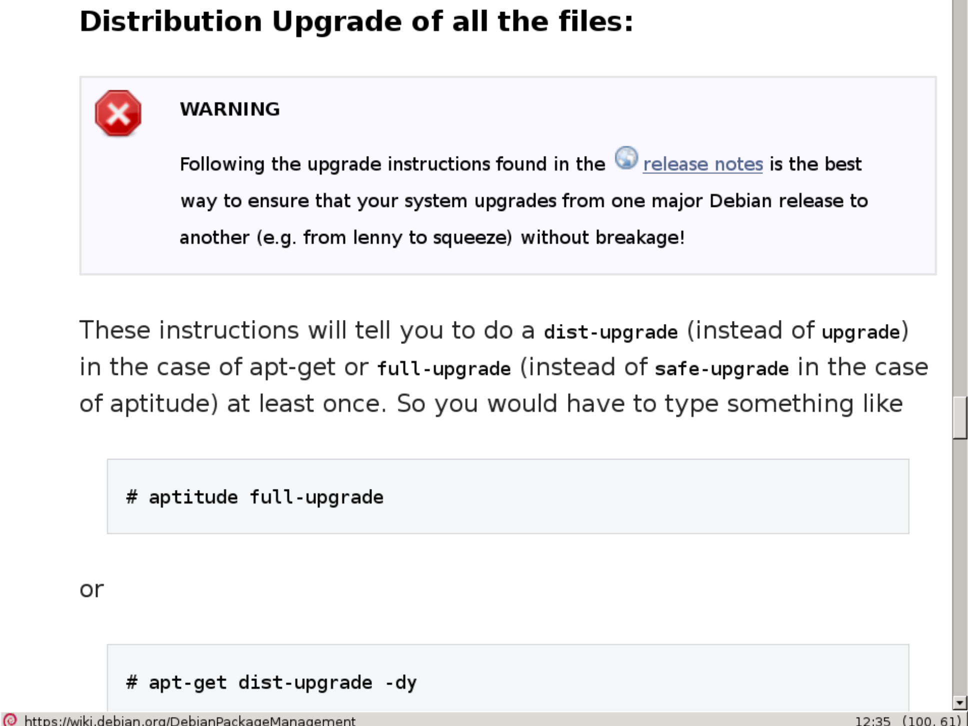
Reproducible Software Deployment with GNU Guix

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 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!

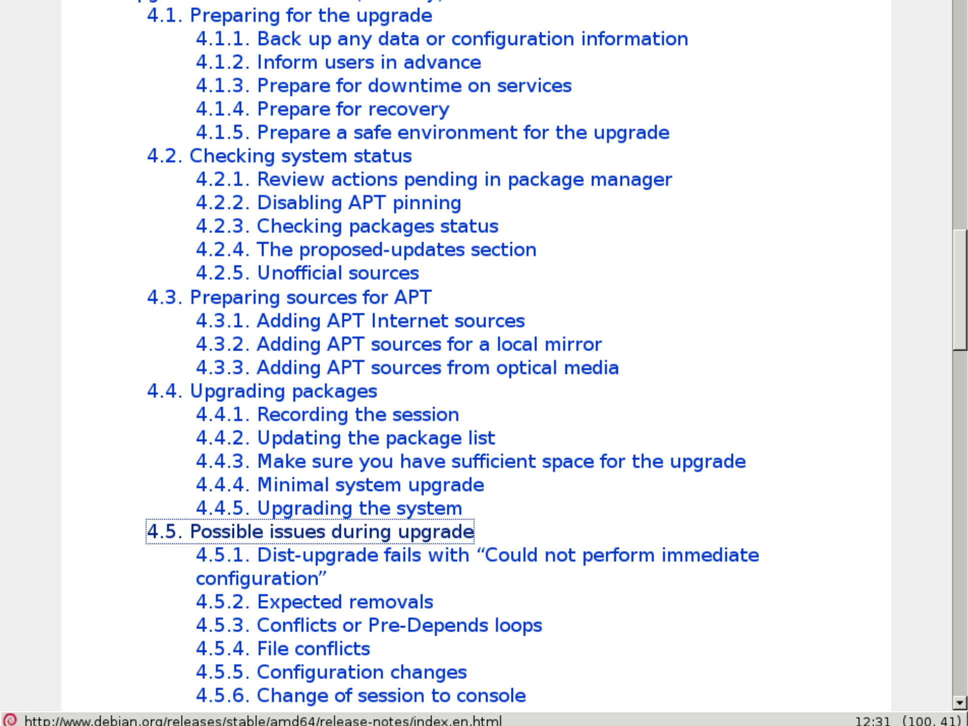
These instructions will tell you to do a **`dist-upgrade`** (instead of **`upgrade`**) in the case of apt-get or **`full-upgrade`** (instead of **`safe-upgrade`** in the case of aptitude) at least once. So you would have to type something like

```
# aptitude full-upgrade
```

or

```
# apt-get dist-upgrade -dy
```

https://wiki.debian.org/DebianPackageManagement 12:35 (100, 61)



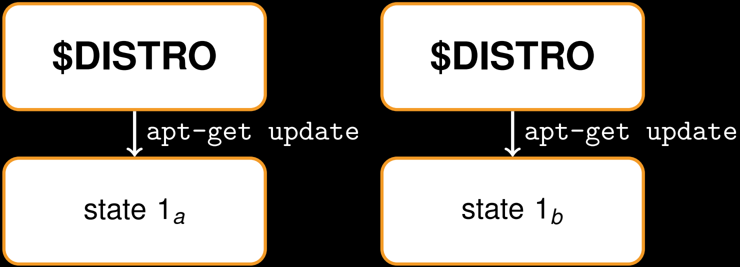
- 4.1. Preparing for the upgrade
  - 4.1.1. Back up any data or configuration information
  - 4.1.2. Inform users in advance
  - 4.1.3. Prepare for downtime on services
  - 4.1.4. Prepare for recovery
  - 4.1.5. Prepare a safe environment for the upgrade
- 4.2. Checking system status
  - 4.2.1. Review 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

http://www.debian.org/releases/stable/amd64/release-notes/index.en.html 12:31 (100, 41)

#2. Stateful system management is intractable.

