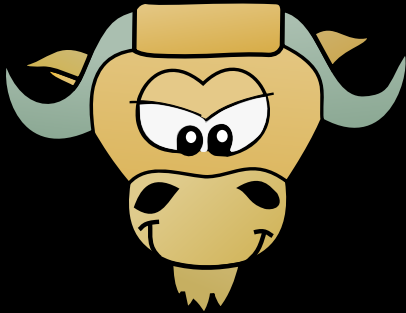


Guix, Functional Package Management for the People

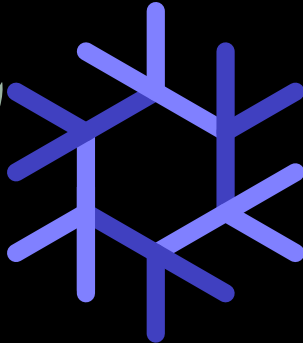
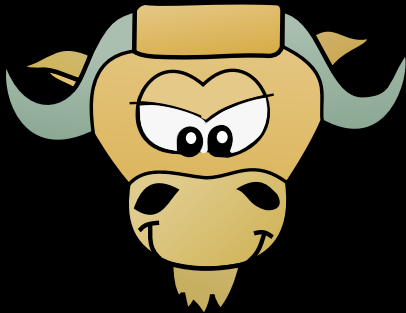
Ludovic Courtès
ludo@gnu.org

GNU Hackers Meeting, July 2012, Düsseldorf

GNUten Tag, Düsseldorf!



GNUten Tag, Düsseldorf!



what's Guix?

<http://gitorious.org/guix/>

- ▶ it's the new thing!
- ▶ IPA: /gi:ks/

what's Guix?

<http://gitorious.org/guix/>

- ▶ it's the new thing!
- ▶ IPA: /gi:ks/
- ▶ **functional package manager!**

what's Guix?

<http://gitorious.org/guix/>

- ▶ it's the new thing!
- ▶ IPA: /gi:ks/
- ▶ **functional package manager!**
- ▶ written in **Guile Scheme!**

what's Guix?

<http://gitorious.org/guix/>

- ▶ it's the new thing!
- ▶ IPA: /gi:ks/
- ▶ **functional package manager!**
- ▶ written in **Guile Scheme!**
- ▶ a new programming layer for **Nix**

what's Guix?

<http://gitorious.org/guix/>

- ▶ it's the new thing!
- ▶ IPA: /gi:ks/
- ▶ **functional package manager!**
- ▶ written in **Guile Scheme!**
- ▶ a new programming layer for **Nix**
- ▶ Nix?

so what's Nix?

<http://nixos.org/nix/>

- ▶ **a functional package manager**

so what's Nix?

<http://nixos.org/nix/>

- ▶ **a functional package manager**
- ▶ **functional, again?**

so what's Nix?

<http://nixos.org/nix/>

- ▶ a **functional package manager**
- ▶ **functional, again?** but the one i use **works great** too!

so what's Nix?

<http://nixos.org/nix/>

- ▶ a **functional package manager**
- ▶ **functional, again?** but the one i use **works great** too!
- ▶ of course it does! more on this later...

and NixOS?

<http://nixos.org/>

- ▶ a free GNU/Linux distro (MIT/X11), est. 2006
- ▶ i686, x86_64, armv5tel
- ▶ \approx 8000 packages, \approx 35 regular contributors (yeah!)
- ▶ transparent binary/source deployment

bells, whistles, and more

- per-user package installation
- transactional upgrades & rollback
- system description & instantiation

the mechanics

- build environments
- building packages
- putting it another way

from Nix to Guix

- rationale
- using it
- a GNU distro?

bells, whistles, and more

per-user package installation

transactional upgrades & rollback
system description & instantiation

the mechanics

build environments

building packages

putting it another way

from Nix to Guix

rationale

using it

a GNU distro?

per-user, unprivileged package installation

```
alice@foo$ nix-env --install gcc-4.5 icecat-3.6
```


per-user, unprivileged package installation

```
alice@foo$ nix-env --install gcc-4.5 icecat-3.6
```

```
bob@foo$ nix-env --install gcc-4.3 icecat-3.7
```

per-user, unprivileged package installation

```
alice@foo$ nix-env --install gcc-4.5 icecat-3.6
alice@foo$ nix-store -q --requisites 'which icecat'
/nix/store/...-glibc-2.10
/nix/store/...-gtk+-2.16.6
/nix/store/...-alsa-lib-1.0.19
...

bob@foo$ nix-env --install gcc-4.3 icecat-3.7
```

per-user, unprivileged package installation

```
alice@foo$ nix-env --install gcc-4.5 icecat-3.6
alice@foo$ nix-store -q --requisites 'which icecat'
/nix/store/...-glibc-2.10
/nix/store/...-gtk+-2.16.6
/nix/store/...-alsa-lib-1.0.19
...
```

```
bob@foo$ nix-env --install gcc-4.3 icecat-3.7
bob@foo$ nix-store -q --requisites 'which icecat'
/nix/store/...-glibc-2.11.1
/nix/store/...-gtk+-2.18.6
/nix/store/...-alsa-lib-1.0.21a
...
```

transparent binary/source deployment

```
alice@foo$ nix-env --install gcc-4.5
installing 'gcc-4.5.3'
these paths will be fetched (20.00 MiB download):
  /nix/store/...-gcc-wrapper-4.5.3
  /nix/store/...-cloog-ppl-0.15.11
  /nix/store/...-gcc-4.5.3
```

transparent binary/source deployment

```
alice@foo$ nix-env --install gcc-4.5
```

```
installing 'gcc-4.5.3'
```

```
these derivations will be built:
```

```
  /nix/store/...-gcc-wrapper-4.5.3.drv
```

```
  /nix/store/...-gcc-4.5.3.drv
```

```
these paths will be fetched (30.00 MiB download):
```

```
  /nix/store/...-cloog-ppl-0.15.11
```

```
  /nix/store/...-gcc-4.5.3.tar.gz
```

bells, whistles, and more

- per-user package installation

- transactional upgrades & rollback**

- system description & instantiation

the mechanics

- build environments

- building packages

- putting it another way

from Nix to Guix

- rationale

- using it

- a GNU distro?

