Functional Package Management with Guix

Ludovic Courtès
ludo@gnu.org

European Lisp Symposium
3 June 2013, Madrid
¡Hola!

(guile)
¡Hola!

(iguile)

NixOS
¡Hola!
what's Guix?

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

▶ functional package manager
  ▶ written in Guile Scheme
  ▶ a new programming layer for Nix
what's Guix?

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

- functional package manager
  - written in Guile Scheme
  - a new programming layer for Nix
- GNU's package manager
  - foundation for the GNU System
  - GNU(/Linux) distro, est. 2012
  - focus on user freedom + consistent user interface
so what’s Nix?
http://nixos.org/nix/

- another **functional package manager**
- basis of Guix
- foundation of NixOS GNU/Linux
  - GNU/Linux distro, est. 2006
  - i686, x86_64, armv5tel
  - $\approx 8000$ packages
so what’s Nix?
http://nixos.org/nix/

- another **functional package manager**
- basis of Guix
- foundation of NixOS GNU/Linux
  - GNU/Linux distro, est. 2006
  - i686, x86_64, armv5tel
  - ≈8000 packages
- more on Nix later...
Guix’s main contributions

1. package description language **embedded in Scheme**

2. **build programs** written in Scheme
Guix’s main contributions

1. package description language embedded in Scheme
   ▶ benefit from Guile’s tooling (compiler, i18n, etc.)
   ▶ leverage Scheme macros for domain-specific languages

2. build programs written in Scheme
Guix’s main contributions

1. package description language **embedded in Scheme**
   - benefit from Guile’s **tooling** (compiler, i18n, etc.)
   - leverage Scheme macros for **domain-specific languages**

2. **build programs** written in Scheme
   - more **expressive** than Bash (!)
   - a single programming language → **two-tier system**
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
functional package management

features

foundations

Nix's approach

from Nix to Guix

rationale

programming interfaces

builder-side code

discussion
per-user, unprivileged package installation

alice@foo$ guix package --install=gcc

bob@foo$ guix package --install=gcc-4.7.3
per-user, unprivileged package installation

alice@foo$ guix package --install=gcc

bob@foo$ guix package --install=gcc-4.7.3
per-user, unprivileged package installation

alice@foo$ guix package --install=gcc
alice@foo$ guix gc --references ‘which gcc’
/nix/store/...-glibc-2.17
/nix/store/...-gcc-4.8.0
...

bob@foo$ guix package --install=gcc-4.7.3
per-user, unprivileged package installation

alice@foo$ guix package --install=gcc
alice@foo$ guix gc --references 'which gcc'
/nix/store/...-glibc-2.17
/nix/store/...-gcc-4.8.0
...

bob@foo$ guix package --install=gcc-4.7.3
bob@foo$ guix gc --references 'which gcc'
/nix/store/...-glibc-2.13
/nix/store/...-gcc-4.7.3
...
alice@foo$ guix package --install=emacs
The following package will be installed:
  emacs-24.3 out /nix/store/...-emacs-24.3

The following files will be downloaded:
  /nix/store/...-emacs-24.3
  /nix/store/...-libxpm-3.5.10
  /nix/store/...-libxext-1.3.1
  /nix/store/...-libxaw-1.0.11
alice@foo$ guix package --install=emacs
The following package will be installed:
   emacs-24.3 out /nix/store/...-emacs-24.3

The following files will be downloaded:
   /nix/store/...-libxext-1.3.1
   /nix/store/...-libxaw-1.0.11
The following derivations will be built:
   /nix/store/...-emacs-24.3.drv
   /nix/store/...-libxpm-3.5.10.drv
transactional upgrades

$ guix package --upgrade
The following packages will be installed:
  hop-2.4.0 out /nix/store/...-hop-2.4.0
  gdb-7.6 out /nix/store/...-gdb-7.6
  geiser-0.4 out /nix/store/...-geiser-0.4
  glibc-2.17 out /nix/store/...-glibc-2.17
  guile-2.0.9 out /nix/store/...-guile-2.0.9
...

(interrupted right in the middle)

$ hop --version ; guile --version
Hop-
guile (GNU Guile)
transactional upgrades

$ guix package --upgrade
The following packages will be installed:
  hop-2.4.0 out /nix/store/...-hop-2.4.0
  gdb-7.6 out /nix/store/...-gdb-7.6
  geiser-0.4 out /nix/store/...-geiser-0.4
  glibc-2.17 out /nix/store/...-glibc-2.17
  guile-2.0.9 out /nix/store/...-guile-2.0.9
...