$DISTRO

$DISTRO



$DISTRO

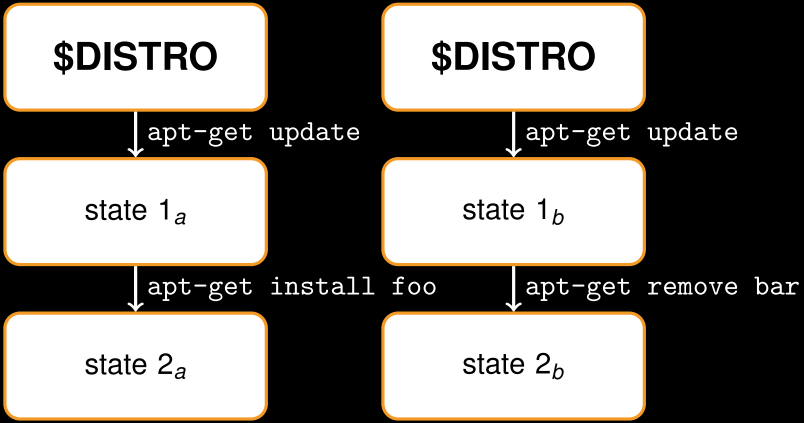
`apt-get update`

state 1_a

$DISTRO

`apt-get update`

state 1_b



$DISTRO

`apt-get update`

state 1_a

`apt-get install foo`

state 2_a

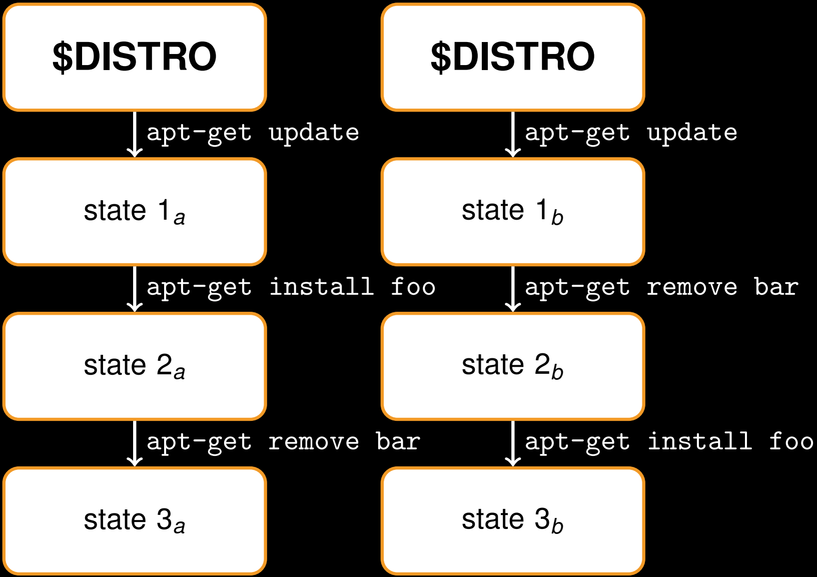
$DISTRO

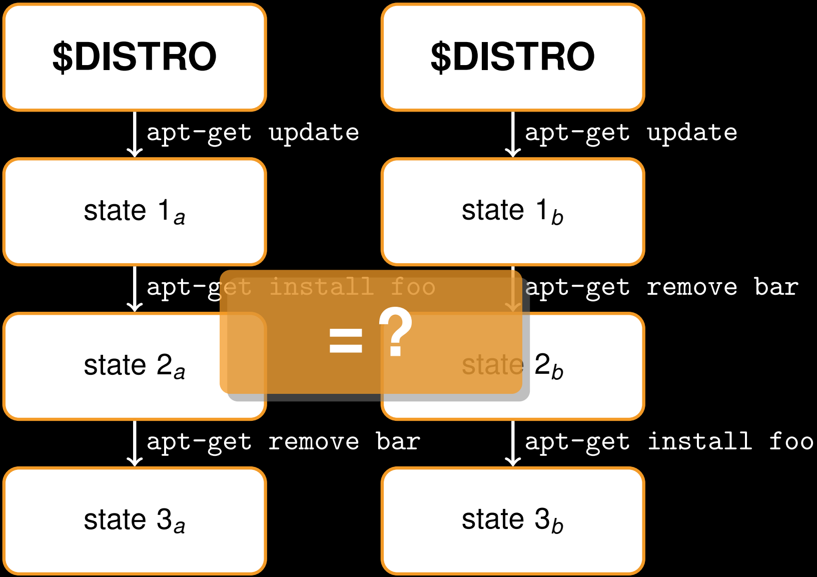
`apt-get update`

state 1_b

`apt-get remove bar`

state 2_b


$DISTRO
apt-get update
state 1_a
apt-get install foo
state 2_a
apt-get remove bar
state 3_a
$DISTRO
apt-get update
state 1_b
apt-get remove bar
state 2_b
apt-get install foo
state 3_b