atomic & transactional upgrades

```
$ nix-env --upgrade '*'  
upgrading 'git-1.6.5' to 'git-1.7.1'  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
upgrading 'gnupg-2.0.12' to 'gnupg-2.0.15'  
upgrading 'gdb-7.0.1' to 'gdb-7.1'  
upgrading 'gnutls-2.8.5' to 'gnutls-2.10.0'  
upgrading 'openoffice.org-3.1.1' to 'openoffice.org-3.2.0'  
upgrading 'coccinelle-0.2.1' to 'coccinelle-0.2.2'  
...
```

atomic & transactional upgrades

```
$ nix-env --upgrade '*'  
upgrading 'git-1.6.5' to 'git-1.7.1'  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
upgrading 'gnupg-2.0.12' to 'gnupg-2.0.15'  
upgrading 'gdb-7.0.1' to 'gdb-7.1'  
upgrading 'gnutls-2.8.5' to 'gnutls-2.10.0'  
upgrading 'openoffice.org-3.1.1' to 'openoffice.org-3.2.0'  
upgrading 'coccinelle-0.2.1' to 'coccinelle-0.2.2'  
...
```

```
$ git --version ; gimp --version  
git version 1.7.1  
GNU Image Manipulation Program version 2.6.9
```



atomic & transactional upgrades

```
$ nix-env --upgrade '*'  
upgrading 'git-1.6.5' to 'git-1.7.1'  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
upgrading 'gnupg-2.0.12' to 'gnupg-2.0.15'  
upgrading 'gdb-7.0.1' to 'gdb-7.1'  
upgrading 'gnutls-2.8.5' to 'gnutls-2.10.0'  
upgrading 'openoffice.org-3.2.0' to 'openoffice.org-3.2.0'  
upgrading 'coccinelle-0.2.2' to 'coccinelle-0.2.2'  
...
```



atomic & transactional upgrades

```
$ nix-env --upgrade '*'  
upgrading 'git-1.6.5' to 'git-1.7.1'  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
upgrading 'gnupg-2.0.12' to 'gnupg-2.0.15'  
upgrading 'gdb-7.0.1' to 'gdb-7.1'  
upgrading 'gnutls-2.8.5' to 'gnutls-2.10.0'  
upgrading 'openoffice.org-3.1.1' to 'openoffice.org-3.2.0'  
upgrading 'coccinelle-0.2.1' to 'coccinelle-0.2.2'  
...
```

(interrupted right in the middle)

```
$ git --version ; gimp --version  
git version 1.6.5  
GNU Image Manipulation Program version 2.6.8
```

atomic & transactional upgrades

```
$ nix-env --upgrade '*'  
upgrading 'git-1.6.5' to 'git-1.7.1'  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
upgrading 'gnupg-2.0.12' to 'gnupg-2.0.15'  
upgrading 'gdb-7.0.1' to 'gdb-7.1'  
upgrading 'gnutls-2.8.5' to 'gnutls-2.10.0'  
upgrading 'openoffice.org-3.1.1' to 'openoffice.org-3.2.0'  
upgrading 'coccinelle-0.2.1' to 'coccinelle-0.2.2'  
...
```

(interrupted right in the middle)

```
$ git --version ; gimp --version  
git version 1.6.5  
GNU Image Manipulation Program version 2.6.8
```



per-user rollback



```
$ gimp --version
```

```
GNU Image Manipulation Program version 2.6.8
```

per-user rollback



```
$ gimp --version
```

```
GNU Image Manipulation Program version 2.6.8
```

```
$ nix-env --upgrade gimp
```

```
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'
```

```
...
```

per-user rollback



```
$ gimp --version  
GNU Image Manipulation Program version 2.6.8
```

```
$ nix-env --upgrade gimp  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
...
```

```
$ gimp --version  
Segmentation Fault
```

per-user rollback



```
$ gimp --version  
GNU Image Manipulation Program version 2.6.8
```

```
$ nix-env --upgrade gimp  
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'  
...
```

```
$ gimp --version  
Segmentation Fault
```

```
$ nix-env --rollback  
switching from generation 278 to 277
```

per-user rollback



```
$ gimp --version
```

```
GNU Image Manipulation Program version 2.6.8
```

```
$ nix-env --upgrade gimp
```

```
upgrading 'gimp-2.6.8' to 'gimp-2.6.9'
```

```
...
```

```
$ gimp --version
```

```
Segmentation Fault
```

```
$ nix-env --rollback
```

```
switching from generation 278 to 277
```

```
$ gimp --version
```

```
GNU Image Manipulation Program version 2.6.8
```


bells, whistles, and more

- per-user package installation

- transactional upgrades & rollback

- system description & instantiation**

the mechanics

- build environments

- building packages

- putting it another way

from Nix to Guix

- rationale

- using it

- a GNU distro?

system description

/etc/nixos/configuration.nix

```
{ pkgs, config, modulesPath, ... }:
```

```
{  
  boot = {  
    kernelPackages = pkgs.linuxPackages_2_6_31;  
    initrd.kernelModules = [ "uhci_hcd" "ata_piix" ];  
    kernelModules = [ "kvm-intel" "sdhci" "fuse" ];  
  
    loader.grub = {  
      device = "/dev/sda";  
      version = 2;  
    };  
  };  
};
```

system description

/etc/nixos/configuration.nix

```
fileSystems =
  [ { mountPoint = "/";
      fsType = "ext3";
      device = "/dev/sda1";
    }
    { mountPoint = "/home";
      fsType = "ext3";
      device = "/dev/sda3";
    }
  ];

swapDevices = [ device = "/dev/sda2"; ];
```

system description

/etc/nixos/configuration.nix

```
networking.hostName = "mylaptop";
```

```
security.extraSetuidPrograms =  
  [ "sudo" "xlaunch" "xscreensaver" "xlock" "wodim" ];
```

```
time.timeZone = "Europe/Paris";
```

```
users = {  
  extraUsers = [  
    { name = "ludo";  
      group = "users";  
      extraGroups = [ "audio" "cdrom" "video" ];  
    }  
  ];  
};
```

system description

/etc/nixos/configuration.nix

```
services = {
  lshd = {
    enable = true;
    rootLogin = true;
  };
  tor.enable = true;
  avahi.enable = true;

  xserver = {
    enable = true;
    videoDriver = "intel";
    driSupport = true;
    synaptics.enable = true;
  };
};
}
```

whole-system instantiation

```
$ sudo nixos-rebuild switch
```

```
...
```

whole-system instantiation

```
$ nixos-rebuild build-vm
```

```
...
```

whole-system instantiation

```
$ nixos-rebuild build-vm
```

...