$ hop --version ; guile --version
Hop-2.4.0
guile (GNU Guile) 2.0.9
transactional upgrades

$ guix package --upgrade
The following packages will be installed:
  hop-2.4.0 out /nix/store/...-hop-2.4.0
  gdb-7.6 out /nix/store/...-gdb-7.6
  geiser-0.4 out /nix/store/...-geiser-0.4
  glibc-2.17 out /nix/store/...-glibc-2.17
  guile-2.0.9 out /nix/store/...-guile-2.0.9
...

(interrupted right in the middle)

$ hop --version ; guile --version
Hop-...
guile (GNU Guile)
transactional upgrades

$ guix package --upgrade
The following packages will be installed:
  hop-2.4.0 out /nix/store/...-hop-2.4.0
  gdb-7.6 out /nix/store/...-gdb-7.6
  geiser-0.4 out /nix/store/...-geiser-0.4
  glibc-2.17 out /nix/store/...-glibc-2.17
  guile-2.0.9 out /nix/store/...-guile-2.0.9
...
(interrupted right in the middle)

$ hop --version ; guile --version
Hop-1.3.1
guile (GNU Guile) 1.8.8
transactional upgrades

```bash
$ guix package --upgrade
The following packages will be installed:
  hop-2.4.0 out /nix/store/...-hop-2.4.0
  gdb-7.6 out /nix/store/...-gdb-7.6
  geiser-0.4 out /nix/store/...-geiser-0.4
  glibc-2.17 out /nix/store/...-glibc-2.17
  guile-2.0.9 out /nix/store/...-guile-2.0.9
...
(interrupted right in the middle)

$ hop --version ; guile --version
Hop-1.3.1
guile (GNU Guile) 1.8.8
```
per-user rollback

$ emacs --version
GNU Emacs 24.2
per-user rollback

$ emacs --version
GNU Emacs 24.2

$ guix package --upgrade=emacs
The following packages will be installed:
  emacs-24.3.1 out /nix/store/...-emacs-24.3.1
...

$ emacs --version
GNU Emacs 24.2

$ guix package --roll-back
switching from generation 43 to 42

$ emacs --version
GNU Emacs 24.2
per-user rollback

$ emacs --version
GNU Emacs 24.2

$ guix package --upgrade=emacs
The following packages will be installed:
   emacs-24.3.1 out /nix/store/...-emacs-24.3.1
   ...

$ emacs --version
Segmentation Fault
per-user rollback

$ emacs --version
GNU Emacs 24.2

$ guix package --upgrade=emacs
The following packages will be installed:
  emacs-24.3.1 out /nix/store/...-emacs-24.3.1
...

$ emacs --version
Segmentation Fault

$ guix package --roll-back
switching from generation 43 to 42
per-user rollback

$ emacs --version
GNU Emacs 24.2

$ guix package --upgrade=emacs
The following packages will be installed:
  emacs-24.3.1 out /nix/store/...-emacs-24.3.1
...

$ emacs --version
Segmentation Fault

$ guix package --roll-back
switching from generation 43 to 42

$ emacs --version
GNU Emacs 24.2
functional package management
features
foundations
Nix's approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
functional package management

regarding the build & installation process of a package as a pure function
controlling the build environment
... as pioneered by Nix

1. one directory per installed package
2. immutable installation directories
3. undeclared dependencies invisible to the build process
4. build performed in chroot, with separate UID, etc.
the store

/nix/store

im276ak...-glibc-2.16 → lib → libc.so.6

/sbin → lshd

l9w6773...-lsh-2.0.4

/bin → lsh

smkabrb...-gnutls-3.0.18 → lib → libgnutls.so

c6jbqmq2...-emacs-24.1 → bin → emacs
user environments

$PATH
/nix/.../profiles
  current
  42

/nix/store
  pp56i0a01si5...-user-env
    bin
      icecat
      ssh
  l9w6773m1msy...-openssh-4.6p1
    bin
      ssh
      rpdqxnilb0cg...-icecat-3.5.4
        bin
          icecat
guix package --upgrade=openssh
user environments

```sh
PATH

/nix/store
└── pp56i0a01si5...-user-env
    └── bin
        └── icecat
        └── ssh

/l9w6773m1msy...-openssh-4.6p1
    └── bin
        └── ssh

/rpdqxnilb0cg...-icecat-3.5.4
    └── bin
        └── icecat

/aqn3wyggq9jzk...-openssh-5.2p1
    └── bin
        └── ssh

/i3d9vh6d8ip1...-user-env
    └── bin
        └── ssh

guix package --upgrade=openssh
```
user environments