$DISTRO

`apt-get update`

state 1_a

`apt-get install foo`

state 2_a

`apt-get remove bar`

state 3_a

$DISTRO

`apt-get update`

state 1_b

`apt-get remove bar`

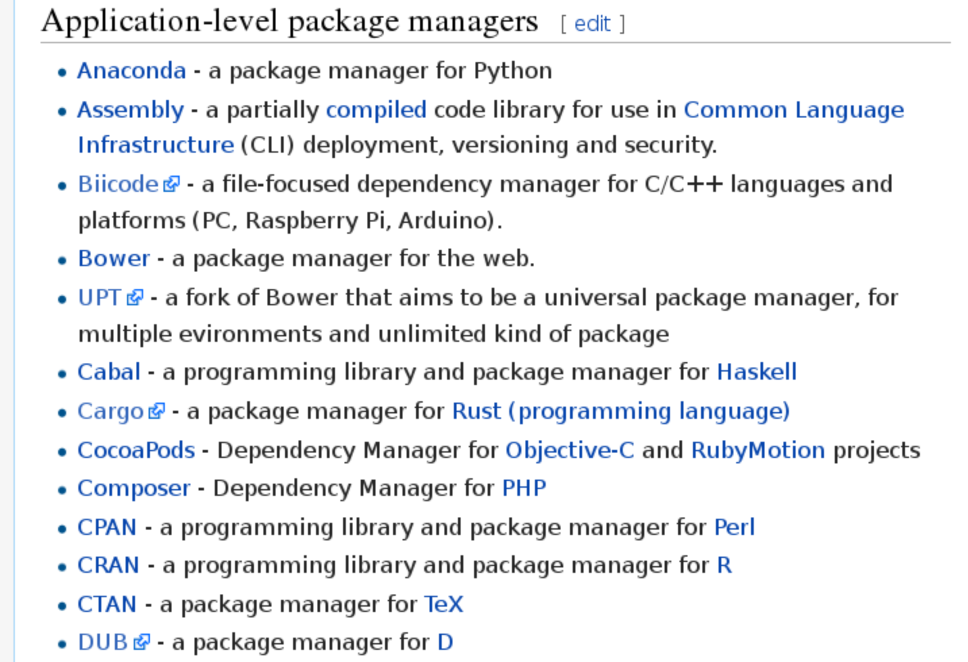
state 2_b

`apt-get install foo`

state 3_b

= ?

#3. It’s worse than this.



Application-level package managers [edit]

- 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 evironments 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.

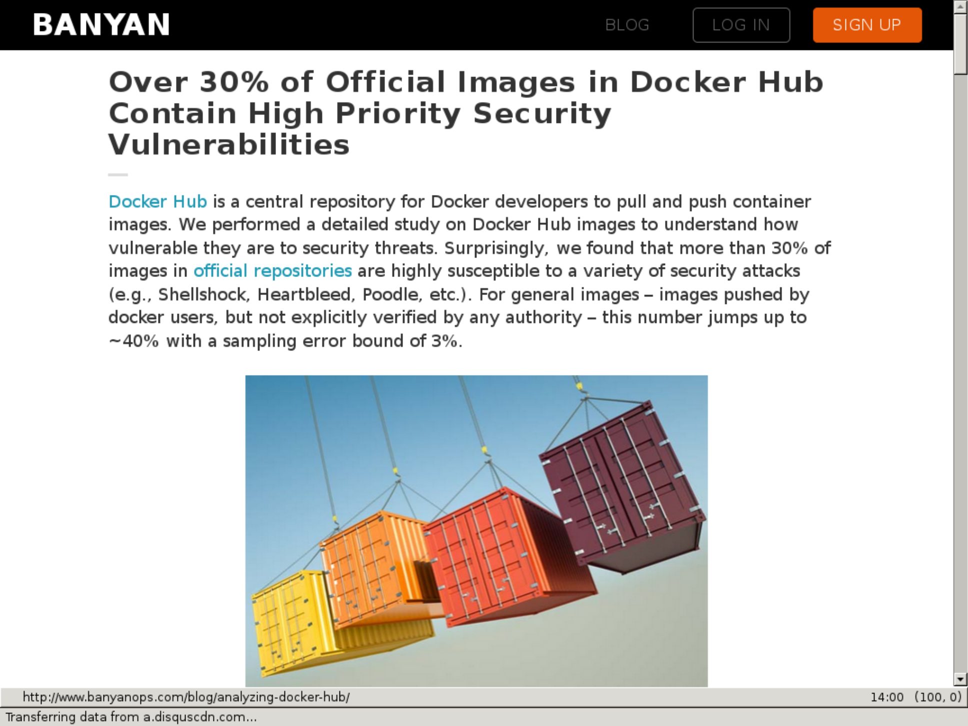
“Let’s Package jQuery: A Javascript Packaging Dystopian Novella” by Chris Webber

`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%.

http://www.banyanops.com/blog/analyzing-docker-hub/ 14:00 (100, 0)

Transferring data from a.disquscdn.com...

HOPE

Functional package management.