whole-system instantiation

```
$ nixos-rebuild build-vm
```

```
...
```

Done. The virtual machine can be started by running `./result/bin/run-my-vm`.

whole-system instantiation

```
<<< NixOS Stage 2 >>>
```

```
running activation script...
setting up /etc...
updating groups...
updating users...
chmod: changing permissions of '/nix/store': Permission denied
starting Upstart...
[ 138.655783] loop: module loaded
[ 138.936756] processor LNXCPU:00: registered as cooling_device0
[ 139.440191] kua: no hardware support
[ 145.577789] sdhci: Secure Digital Host Controller Interface driver
[ 145.581322] sdhci: Copyright(c) Pierre Ossman
[ 147.764600] fuse init (API version 7.13)
[ 152.056203] udev: starting version 154
[ 163.352584] sr 1:0:0:0: Attached scsi generic sg0 type 5
[ 166.209818] cirrusfb 0000:00:02.0: BAR 0: can't reserve mem region [0xf0000000-0xf1ffffff]
[ 166.214345] cirrusfb 0000:00:02.0: cannot reserve region 0xf0000000, abort
[ 166.312222] cirrusfb: probe of 0000:00:02.0 failed with error -16
[ 166.595950] input: PC Speaker as /devices/platform/pcspkr/input/input2
[ 166.721137] piix4_smbus 0000:00:01.3: SMBus Host Controller at 0xb100, revision 0
[ 169.037742] input: Power Button as /devices/LNXSYSTM:00/LNXPWRBN:00/input/input3
[ 169.047669] ACPI: Power Button [PWRF]
[ 185.296443] FDC 0 is a S82078B
[ 187.263327] parport_pc 00:05: reported by Plug and Play ACPI
[ 187.263327] parport0: PC-style at 0x378, irq 7 [PCSPK(...)]
[ 187.623937] rtc_cmos 00:01: rtc core: registered rtc_cmos as rtc0
[ 187.890654] ppdev: user-space parallel port driver
[ 188.045505] rtc0: alarms up to one day, 114 bytes nvram, hpet irqs
[ 190.517632] input: InExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/inpu
t4
```

```
<<< Welcome to NixOS (x86_64) - Kernel 2.6.32.14 (tty1) >>>
```

```
nixey login: Wooww! NixOS booted in a VM!
```

whole-system instantiation

```
$ sudo nixos-rebuild test
```

```
...
```




“activates” the configuration (restarts daemons, etc.)

whole-system instantiation

```
$ sudo nixos-rebuild switch
```

```
...
```

activates the configuration & makes it the **boot default**



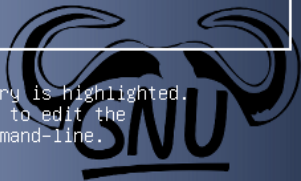
whole-system instantiation

GNU GRUB version 1.97.2

NixOS - Default

```
NixOS - Configuration 37 (2010-06-17 18:53:50 - 2.6.32.14)
NixOS - Configuration 36 (2010-05-20 23:41:03 - 2.6.32.13)
NixOS - Configuration 35 (2010-05-06 10:05:20 - 2.6.32.12)
NixOS - Configuration 34 (2010-05-05 21:18:01 - 2.6.32.12)
NixOS - Configuration 33 (2010-05-04 19:02:43 - 2.6.32.12)
NixOS - Configuration 32 (2010-02-22 14:12:49 - 2.6.32.8)
NixOS - Configuration 31 (2010-02-18 19:05:34 - 2.6.32.8)
NixOS - Configuration 30 (2010-02-13 19:17:49 - 2.6.32.8)
NixOS - Configuration 29 (2010-02-13 18:24:13 - 2.6.29.6)
NixOS - Configuration 28 (2010-02-12 22:37:06 - 2.6.32.8)
NixOS - Configuration 27 (2010-02-10 13:01:35 - 2.6.32.7)
NixOS - Configuration 26 (2010-02-04 23:00:19 - 2.6.32.7)
```

Use the `↑` and `↓` keys to select which entry is highlighted.
Press `enter` to boot the selected OS, `'e'` to edit the
commands before booting or `'c'` for a command-line.



system-wide rollback

```
$ nixos-rebuild switch --rollback
```

```
...
```

system-wide rollback

```
$ nixos-rebuild switch --rollback
```

```
...
```

... and voilà.

so you're already convinced...

so you're already convinced...

Yes!

tell me more!

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics

build environments

building packages

putting it another way

from Nix to Guix

rationale

using it

a GNU distro?

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics

build environments

building packages
putting it another way

from Nix to Guix

rationale
using it
a GNU distro?

build environments & reproducibility

- ▶ **versions** of the dependencies
- ▶ **compiler**
- ▶ **compilation options**, and those of dependencies
- ▶ **miscellaneous** (locale, timezone, etc.)
- ▶ **paths**

build environments & reproducibility

- ▶ **versions** of the dependencies
- ▶ **compiler**
- ▶ **compilation options**, and those of dependencies
- ▶ **miscellaneous** (locale, timezone, etc.)
- ▶ **paths**

`-I/path/to/headers`

`$CPATH`

`-L/path/to/lib`

`$LIBRARY_PATH`

build environments & reproducibility

- ▶ **versions** of the dependencies
- ▶ **compiler**
- ▶ **compilation options**, and those of dependencies
- ▶ **miscellaneous** (locale, timezone, etc.)
- ▶ **paths**

`-I/path/to/headers`

`$CPATH`

`-L/path/to/lib`

`$LIBRARY_PATH`

`$LD_LIBRARY_PATH`

`RPATH`

`RUNPATH`

build environments & reproducibility

- ▶ **versions** of the dependencies
- ▶ **compiler**
- ▶ **compilation options**, and those of dependencies
- ▶ **miscellaneous** (locale, timezone, etc.)
- ▶ **paths**