$PATH

/nix/store
  pp56i0a01si5...-user-env
    bin
    icecat
    ssh
  l9w6773m1msy...-openssh-4.6p1
    bin
    ssh
  rpdqxnilb0cg...-icecat-3.5.4
    bin
    icecat
  aqn3wygq9jzk...-openssh-5.2p1
    bin
    ssh
  i3d9vh6d8ip1...-user-env
    bin
    ssh
    icecat

guix package --upgrade=openssh
user environments

$PATH

/nix/store

- pp56i0a01si5...-user-env
  - bin
    - icecat
    - ssh
  - l9w6773m1msy...-openssh-4.6p1
    - bin
      - ssh
  - rpdqxnilb0cg...-icecat-3.5.4
    - bin
      - icecat
  - aqn3wygq9jzk...-openssh-5.2p1
    - bin
      - ssh
  - i3d9vh6d8ip1...-user-env
    - bin
      - ssh
      - icecat

guix package --upgrade=openssh
user environments

PATH

/nix/.../profiles

current

43

/nix/store

-rpdqxnilb0cg...-icecat-3.5.4

-bin

-icecat

-aqn3wygq9jzk...-openssh-5.2p1

-bin

-ssh

-i3d9vh6d8ip1...-user-env

-bin

-ssh

-icecat

guix gc
store file names

$ guix build guile
store file names

$ guix build guile
/nix/store/h2g4sc09h4...-guile-2.0.9

hash of all the dependencies
$ guix build guile
/nix/store/ h2g4sc09h4... -guile-2.0.9

$ guix gc --references /nix/store/...-guile-2.0.9
/nix/store/4jl83jgzaac...-glibc-2.17
/nix/store/iplay43cg58...-libunistring-0.9.3
/nix/store/47p47v92cj9...-libffi-3.0.9
/nix/store/drkwck2j965...-gmp-5.0.5
...

store file names
complete dependency specification

build-time dependencies of GNU Hello
complete dependency specification

build-time dependencies of GNU Hello

... down to the compiler’s compiler!
complete dependency specification

run-time dependencies of GNU Hello

linux-headers-2.6.28.5

/libc-2.11

hello-2.3

run-time dependencies inferred by conservative scanning
functional packaging summarized

- **immutable** software installations
- builds/installs have **no side effects**
- build & deployment \(\equiv\) **calling a build function**
- the store \(\equiv\) **memoization**
- garbage collection...
functional package management
features
foundations
**Nix’s approach**

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
Nix is twofold

functional package deployment

- the store
- file name hashes
- user environments
- transactional upgrades, etc.
- ...
Nix is twofold

- functional package deployment
  - the store
  - file name hashes
  - user environments
  - transactional upgrades, etc.
  - ...

- Nix packaging language
  - to describe package composition
  - external DSL
  - dynamically-typed, lazy
  - easy integration of Bash snippets
  - ...

Nix multi-user setup

build processes
chroot, separate UIDs

Nix tools
interpreter
client lib

Nix build daemon
Nix multi-user setup

- build processes
  - chroot, separate UIDs

- Nix tools
  - interpreter
  - client lib

- Nix build daemon

RPCs
Nix multi-user setup

- build processes
  - chroot, separate UIDs
- Bash, make, etc.
- Bash, make, etc.
- Bash, make, etc.