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$

where $f =$ `./configure && make && make install`

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

$$
\begin{aligned}
&\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils}) \\
&\texttt{hwloc} = g(\texttt{pciaccess}, \texttt{gcc}, \texttt{make}, \texttt{coreutils}) \\
&\texttt{gcc} = h(\texttt{make}, \texttt{coreutils}, \texttt{gcc}_0) \\
&\ldots
\end{aligned}
$$

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$

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

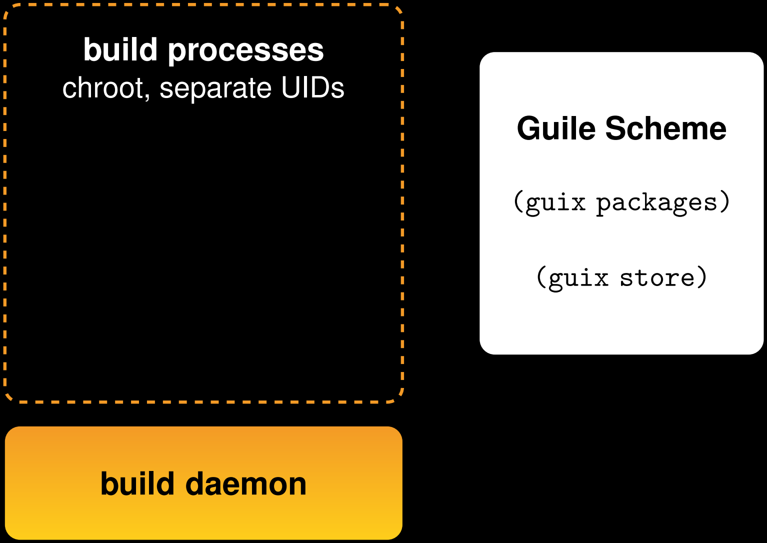
$$\texttt{gcc} = h(\texttt{make}, \texttt{coreutils}, \texttt{gcc}_0)$$

$$\ldots$$

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

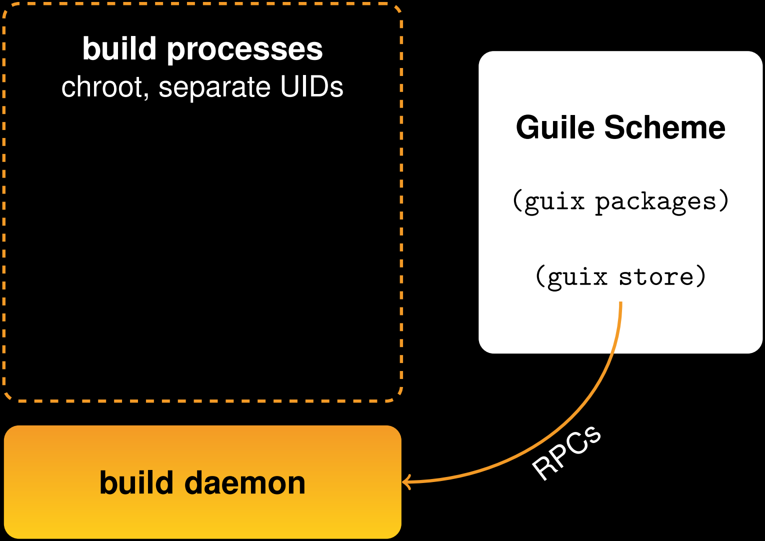
chroot, separate UIDs

Guile Scheme

`(guix packages)`

`(guix store)`

build daemon



build processes

chroot, separate UIDs

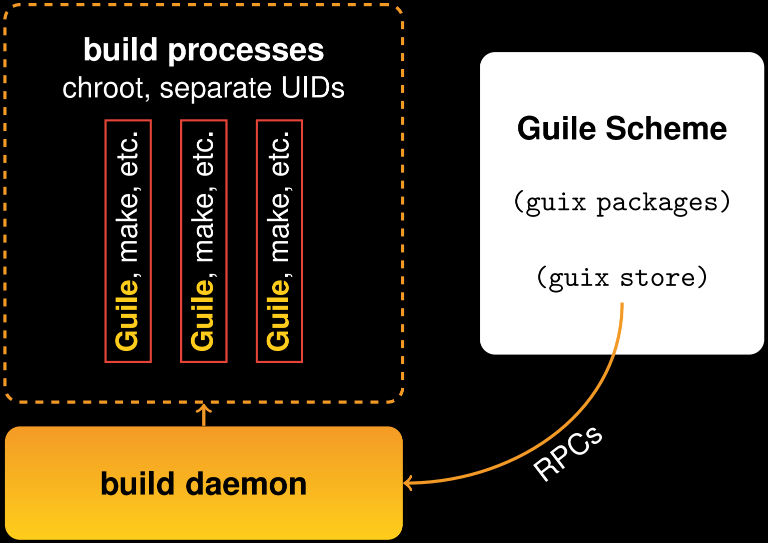
Guile Scheme

`(guix packages)`

`(guix store)`

RPCs

build daemon



build processes

chroot, separate UIDs

Guile, make, etc.

Guile, make, etc.

Guile, make, etc.