`-I/path/to/headers`

`$CPATH`

`-L/path/to/lib`

`$LIBRARY_PATH`

`$LD_LIBRARY_PATH`

`RPATH`

`RUNPATH`

`$PYTHONPATH`

`$CLASSPATH`

`$XML_CATALOG_FILES`

`$PERL5LIB`

`$GUILLE_LOAD_PATH`

how Nix controls the build environment

how Nix controls the build environment

1. one directory per installed package

how Nix controls the build environment

1. one directory per installed package
2. immutable installation directories

how Nix controls the build environment

1. one directory per installed package
2. immutable installation directories
3. undeclared dependencies invisible to the build process (POLA)

how Nix controls the build environment

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

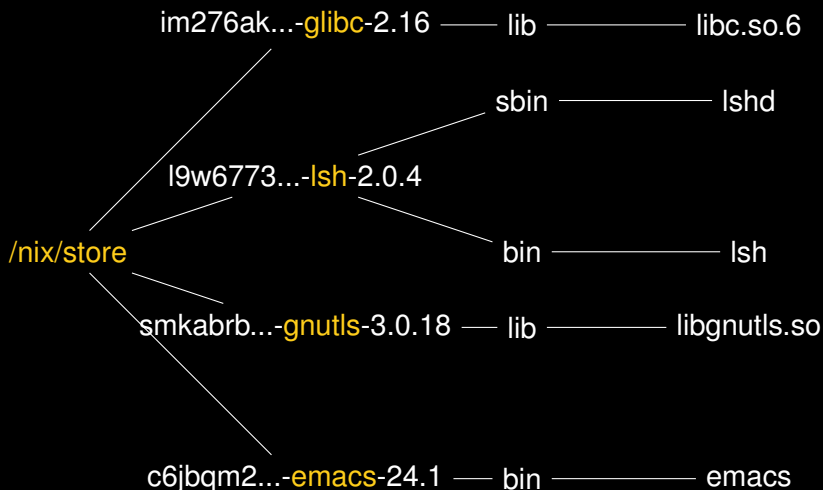
bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics

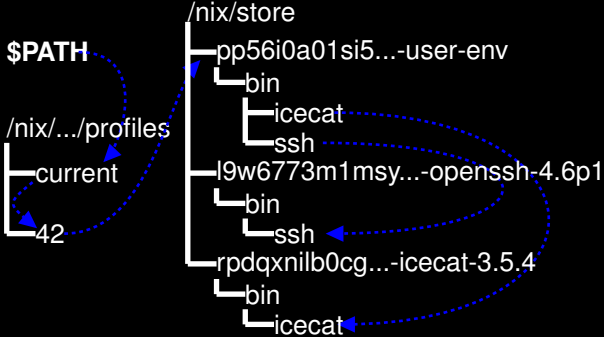
build environments
building packages
putting it another way

from Nix to Guix
rationale
using it
a GNU distro?