Nix tools
- interpreter
- client lib

Nix build daemon

RPCs
Nix language build primitive

derivation  {
    name = "foo";
    system = "x86_64-linux";
    builder = "${./static-bash}";
    args = [ "-c" "echo hello > "$out" ];
}


Nix language build primitive

derivation {
  name = "foo";
  system = "x86_64-linux";
  builder = "${./static-bash}";
  args = [ "-c" "echo hello > "$out" ];
}

function call

/nix/store/...-foo

named arguments
Nix language build primitive

```nix
let dep = derivation {
    name = "foo";
    system = "x86_64-linux";
    builder = "${./static-bash}";
    args = [ "-c" "echo hello > "$out" " ];
};
in derivation {
    name = "bar";
    system = "x86_64-linux";
    builder = "${./static-bash}";
    args = [ "-c"
        '' mkdir -p "$out"
        ln -s "${dep} /some-result" "$out/my-result"
        '' ];
    PATH = "${coreutils}/bin";
}
```
Nix language build primitive

```nix
let dep = derivation {
    name = "foo";
    system = "x86_64-linux";
    builder = "${./static-bash}";
    args = [ "-c" "echo hello > "$out" ];
};

in derivation {
    name = "bar";
    system = "x86_64-linux";
    builder = "${./static-bash}";
    args = [ "-c"
        "mkdir -p "$out"
        ln -s "${dep} /some-result" "$out/my-result"
        ];
    PATH = "${coreutils}/bin";
}
```

expands to /nix/store/...-foo
{ fetchurl, stdenv }:

stdenv . mkDerivation {
    name = "hello-2.3";
    src = fetchurl {
        url = mirror://gnu/hello/hello-2.3.tar.bz2;
        sha256 = "0c7vijq8y68...";
    };

    buildInputs = [ gettext ];
    preCheck = "echo 'Test suite coming up!';"
};

meta = {
    description = "Produces a friendly greeting";
    homepage = http://www.gnu.org/software/hello/;
    license = "GPLv3+";
};
Nix language high-level packaging

```nix
{ fetchurl, stdenv, gettext }:

stdenv.mkDerivation {
  name = "hello-2.3";
  src = fetchurl {
    url = mirror://gnu/hello/hello-2.3.tar.bz2;
    sha256 = "0c7vijq8y68...";
  };
  buildInputs = [ gettext ];
}

meta = {
  description = "Produces a friendly greeting";
  homepage = http://www.gnu.org/software/hello/;
  license = "GPLv3+";
};
```
Nix language high-level packaging

```{fetchurl, stdenv, gettext}

stdenv.mkDerivation {
    name = "hello-2.3";
    src = fetchurl {
        url = mirror://gnu/hello/hello-2.3.tar.bz2;
        sha256 = "0c7vijq8y68...";
    };
    buildInputs = [ gettext ];
    preCheck = "echo 'Test suite coming up!’";
    meta = {
        description = "Produces a friendly greeting";
        homepage = http://www.gnu.org/software/hello/;
        license = "GPLv3+";
    };
}
```
and now for parentheses...
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
The truth is that Lisp is not the right language for any particular problem. Rather, Lisp encourages one to attack a new problem by implementing new languages tailored to that problem.

– Albelson & Sussman, 1987
from Nix...

functional package deployment

- the store
- file name hashes
- user environments
- transactional upgrades, etc.
- ...

Nix packaging language

- to describe package composition
- external DSL
- dynamically-typed, lazy
- easy integration of Bash snippets
- ...

...
from Nix to Guix

functional package deployment

- the store
- file name hashes
- user environments
- transactional upgrades, etc.
- ...

reuse this

- Nix packaging language

- to describe package composition
- external DSL
- dynamically-typed, lazy
- easy integration of Bash snippets
- ...

Scheme!
Guix architecture

build processes
chroot, separate UIDs

Guile, make, etc.
Guile, make, etc.
Guile, make, etc.

Guile

(guix packages)
(guix store)

Nix build daemon

RPCs
thesis

1. Scheme + EDSL at least as expressive as the Nix language
2. Scheme better suited than the shell for build programs
3. Guix provides a unified & extensible programming environment
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
programming interface layers

1. **declarative** packaging layer
2. Scheme **build expressions**
3. derivation **primitive** (from Nix)
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
      (method url-fetch)
      (uri (string-append
        "http://ftp.gnu.org/.../hello-" version ".tar.gz"))
      (sha256 (base32 "0wqd...dz6")))))
  (build-system gnu-build-system)
  (synopsis "GNU Hello")
  (description "Produce a friendly greeting.")
  (home-page "http://www.gnu.org/software/hello/")
  (license gpl3+)))
how do we reach this level of abstraction?
Nix’s derivation primitive in Scheme

(let* ((store (open-connection))
        (bash (add-to-store store "static-bash"
                          #t "sha256"
                          "/static-bash")))