Guile Scheme

`(guix packages)`

`(guix store)`

RPCs

build daemon

```
$ 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

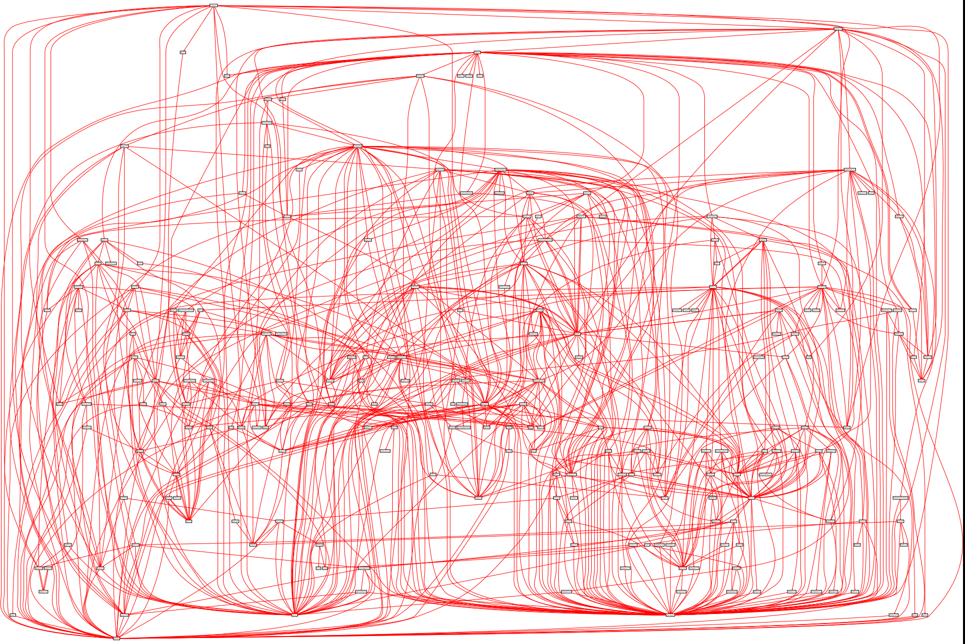
```
$ guix package -i gcc-toolchain coreutils sed grep
…

$ eval ‘guix package --search-paths‘
…

$ guix package --manifest=my-software.scm
…
```

demo

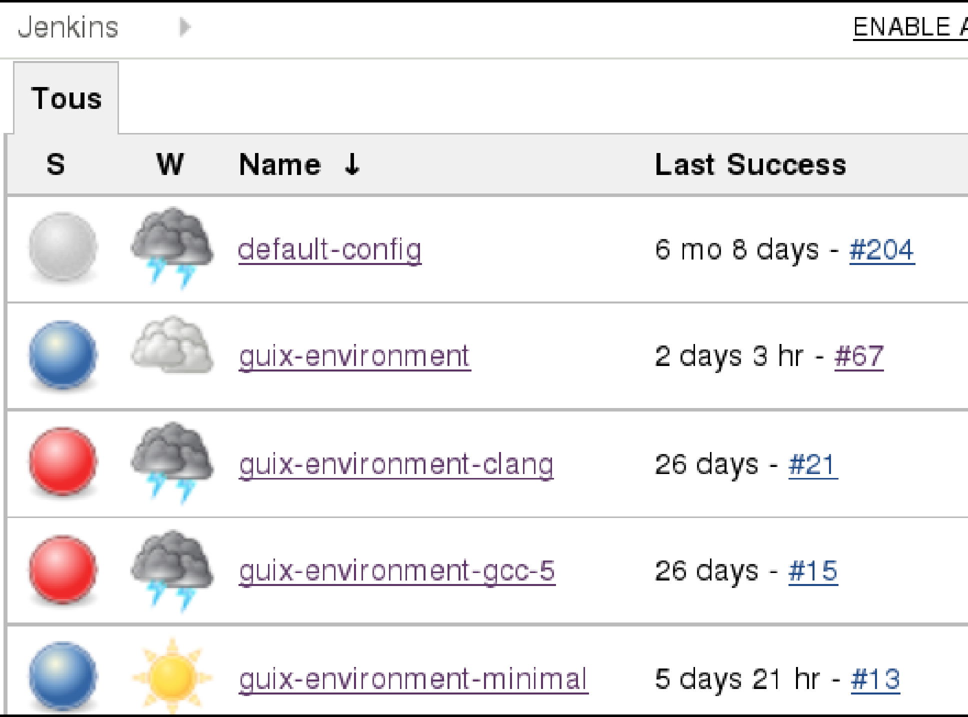
Want your PhD student to hack on GNUnet?

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
…
```



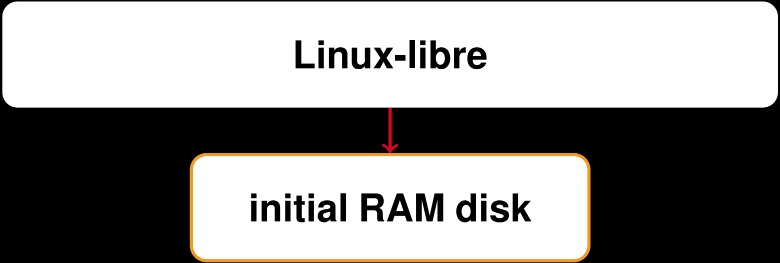
Jenkins ▶ ENABLE A

Tous

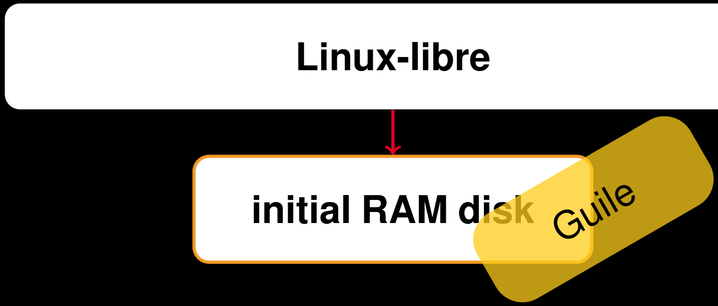
S	W	Name ↓	Last Success
⚪	⛈	default-config	6 mo 8 days - #204
🔵	☁	guix-environment	2 days 3 hr - #67
🔴	⛈	guix-environment-clang	26 days - #21
🔴	⛈	guix-environment-gcc-5	26 days - #15
🔵	☀	guix-environment-minimal	5 days 21 hr - #13

Whole-system deployment.