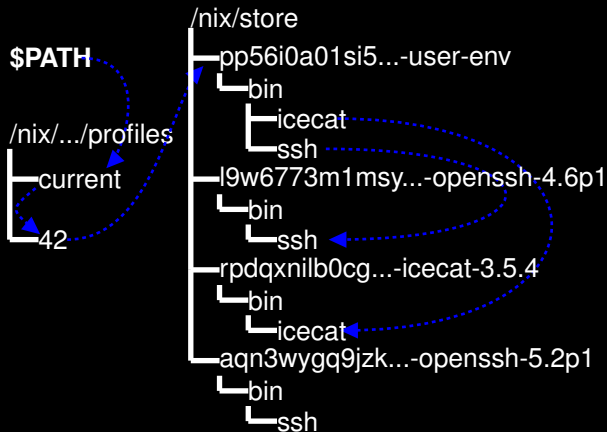
the store



user environments

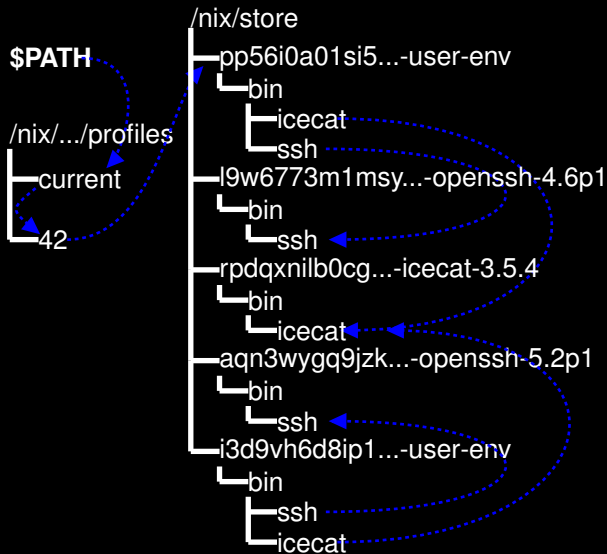


user environments



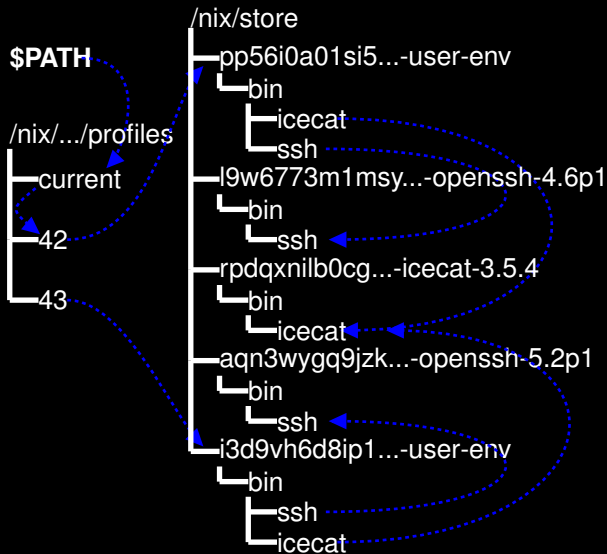
```
nix-env --upgrade openssh
```

user environments



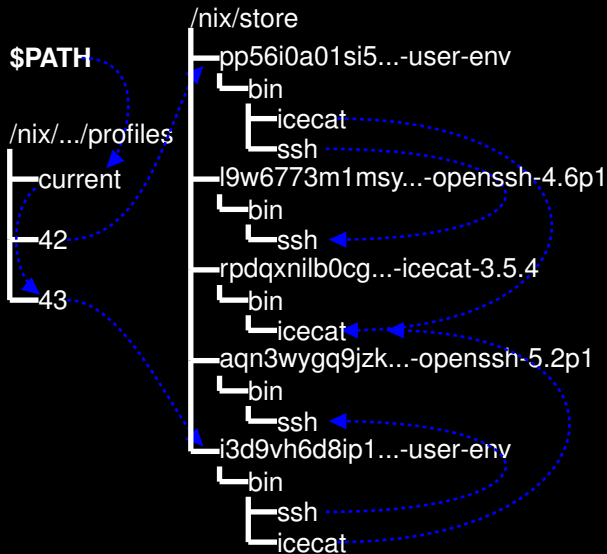
`nix-env --upgrade openssh`

user environments



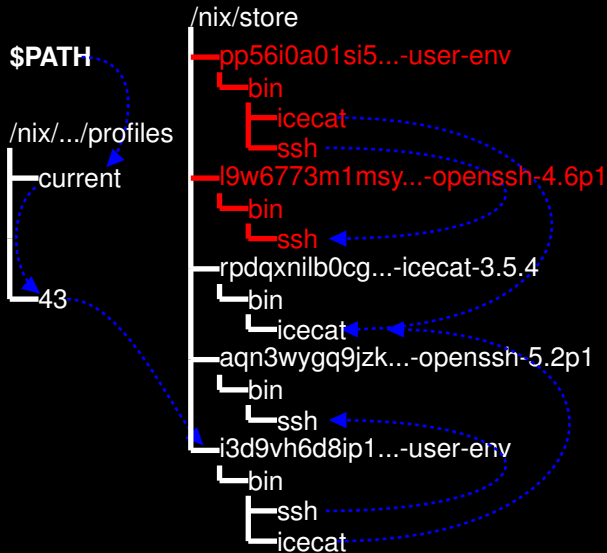
`nix-env --upgrade openssh`

user environments



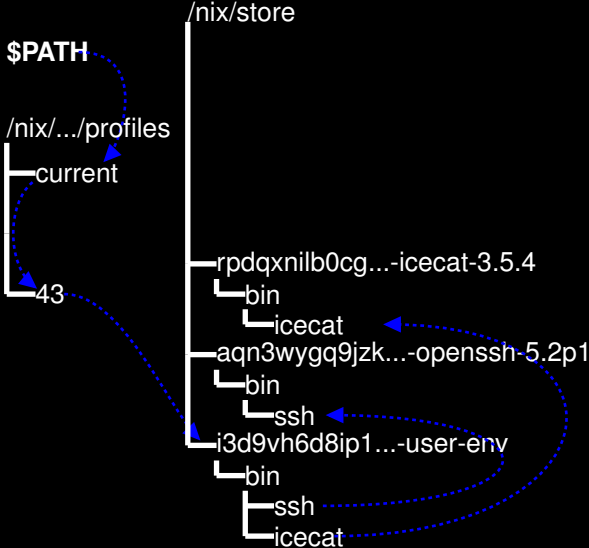
```
nix-env --upgrade openssh
```

user environments



`nix-env --remove-generations old`

user environments



`nix-collect-garbage`

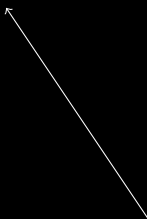
store paths

```
$ nix-build -A guile
```

store paths

```
$ nix-build -A guile  
/nix/store/ h2g4sc09h4... -guile-2.0.6
```

hash of *all* the dependencies



store paths

```
$ nix-build -A guile  
/nix/store/h2g4sc09h4...-guile-2.0.6
```

```
$ nix-store -q --requisites 'which guile'  
/nix/store/4jl83jgzaac...-glibc-2.16  
/nix/store/iplay43cg58...-libunistring-0.9.3  
/nix/store/47p47v92cj9...-libffi-3.0.9  
/nix/store/drkwck2j965...-gmp-5.0.5  
...
```

store paths

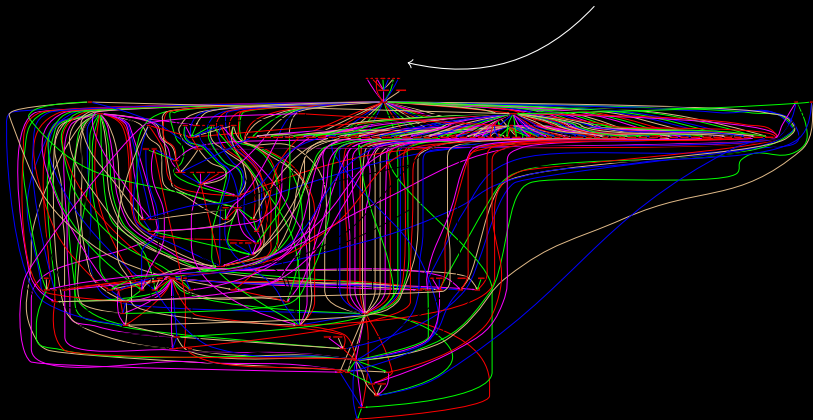
```
$ nix-build -A guile  
/nix/store/h2g4sc09h4...-guile-2.0.6
```

```
$ nix-store -q --requisites 'which guile'  
/nix/store/4jl83jgzaac...-glibc-2.16  
/nix/store/iplay43cg58...-libunistring-0.9.3  
/nix/store/47p47v92cj9...-libffi-3.0.9  
/nix/store/drkwck2j965...-gmp-5.0.5  
...
```

```
$ nix-copy-closure --to alice@example.com 'which guile'  
...
```

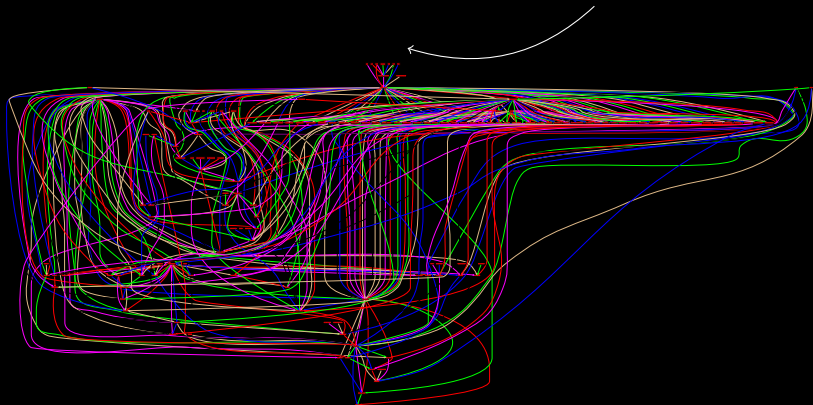
complete dependency specification

build-time dependencies of GNU Hello



complete dependency specification

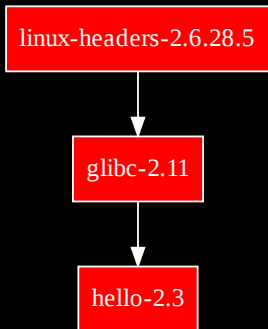
build-time dependencies of GNU Hello



... down to the compiler's compiler!