(derivation store "example-1.0"
             "x86_64-linux"
             bash
             '("-c" "echo hello > $out")
             '((("HOME" . "/homeless")
                  '()))

=> "/nix/store/nsswy...-example-1.0 .drv"
=> #<derivation "example-1.0" ...>
Nix's derivation primitive in Scheme

(let* ((store (open-connection))
       (bash (add-to-store store "static-bash"
                     #t "sha256"
                     "/static-bash")))

(derivation store "example-1.0"
 "x86_64-linux"
 bash
 '("-c" "echo hello > $out")
 '(("HOME" . "/homeless")
  ()
  ))

=> "/nix/store/nsswy...-example-1.0 .drv "
=> #<derivation "example-1.0" ...>
Nix's derivation primitive in Scheme

(let* ((store (open-connection))
        (bash (add-to-store store "static-bash"
                     #t "sha256"
                     "./static-bash")))

(derivation store "example-1.0"
             "x86_64-linux"
             bash /
             /nix/store/...-static-bash
             '("-c" "echo hello > $out")
             '(("HOME" ./ /homeless"))
             '()))

=> "/nix/store/nsswy...-example-1.0 .drv"
=> #<derivation "example-1.0" ...>
Nix’s derivation primitive in Scheme

(let* ((store (open-connection))
        (bash (add-to-store store "static-bash"
                         #t "sha256"
                         ".\/static-bash")))

(derivation store "example-1.0"
     "x86_64-linux"
     bash
     "echo hello > $out"
     "HOME" . "/homeless"
     )

=> "/nix/store/nsswy...-example-1.0 .drv"
=> #<derivation "example-1.0" ...>
(let* ((store (open-connection))
       (builder '(begin
                  (mkdir %output)
                  (call-with-output-file
                   (string-append %output="/test")
                   (lambda (p)
                     (display '(hello guix) p))))))
(drv (build-expression->derivation
      store "foo" "x86_64-linux"
      builder
      '((("HOME" . "~/nowhere")))))
(build-derivations store (list drv)))
build expressions

(let* ((store (open-connection))

  (builder '(begin
              (mkdir %output)
              (call-with-output-file
               (string-append %output "\test")
               (lambda (p)
                  (display '(hello guix) p))))))

(drv (build-expression->derivation
       store "foo" "x86_64-linux"
       builder
       '((("HOME" . "/nowhere"))))

  (build-derivations store (list drv)))
compute derivation for this builder, system, and env. vars

(let* ((store (open-connection)))
  (builder '( begin
              (mkdir %output)
              (call-with-output-file
               (string-append %output "/test")
               (lambda (p)
                (display '(hello guix) p))))))
  (drv (build-expression->derivation
        store "foo" "x86_64-linux"
        builder
        '(("HOME" . "/nowhere"))))
  (build-derivations store (list drv)))
(let* ((store (open-connection)) )

(builder '( begin

  (mkdir %output)
  (call-with-output-file
   (string-append %output "\test")
   (lambda (p)
     (display '(hello guix) p)))))))

(drv ( build-expression->derivation
       store "foo" "x86_64-linux"
       builder
       '((("HOME" . "/nowhere")))))

(build-derivations store (list drv)))
build expressions

(let* ((store  (open-connection))
       (builder  '(begin
                   (mkdir %output)
                   (call-with-output-file
                    (string-append %output "/test")
                    (lambda (p)
                    (display '(hello guix) p))))))

(build-derivations store (list drv))