Linux-libre


Linux-libre

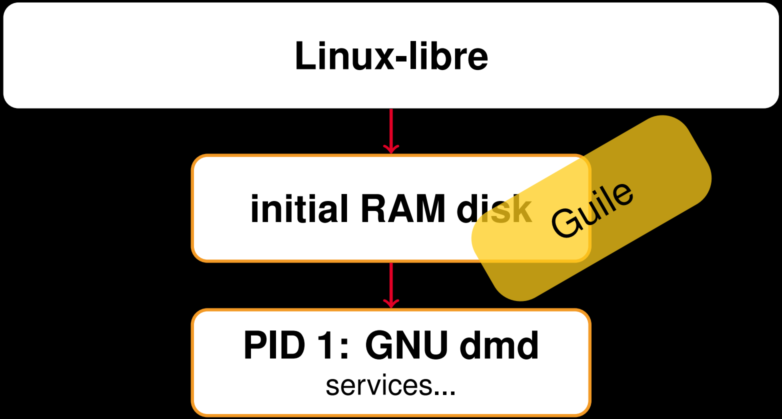
initial RAM disk



Linux-libre

initial RAM disk

Guile



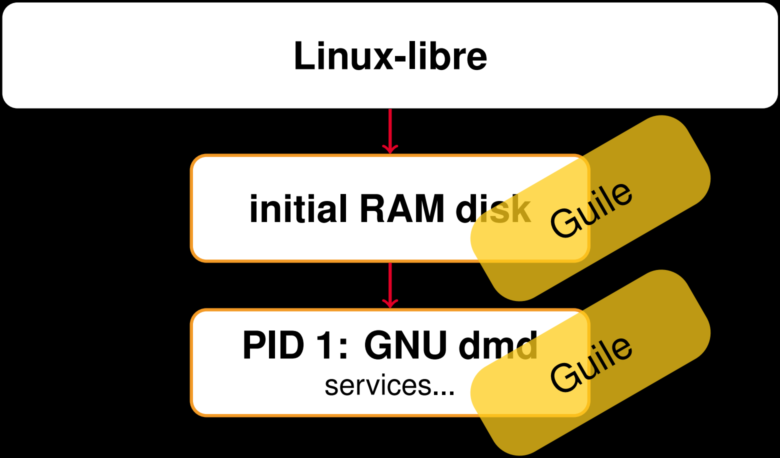
Linux-libre

initial RAM disk

Guile

PID 1: GNU dmd

services...



Linux-libre

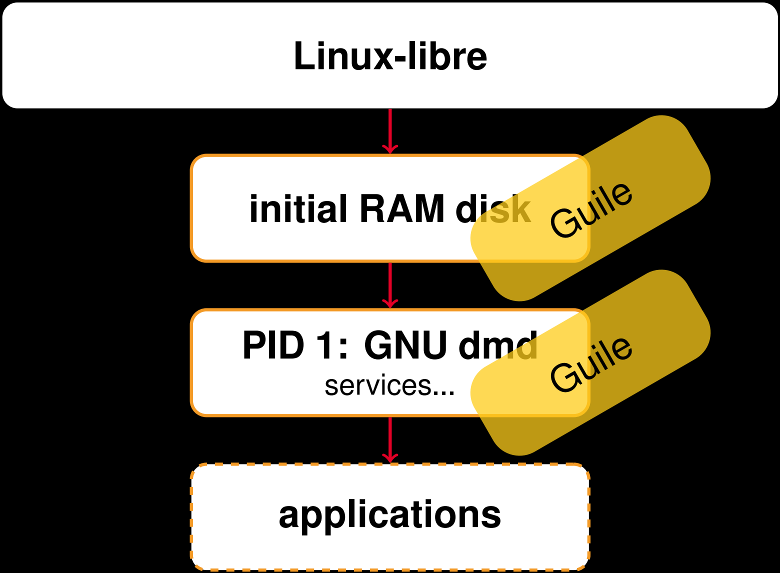
initial RAM disk

Guile

PID 1: GNU dmd

services...

Guile



Linux-libre

initial RAM disk

Guile

PID 1: GNU dmd

services...

Guile

applications

Trustworthiness.

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

— Lucas Nussbaum, FOSDEM 2015

Transparent binary/source deployment

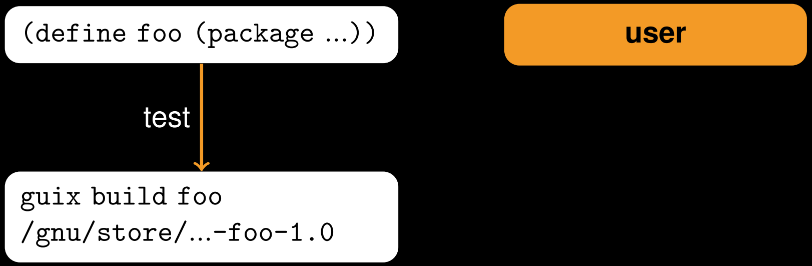
```
alice@foo$ guix package --install=emacs
The following package will be installed:
   emacs-24.5 /gnu/store/...-emacs-24.5

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 ...))
```