complete dependency specification

run-time dependencies of GNU Hello



run-time dependencies **inferred** by conservative scanning

packaging using the Nix language

```
{ fetchurl, stdenv } :  
stdenv . mkDerivation {  
  name = "hello-2.3";  
  src = fetchurl {  
    url = mirror://gnu/hello/hello-2.3.tar.bz2;  
    sha256 = "0c7vijq8y68...";  
  };  
  
  meta = {  
    description = "Produces a friendly greeting";  
    homepage = http://www.gnu.org/software/hello/;  
    license = "GPLv3+";  
  };  
}
```

function definition

formal parameters

function call

packaging using the Nix language

gcc, make, etc.

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

```
stdenv.mkDerivation {
```

```
  name = "hello-2.3";
```

```
  src = fetchurl {
```

```
    url = mirror://gnu/hello/hello-2.3.tar.bz2;
```

```
    sha256 = "0c7vijq8y68...";
```

```
  };
```

```
  buildInputs = [ gettext ];
```

dependency

```
  meta = {
```

```
    description = "Produces a friendly greeting";
```

```
    homepage = http://www.gnu.org/software/hello/;
```

```
    license = "GPLv3+";
```

```
  };
```

```
}
```

packaging using the Nix language

```
{ 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";
```

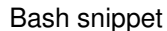
```
    homepage = http://www.gnu.org/software/hello/;
```

```
    license = "GPLv3+";
```

```
  };
```

```
}
```

Bash snippet



package composition with the Nix language

all-packages.nix

```
gettext = import ../development/libraries/gettext {  
  inherit fetchurl stdenv libiconv;  
};
```

...

```
hello = import ../applications/misc/hello {  
  inherit fetchurl stdenv;  
};
```

actual parameters

function call

The “Corresponding Source” for a work in object code form means **all the source code needed to generate**, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities.

The “Corresponding Source” for
a work in object code form means
all the **is needed to**
gene executable work, ex-
and to modify the work, including code
scripts to control those activities.

Nix makes sure users get the
Corresponding Source

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics

build environments
building packages
putting it another way

from Nix to Guix
rationale
using it
a GNU distro?

Nix implements a *functional* software deployment model.

Nix implements a *functional* software deployment model.

- ▶ **immutable** software installations

Nix implements a *functional* software deployment model.

- ▶ **immutable** software installations
- ▶ builds/installs have **no side effects**

Nix implements a *functional* software deployment model.

- ▶ **immutable** software installations
- ▶ builds/installs have **no side effects**
- ▶ build & deployment \equiv calling the build function
- ▶ Nix store \equiv **cache** of function call results

Nix implements a *functional* software deployment model.

- ▶ **immutable** software installations
- ▶ builds/installs have **no side effects**
- ▶ build & deployment \equiv calling the build function
- ▶ Nix store \equiv **cache** of function call results
- ▶ garbage collection...

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics
build environments
building packages
putting it another way

from Nix to Guix
rationale
using it
a GNU distro?

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics
build environments
building packages
putting it another way

from Nix to Guix

rationale

using it

a GNU distro?

so what's the point of Guix?

keeping Nix's **build & deployment model**

so what's the point of Guix?

keeping Nix's **build & deployment model**

using **Scheme** as the packaging language

so what's the point of Guix?

keeping Nix's **build & deployment model**

using **Scheme** as the packaging language

adding **GNU hackers** to the mix

why Guile Scheme instead of the Nix language?

- ▶ because it rocks!

why Guile Scheme instead of the Nix language?

- ▶ because it rocks!
- ▶ because it's GNU!

why Guile Scheme instead of the Nix language?

- ▶ because it rocks!
- ▶ because it's GNU!
- ▶ it has a compiler, Unicode, gettext, libraries, etc.

why Guile Scheme instead of the Nix language?

- ▶ because it rocks!
- ▶ because it's GNU!
- ▶ it has a compiler, Unicode, gettext, libraries, etc.
- ▶ it supports **embedded DSLs** via macros

why Guile Scheme instead of the Nix language?

- ▶ because it rocks!
- ▶ because it's GNU!
- ▶ it has a compiler, Unicode, gettext, libraries, etc.
- ▶ it supports **embedded DSLs** via macros
- ▶ can be used both for composition *and* build scripts

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics
build environments
building packages
putting it another way

from Nix to Guix

rationale

using it

a GNU distro?

Guix's declarative packaging layer

```
(define-public hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
              (method http-fetch)
              (uri (string-append
                    "http://ftp.gnu.org/.../hello-" version
                    ".tar.gz"))
              (sha256 (base32 "0wqd...dz6")))))
  (build-system gnu-build-system)
  (arguments '(:configure-flags '--disable-silent-rules)))
  (inputs '(("gawk" , gawk)))
  (description "GNU Hello")
  (long-description "GNUten Tag, Düsseldorf!")
  (home-page "http://www.gnu.org/software/hello/")
  (license "GPLv3+"))
```

Guix's declarative packaging layer

```
(define-public hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
              (method http-fetch)
              (uri (string-append
                    "http://ftp.gnu.org/.../hello-" version
                    ".tar.gz"))
              (sha256 (base32 "0wqd...dz6")))))
  (build-system gnu-build-system)
  (arguments '(:configure-flags '--disable-silent-rules)))
  (inputs '(("gawk" , gawk))) ← dependencies
  (description "GNU Hello")
  (long-description "GNUten Tag, Düsseldorf!")
  (home-page "http://www.gnu.org/software/hello/")
  (license "GPLv3+"))
```