(build it!)
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
      (method url-fetch)
      (uri (string-append
            "http://ftp.gnu.org/.../hello-
            " version
            ".tar.gz"))
      (sha256 (base32 "0wqd...dz6"))))
  (build-system gnu-build-system )
  (inputs '(('"gawk" , gawk )))
  (synopsis "GNU Hello")
  (description "Produce a friendly greeting.")
  (home-page "http://www.gnu.org/software/hello/")
  (license gpl3+))))
(define hello
  (package
   (name "hello")
   (version "2.8")
   (source (origin
             (method url-fetch)
             (uri (string-append
                   "http://ftp.gnu.org/.../hello-" version
                   ".tar.gz"))
             (sha256 (base32 "0wqd...dz6"))))
   (build-system gnu-build-system ))
  (inputs `(('"gawk" , gawk ))))
  (synopsis "GNU Hello")
  (description "Produce a friendly greeting")
  (home-page "http://www.gnu.org/software/hello/")
  (license gpl3+)))
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
      (method url-fetch)
      (uri (string-append
        "http://ftp.gnu.org/.../hello-" version
        ".tar.gz"))
      (sha256 (base32 "0wqd...dz6"))))
    (build-system gnu-build-system)
    (inputs '(("gawk" , gawk )))
    (synopsis "GNU Hello")
    (description "Produce a friendly greeting")
    (home-page "http://www.gnu.org/software/hello/")
    (license gpl3+)))
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
      (method url-fetch)
      (uri (string-append
            "http://ftp.gnu.org/.../hello-
            .tar.gz"))
      (sha256 (base32 "0wqd...dz6"))))
    (build-system gnu-build-system)
    (inputs `(('"gawk" , my-other-awk ))))
    (synopsis "GNU Hello")
    (description "Produce a friendly greeting.")
    (home-page "http://www.gnu.org/software/hello/"
    (license gpl3+)))
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (method url-fetch)
      (uri (string-append
        "http://ftp.gnu.org/.../hello-
        " version
        ".tar.gz"))
      (sha256 (base32 "0wqd...dz6")))
    (build-system gnu-build-system)
    (inputs `(('"gawk" , gawk))))
  (synopsis "GNU Hello")
  (description "Produce a friendly greeting.")
  (home-page "http://www.gnu.org/software/hello/"))
  (license gpl3+)})
(define gnu-build-system
(build-system (name 'gnu)
  (description "/configure && make && make install")
  (build gnu-build)
  (cross-build gnu-cross-build)))
(define gnu-build-system
  (build-system (name 'gnu)
    (description "./configure && make && make install")
    (build gnu-build)
    (cross-build gnu-cross-build)))

- python-build-system → python setup.py
- perl-build-system → perl Makefile.PL
- cmake-build-system → cmake .
(use-modules (guix packages) (guix store)
  (gnu packages base))

(define store
  (open-connection))

(build-derivations (list drv))

connect to the Nix build daemon

package? hello

=> #t
building packages

(use-modules (guix packages) (guix store)
 (gnu packages base))

(define store
  (open-connection))

(package? hello)
=> #t

(define drv (package-derivation store hello))

compute “derivation”—i.e., build promise
building packages

(use-modules (guix packages) (guix store)
  (gnu packages base))

(define store
  (open-connection))

(package? hello)
=> #t

(define drv (package-derivation store hello))
drv
=> "/nix/store/xyz...-hello-2.8.drv"
building packages

(use-modules (guix packages) (guix store)
           (gnu packages base))

(define store
   (open-connection))

(package? hello)
=> #t

(define drv (package-derivation store hello))
drv
=> "/nix/store/xyz...-hello-2.8.drv"

(build-derivations (list drv))
... daemon builds/downloads package on our behalf...
building packages

(use-modules (guix packages) (guix store)
  (gnu packages base))

(define store
  (open-connection))

(package? hello)
=> #t

(define drv (package-derivation store hello))
drv
=> "/nix/store/xyz...-hello-2.8.drv"

(build-derivations (list drv))
... daemon builds/downloads package on our behalf...
=> "/nix/store/pqr...-hello-2.8"
building packages

$ guix build hello
$ guix build hello
The following derivations will be built:
    /nix/store/4gy79...-gawk-4.0.0.drv
    /nix/store/7m2r9...-hello-2.8.drv
...
/nix/store/71aj1...-hello-2.8
$ guix build --target=armel-linux-gnueababi hello
The following derivations will be built:
  /nix/store/1gm99...-gcc-armel-linux-gnu-4.8.1.drv
  /nix/store/71ah1...-hello-2.8.drv
...
/nix/store/7m2r9...-hello-2.8
packages based on existing ones

copy fields from hello except for version and source

(package (inherit hello)
  (version "2.7")
  (source
    (origin
      (method url-fetch)
      (uri "mirror://gnu/hello/hello-2.7.tar.gz")
      (sha256
        (base32 "7dqw3..."))))
(define (static-package p)
  ;; Return a statically-linked variant of P.
  (package (inherit p)
    (arguments
     (#:configure-flags ('("--disable-shared"
                           "LDFLAGS=-static")
                       ,@(package-arguments p))))))
system-dependent arguments