user

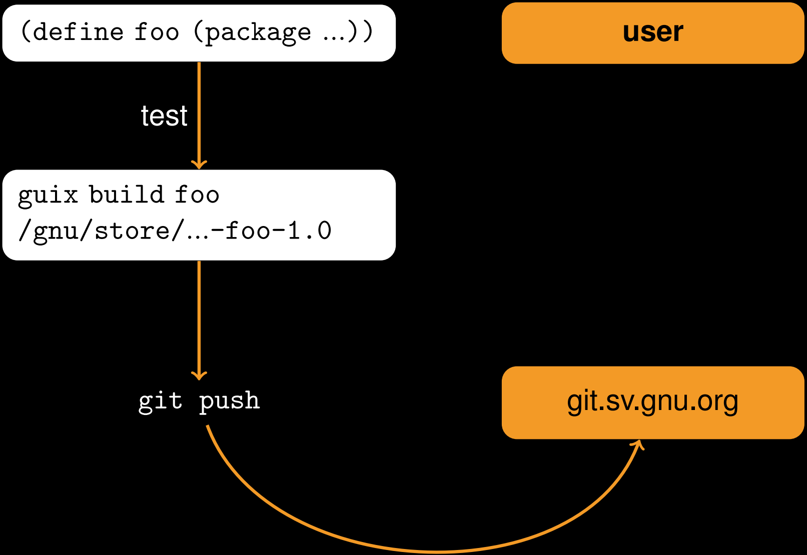


user

```
(define foo (package ...))
```

test

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

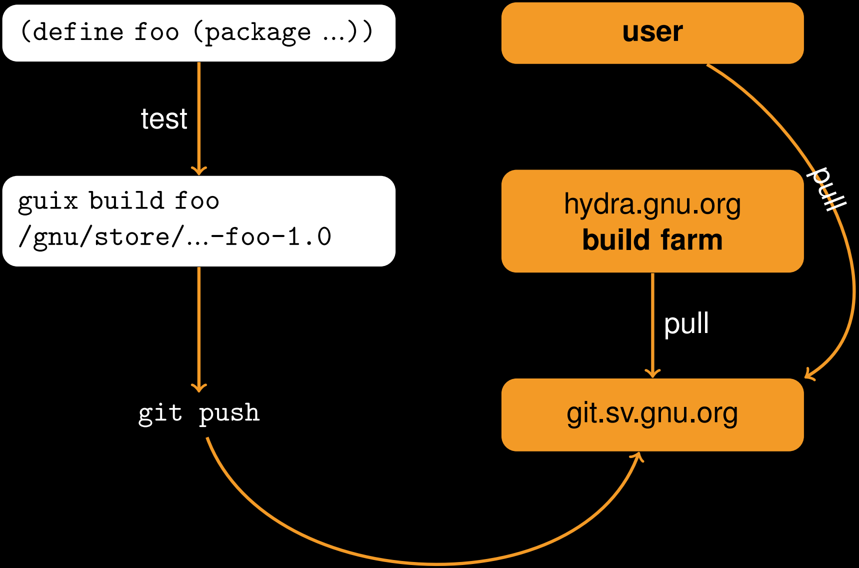


```
(define foo (package ...))
```

test

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

```
git push
```



```
(define foo (package …))
```