Guix's declarative packaging layer

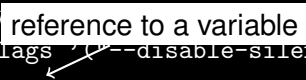
```
(define-public hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
              (method http-fetch)
              (uri (string-append
                    "http://ftp.gnu.org/.../hello-" version
                    ".tar.gz"))
              (sha256 (base32 "0wqd...dz6")))))
  (build-system gnu-build-system)
  (arguments '(:configure-flags ("--disable-silent-rules")))
  (inputs '(("gawk" ,gawk)))
  (description "GNU Hello")
  (long-description "GNUten Tag, Düsseldorf!")
  (home-page "http://www.gnu.org/software/hello/")
  (license "GPLv3+"))
```

reference to a variable

dependencies

Guix's declarative packaging layer

```
(define-public hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
              (method http-fetch)
              (uri (string-append
                    "http://ftp.gnu.org/.../hello-" version
                    ".tar.gz"))
              (sha256 (base32 "0wqd...dz6"))))
    (build-system gnu-build-system reference to a variable)
    (arguments '(:configure-flags '("--disable-silent-rules")))
    (inputs '(("gawk" , my-other-awk)))
    (description "GNU Hello")
    (long-description "GNUten Tag, Düsseldorf!")
    (home-page "http://www.gnu.org/software/hello/")
    (license "GPLv3+")))
```



Guix's declarative packaging layer

```
(define-public hello
  (package
    (name "hello")
    (version "1.0")
    (source (origin
      (method http-fetch)
      (uri (string-append
          "http://ftp.gnu.org/.../hello-" version
          ".tar.gz"))
      (sha256 (base32 "0wqd...dz6"))))
    (build-system gnu-build-system)
    (arguments '(:configure-flags '--disable-silent-rules))
    (inputs '("gawk" , gawk))
    (description "GNU Hello")
    (long-description "GNUten Tag, Düsseldorf!")
    (home-page "http://www.gnu.org/software/hello/")
    (license "GPLv3+")))
```

Guix's declarative packaging layer

```
(define-public hello
  (package
    (name "hello")
    (version "1.0")
    (source (origin
      (method http-fetch)
      (uri (string-append "http://ftp.gnu.org/.../hello-" version
        ".tar.gz"))
      (sha256 (base32 "0wq...dz6")))))
    (build-system gnu-build-system)
    (arguments '(:configure-flags '--disable-silent-rules))
    (inputs '("gawk" , gawk))
    (description "GNU Hello")
    (long-description "GNUten Tag, Düsseldorf!")
    (home-page "http://www.gnu.org/software/hello/")
    (license "GPLv3+")))
```

./configure && make install...

depends on gcc, make, bash, etc.

customized package declaration

```
(define-public gawk
  (package
    (name "gawk")
    (version "4.0.0")
    (source (origin (method http-fetch)
                    (uri "http://ftp.gnu.org/...")
                    (sha256 (base32 "0sss..."))))
    (build-system gnu-build-system)
    (arguments
      (case-lambda
        (( system )
         ; native builds
         (if (string=? system "i686-cygwin")
             '(:tests? #f) ; work around test failure
             '(:parallel-tests? #f))) ; seq. test suite
        ((system cross-system)
         ; cross builds
         (arguments cross-system)))) ; same as above
    (inputs '(("libsigsegv" ,libsigsegv)))
    (home-page "http://www.gnu.org/software/gawk/")
    (description "GNU Awk")))
```

customized package declaration

```
(define-public gawk
  (package
    (name "gawk")
    (version "4.0.0")
    (source (origin (method http-fetch)
                    (uri "http://ftp.gnu.org/...")
                    (sha256 (base32 "0sss...")))))
  (build-system gnu-build-system)
  (arguments
    (case-lambda
      ((system) ; native builds
       (if (string=? system "i686-cygwin")
           '(:tests? #f) ; work around test failure
           '(:parallel-tests? #f))) ; seq. test suite
      ((system cross-system) ; cross builds
       (arguments cross-system)))) ; same as above
  (inputs '(("libsigsegv" ,libsigsegv)))
  (home-page "http://www.gnu.org/software/gawk/")
  (description "GNU Awk"))
```

build options based on target

customized package declaration

```
(define-public guile-1.8
  (package ...
    (arguments
      '(:configure-flags '("--disable-error-on-warning"))

      ))

  (inputs '(
    ("gawk" ,gawk)
    ("readline" ,readline)))
```

customized package declaration

```
(define-public guile-1.8
  (package ...
    (arguments
      '(:configure-flags '("--disable-error-on-warning")
        #:patches (list (assoc-ref %build-inputs "patch/snarf")))
      ))
  (inputs '(("patch/snarf" "distro/guile-1.8.patch")
            ("gawk" ,gawk)
            ("readline" ,readline))))
```

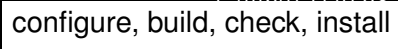
customized package declaration

```
(define-public guile-1.8
  (package ...
    (arguments
      '(:configure-flags '("--disable-error-on-warning")
        #:patches (list (assoc-ref %build-inputs "patch/snarf")))

      #:phases
      (alist-cons-before 'configure 'patch-search-path
        (lambda* (#:key outputs #:allow-other-keys)
          (substitute* "libguile/dynl.c"
            (re #~{.*$} match)
            (format #f
              " ~a~% lt_dladdsearchdir(\"~a/lib\");~%"
              match (assoc-ref outputs "out")))))
        %standard-phases )))

  (inputs '(("patch/snarf" "distro/guile-1.8.patch")
            ("gawk" ,gawk)
            ("readline" ,readline)))
```

configure, build, check, install



customized package declaration

```
(define-public guile-1.8
  (package ...
    (arguments
      '(:configure-flags '("--disable-error-on-warning")
        #:patches (list (assoc-ref %build-inputs "patch/snarf")))

      #:phases
      (alist-cons-before 'configure 'patch-search-path
        (lambda* (#:key outputs #:allow-other-keys)
          (substitute* "libguile/dynl.c"
            (repl-regexp "lt_dldsearchdir.*$" match)
            (format #f
              " ~a~% lt_dldsearchdir(\"~a/lib\");~%"
              match (assoc-ref outputs "out")))))
          %standard-phases )))
    (inputs '(("patch/snarf" "distro/guile-1.8.patch")
              ("gawk" ,gawk)
              ("readline" ,readline))))
```

add a phase before configure

configure, build, check, install

customized package declaration

```
(define-public guile-1.8
  (package ...
    (arguments
      '(:configure-flags '("--disable-error-on-warning")
        #:patches (list (assoc-ref %build-inputs "patch/snarf")))

      #:phases
      (alist-cons-before 'configure 'patch-search-path
        (lambda* (#:key outputs #:allow-other-keys)
          (substitute* "libguile/dynl.c"
            (("lt_dlinit.*$" match)
              (format #f
                "~a~% lt_dladdsearchdir(\"~a/lib\");~%"
                match (assoc-ref outputs "out")))))
          %standard-phases )))

    (inputs '(("patch/snarf" "distro/guile-1.8.patch")
              ("gawk" ,gawk)
              ("readline" ,readline))))
```

patch things up à la sed

building packages

```
(use-modules (guix packages) (guix store)
             (distro base))
```

```
(define store
  (open-connection) )
```

connect to the Nix build daemon



```
(package? hello)
=> #t
```

building packages

```
(use-modules (guix packages) (guix store)
             (distro base))
```