(define gawk
 (package
  (name "gawk")
  (version "4.0.2")
  (source (origin (method url-fetch)
      (uri "http://ftp.gnu.org/...")
      (sha256 (base32 "0sss..."))))
 (build-system gnu-build-system)
 (arguments
  (if (string-prefix? "i686" (%current-system))
    '(:tests? #f) ; skip tests on 32-bit hosts
    '()))
 (inputs `(("libsigsegv" ,libsigsegv)))
 (home-page "http://www.gnu.org/software/gawk/")
 (synopsis "GNU Awk"))
system-dependent arguments

(define gawk
  (package
    (name "gawk")
    (version "4.0.2")
    (source (origin (method url-fetch
                     (uri "http://ftp.gnu.org/...
                     (sha256 (base32 "0sss..."))))))
    (build-system gnu-build-system)
    (arguments
      (if (string-prefix? "i686" (%current-system))
        '(#:tests? #f) ; skip tests on 32-bit hosts
        '()))
    (inputs '(("libsigsegv" ,libsigsegv)))
    (home-page "http://www.gnu.org/software/gawk/")
    (synopsis "GNU Awk")))

dynamically-scoped parameter (SRFI-39)

evaluated within the dynamic extent of package-derivation
under the hood: fancy records

\[
\begin{align*}
\text{(define-record-type* <package>)} \\
& \quad \text{package make-package package?} \\
& \quad \ \text{(name package-name)} \\
& \quad \text{(version package-version)} \\
& \quad \text{(source package-source)} \\
& \quad \text{(build-system package-build-system)} \\
& \quad \text{(arguments package-arguments} \\
& \quad \quad \quad \text{(default '()) (thunked))} \\
& \quad \text{(inputs package-inputs} \\
& \quad \quad \quad \text{(default '()) (thunked))} \\
& \quad \text{)}
\end{align*}
\]

\[
\begin{align*}
& \text{;; ...} \\
& \text{(location package-location} \\
& \quad \text{(default (current-source-location))})
\end{align*}
\]
under the hood: fancy records

```
(define-record-type* <package>
  package  make-package package?

  (name package-name)
  (version package-version)
  (source package-source)
  (build-system package-build-system)
  (arguments package-arguments
    (default '()) (thunked))

  (inputs package-inputs
    (default '()) (thunked))

  ;; ...)

  (location package-location
    (default (current-source-location)))))
```
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
builder side of GNU Build System

```scheme
(define %standard-phases
  `((configure . configure)
   (build . build)
   ;; ...
   ))

(define* (gnu-build #:key (phases %standard-phases)
                   #:allow-other-keys
                   #:rest args)
  ;; Run all the PHASES in order, passing them ARGS.
  (every (match-lambda
           ((name . proc)
            (format #t "starting phase ‘~a’~%" name)
            (let ((result (apply proc args)))
              (format #t "phase ‘~a’ done~%" name) result)))
           phases))
```
inserting a build phase

(define howdy
  (package (inherit hello)
    (arguments
      '(#:phases
        (alist-cons-after
          'configure 'change-hello
          (lambda* (#:key system #:allow-other-keys)
            (substitute* "src/hello.c"
              (("Hello, world!"
                  (string-append "Howdy! Running on "
                    system "."))))
            %standard-phases ))))))
(define howdy
  (package (inherit hello)
    (arguments
      '(:phases
        (alist-cons-after
          'configure 'change-hello
          (lambda* (:key system #:allow-other-keys)
            (substitute* "src/hello.c"
              ("Hello, world!"
               (string-append "Howdy! Running on "
                              system "."))))
        %standard-phases)))
builder-side expression)
inserting a build phase

(define howdy
  (package (inherit hello)
    (arguments
      '(#:phases
         (alist-cons-after
          'configure 'change-hello
          (lambda* (#:key system #:allow-other-keys)
            (substitute* "src/hello.c" "Hello, world!" "Howdy! Running on " system "."))
         %standard-phases )))))
inserting a build phase

(define howdy
  (package (inherit hello)
    (arguments
      '(#:phases
        (alist-cons-after
          'configure 'change-hello
          (lambda* (#:key system #:allow-other-keys)
            (substitute* "src/hello.c"
              ("Hello, world!"
               (string-append "Howdy! Running on "
                              system "."))))
        %standard-phases))))
downloading sources