test

```
guix build foo
/gnu/store/…-foo-1.0
```

```
git push
```

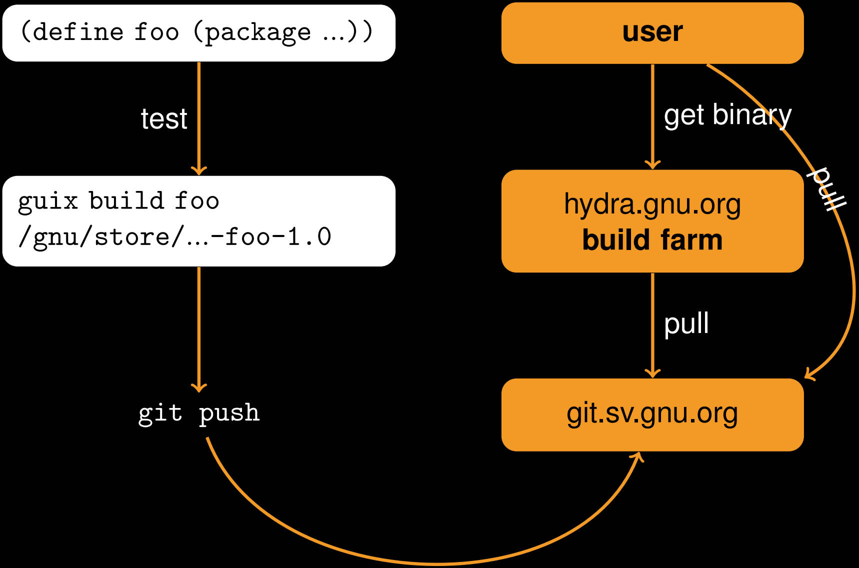
user

pull

hydra.gnu.org
build farm

pull

git.sv.gnu.org



```
(define foo (package ...))
```

test

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

```
git push
```

user

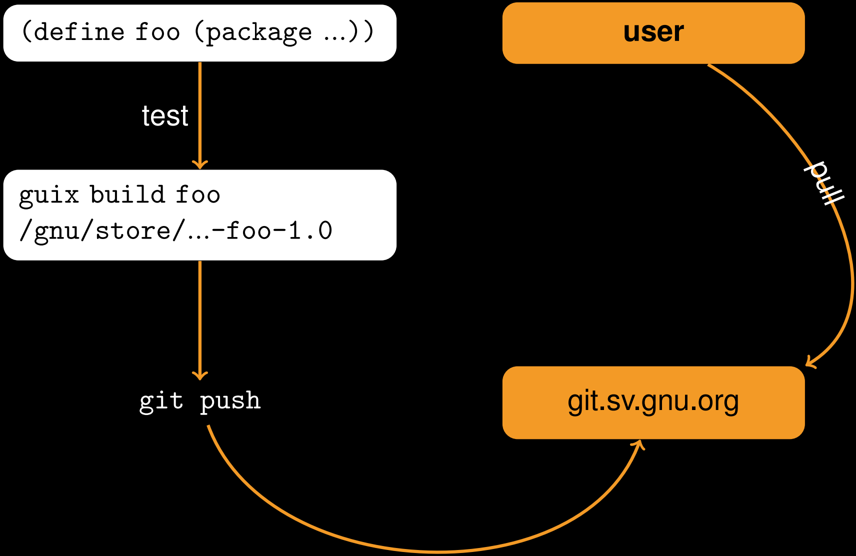
get binary

pull

hydra.gnu.org
build farm

pull

git.sv.gnu.org



```
(define foo (package ...))
```

test

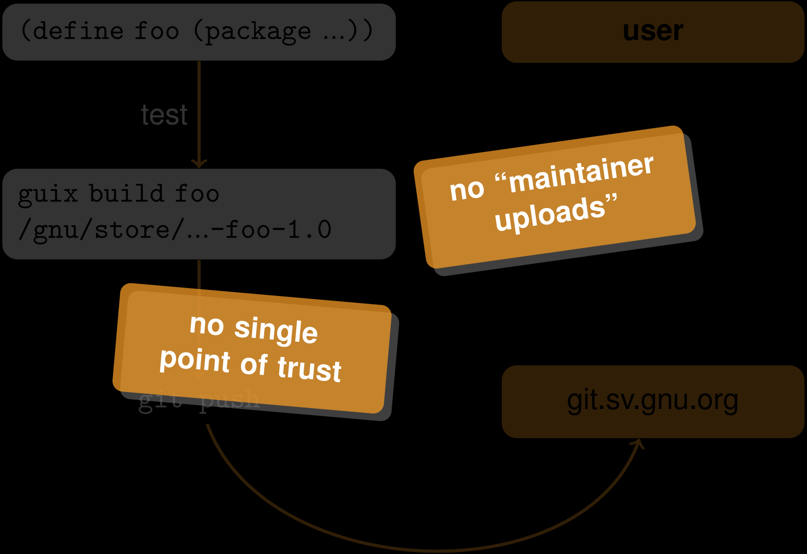
```
guix build foo
/gnu/store/...-foo-1.0
```

```
git push
```

user

pull

git.sv.gnu.org



```
(define foo (package …))
```

test

```
guix build foo
/gnu/store/…-foo-1.0
```

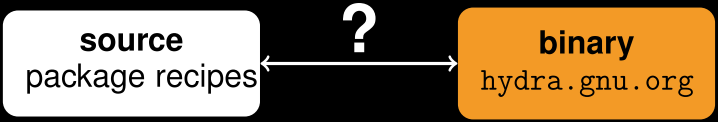
git push

user

git.sv.gnu.org

no “maintainer uploads”

no single point of trust



source
package recipes

?

binary
`hydra.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`
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

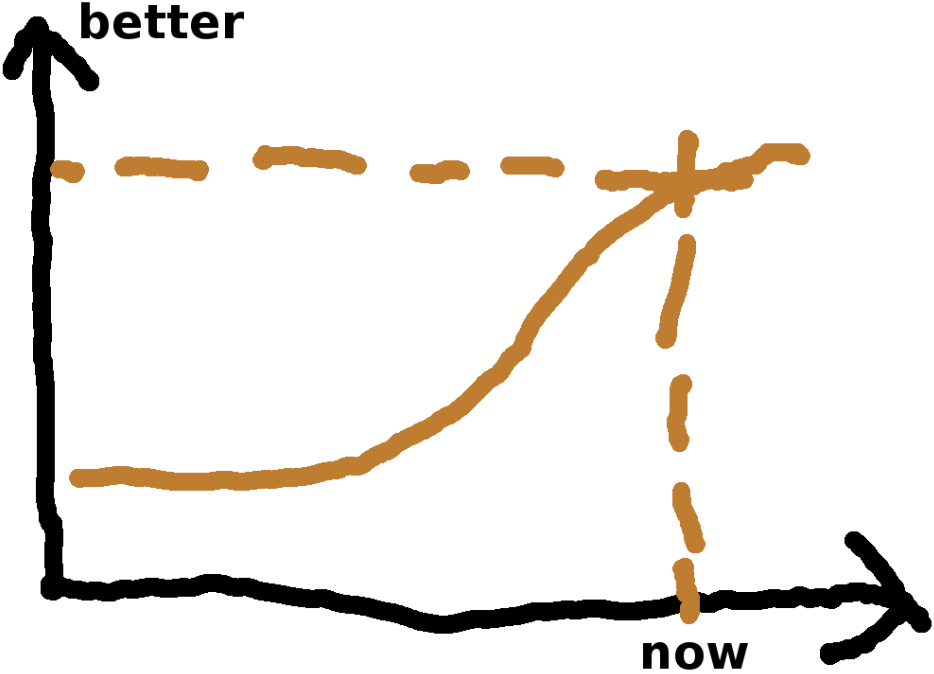
1. **Bit-reproducible builds**
   - we have **isolated build environments**!
   - ... but we need builds to be **deterministic**
   - `http://reproducible-builds.org`
2. **No single binary provider**
   - `guix publish`
   - publish over GNUnet? (GSoC 2015)
3. **Tools for users to challenge binaries**

```
$ guix challenge --substitute-urls="http://hydra.gnu.org ht
/gnu/store/...-openssl-1.0.2d contents differ:
  local hash: 0725l22...
  http://hydra.gnu.org/...-openssl-1.0.2d: 0725l22...
  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.


better
now

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**!

GuixSD

ludo@gnu.org http://gnu.org/software/guix/