Tools for users to challenge binaries**
The path to greater user control

1. **Bit-reproducible builds**
   - we have *isolated build environments*!
   - ... but we need builds to be *deterministic*
   - [http://reproducible-builds.org](http://reproducible-builds.org)

2. **No single binary provider**
   - *guix publish*
   - publish over GNUnet? (GSoC 2015)

3. **Tools for users to challenge binaries**

/gnu/store/...-openssl-1.0.2d contents differ:
  local hash: 0725122...
  http://hydra.gnu.org/...-openssl-1.0.2d: 0725122...
  http://guix.example.org/...-openssl-1.0.2d: 1zy4fma...

/gnu/store/...-git-2.5.0 contents differ:
  local hash: 00p3bmr...
  http://hydra.gnu.org/...-git-2.5.0: 069nb85...
  http://guix.example.org/...-git-2.5.0: 0mdqa9w...

/gnu/store/...-pius-2.1.1 contents differ:
  local hash: 0k4v3m9...
  http://hydra.gnu.org/...-pius-2.1.1: 0k4v3m9...
  http://guix.example.org/...-pius-2.1.1: 1cy25x1...
Status.
Timeline

- Nov. 2012 — dubbed GNU
- Jan. 2013 — 0.1
- Apr. 2014 — 0.6, signed binaries, guix system
- July 2014 — 0.7, installable operating system
- 29 Jan. 2015 — 0.8.1, ARMv7 port
- Aug. 2015 — Reproducibility in Parallel Computing Workshop (RepPar)
- 5 Nov. 2015 — 0.9.0, new service framework, etc.
Status

- full-featured package manager
- 2,600+ packages, 4 platforms
- Guix System Distribution\(^\beta\)
- binaries at http://hydra.gnu.org
- tooling: auto-update, “linting”, etc.
- l10n: 8 languages!
25 contributors each month
... and lots of friendly people!
≈400 commits per month
≈200–500 new packages per release
your help needed!

- install the distribution
- use it, report bugs, add packages
- help with the **infrastructure** + admin
- **donate** hardware/money
- share your **ideas**!