(origin
    (method url-fetch )
    (uri (string-append "mirror://gnu/gcc/gcc-
        version "/gcc-
        version ".tar.bz2"))
    (sha256 (base32 "1hx9...")))
downloading sources

use Guile HTTP(S)/FTP client

(origin
  (method url-fetch )
  (uri (string-append "mirror://gnu/gcc/gcc-
                    version "/gcc-
                    version ".tar.bz2"))
  (sha256 (base32 "1hx9..."))))
downloading sources

use Guile HTTP(S)/FTP client

(origin
  (method url-fetch)
  (uri (string-append "mirror://gnu/gcc/gcc-
                   version "/gcc-
                   version ".tar.bz2")))

(sha256 (base32 "1hx9...")))

how is the very first tarball downloaded?
bootstrapping the distribution

0. statically-linked binaries of mkdir, tar, xz, bash, and Guile
bootstrapping the distribution

0. statically-linked binaries of `mkdir`, `tar`, `xz`, `bash`, and `Guile`
1. derivation runs Bash script to untar `Guile`
0. statically-linked binaries of `mkdir`, `tar`, `xz`, `bash`, and Guile
1. derivation runs Bash script to untar Guile
2. use Guile to download statically-linked binaries of GCC, Binutils, libc, Coreutils et al., and Bash
bootstrapping the distribution

0. statically-linked binaries of mkdir, tar, xz, bash, and Guile
1. derivation runs Bash script to untar Guile
2. use Guile to download statically-linked binaries of GCC, Binutils, libc, Coreutils et al., and Bash
3. use that to build GNU Make
4. ...
functional package management
features
foundations
Nix’s approach

from Nix to Guix
rationale
programming interfaces
builder-side code

discussion
status

- API/language support for builds & composition
- builder-side libs equiv. to `wget`, `find`, `grep`, `sed`, etc.
- expressive enough to build a variety of packages
benefits of DSL embedding

1. Guile tools readily available

2. simplified implementation of auxiliary tools
benefits of DSL embedding

1. Guile tools readily available
   - libraries, macros, compiler, etc.
   - i18n support (for package descriptions)
   - development environment: Emacs + Geiser

2. simplified implementation of auxiliary tools
   - off-line & on-line package auto-updater
   - description synchronization with external DB
   - searching packages by keyword
GNU/Linux distribution

- installable atop a running GNU/Linux system
- self-contained (pure!)
- transactional upgrade/roll-back, pre-built binaries, etc.
- \( \approx 400 \) packages
  - TeX Live, Xorg, GCC, ...
  - and 6 Scheme implementations! :-(
pushing the limits: booting to Guile

(expression->initrd
  '(begin
      (mkdir "/proc")
      (mount "none" "/proc" "proc")
  
  ;; Load Linux kernel modules.
  (let ((slurp (lambda (module)
                 (call-with-input-file
                  (string-append "/modules/" module)
                  get-bytevector-all)))
        (for-each (compose load-linux-module slurp)
                  (list "md4.ko" "ecb.ko" "cifs.ko")))
  
  ;; Turn eth0 up.
  (let ((sock (socket AF_INET SOCK_STREAM 0)))
    (set-network-interface-flags sock "eth0" IFF_UP))
  
  ;; At last, the warm and friendly REPL.
  (start-repl)
road map

- short-term
  - tweak more packages for cross-compilation
  - port to mips64el (N64), and armel (?)
  - more packages: GTK+ stack, applications
road map

- **short-term**
  - tweak more packages for cross-compilation
  - port to mips64el (N64), and armel (?)
  - more packages: GTK+ stack, applications

- **medium-term**
  - stand-alone, bootable distribution!
  - with NixOS-style whole-system configuration EDSL
  - with the Guile-powered DMD init system
road map

- **short-term**
  - tweak more packages for cross-compilation
  - port to mips64el (N64), and armel (?)
  - more packages: GTK+ stack, applications

- **medium-term**
  - stand-alone, bootable distribution!
  - with NixOS-style whole-system configuration EDSL
  - with the Guile-powered DMD init system

Your help needed!
the first no-compromise GNU distribution
summary

- **features**
  - transactional upgrades; rollback; per-user profiles
  - full power of Guile to build & compose packages
  - unified packaging development environment

- **foundations**
  - purely functional package management
  - packaging DSL embedded in Scheme
  - second tier: flexible builds programs in Scheme