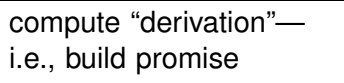
```
(define store
  (open-connection) )
```

```
(package? hello)
```

```
=> #t
```

```
(define drv (package-derivation store hello))
```

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



building packages

```
(use-modules (guix packages) (guix store)
             (distro 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)
             (distro 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))
```

... Nix daemon builds/downloads package on our behalf...

building packages

```
(use-modules (guix packages) (guix store)
             (distro 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))
```

```
... Nix daemon builds/downloads package on our behalf...
```

```
=> "/nix/store/pqr...-hello-2.8"
```

building packages

```
$ guix-build hello
```

building packages

```
$ 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
```


under the hood

```
(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)))
```

under the hood

connect to the build daemon



```
(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)))
```

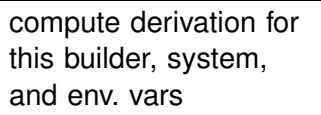
under the hood

build script, to be eval'd in chroot

```
(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)))
```


under the hood

```
(let* ((store (open-connection) )
      (builder '( begin
                  (mkdir %output)
                  compute derivation for this builder, system,
                  and env. vars 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)))
```



under the hood

```
(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)))
```



derivation primitive

```
(let* ((store (open-connection))
      (builder
        (add-text-to-store store "my-builder.sh"
                          "echo hello > \"$out\""
                          '())))
      (drv
        (derivation store "foo" "x86_64-linux"
                    "/bin/sh" '(,builder)
                    '(("HOME" . "/homeless")
                      ("PATH" . "/nothing:/here")))
        '((,builder))))
(build-derivations store (list drv))
```

status

- ▶ good API/language support for builds & composition
- ▶ expressive enough to build weird packages

status

- ▶ good API/language support for builds & composition
- ▶ expressive enough to build weird packages
- ▶ mini Guix-based distro!
- ▶ ... bootstrapped with Nixpkgs

tentative road map

- ▶ user environment builders + `guix-env` command
- ▶ Guix distro bootstrapped
- ▶ Guix support in Hydra
- ▶ distro supports whole-system configuration

tentative road map

- ▶ user environment builders + `guix-env` command
- ▶ Guix distro bootstrapped
- ▶ Guix support in Hydra
- ▶ distro supports whole-system configuration
- ▶ distro has a name

tentative road map

- ▶ user environment builders + `guix-env` command
- ▶ Guix distro bootstrapped
- ▶ Guix support in Hydra
- ▶ distro supports whole-system configuration
- ▶ distro has a name
- ▶ **you can help!**

bells, whistles, and more
per-user package installation
transactional upgrades & rollback
system description & instantiation

the mechanics
build environments
building packages
putting it another way

from Nix to Guix
rationale
using it
a GNU distro?

why would GNU need a distro?

- ▶ **direct connection** between GNU users & developers
 - ▶ direct **bug** stream
 - ▶ direct **release** stream

why would GNU need a distro?

- ▶ **direct connection** between GNU users & developers
 - ▶ direct **bug** stream
 - ▶ direct **release** stream
- ▶ **improved integration & cooperation**
 - ▶ GNU hackers **know how** to package their software
 - ▶ if GNU foo x.(y + 1) breaks GNU bar, **address that directly**

why would GNU need a distro?

- ▶ **direct connection** between GNU users & developers
 - ▶ direct **bug** stream
 - ▶ direct **release** stream
- ▶ **improved integration & cooperation**
 - ▶ GNU hackers **know how** to package their software
 - ▶ if GNU foo x.(y + 1) breaks GNU bar, **address that directly**
- ▶ following **free software distro guidelines**

why would GNU need a distro?

- ▶ **direct connection** between GNU users & developers
 - ▶ direct **bug** stream
 - ▶ direct **release** stream
- ▶ **improved integration & cooperation**
 - ▶ GNU hackers **know how** to package their software
 - ▶ if GNU foo x.(y + 1) breaks GNU bar, **address that directly**
- ▶ following **free software distro guidelines**
- ▶ **branding!**

why Guix-based?

- ▶ **technically superior** model & features
- ▶ **traceable** source-to-binary mapping
- ▶ extensible, i18n'd

why Guix-based?

- ▶ **technically superior** model & features
- ▶ **traceable** source-to-binary mapping
- ▶ extensible, i18n'd
- ▶ **Guile** is the official packaging language? :-)

summary

parentheses + weird paths

summary

**parentheses + weird paths
right, but more importantly...**

summary

- ▶ **features**

- ▶ per-user, unprivileged installation
- ▶ transactional upgrades; rollback
- ▶ full power of Guile to build & compose packages

- ▶ **foundations**

- ▶ purely functional package management
- ▶ traceable package source & dependencies
- ▶ completely bootstrapped

ludo@gnu.org

<http://gitorious.org/guix/>

Copyright © 2010, 2012 Ludovic Courtès ludo@gnu.org.

Picture of user environments is:

Copyright © 2009 Eelco Dolstra e.dolstra@tudelft.nl.

Copyright of other images included in this document is held by their respective owners.

This work is licensed under the [Creative Commons Attribution-Share Alike 3.0](http://creativecommons.org/licenses/by-sa/3.0/) License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

At your option, you may instead copy, distribute and/or modify this document under the terms of the [GNU Free](http://www.gnu.org/licenses/gfdl.html)

[Documentation License, Version 1.3 or any later version](http://www.gnu.org/licenses/gfdl.html) published by the Free Software Foundation; with no

Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at

<http://www.gnu.org/licenses/gfdl.html>.