Building a whole distro on top of a minimalistic language

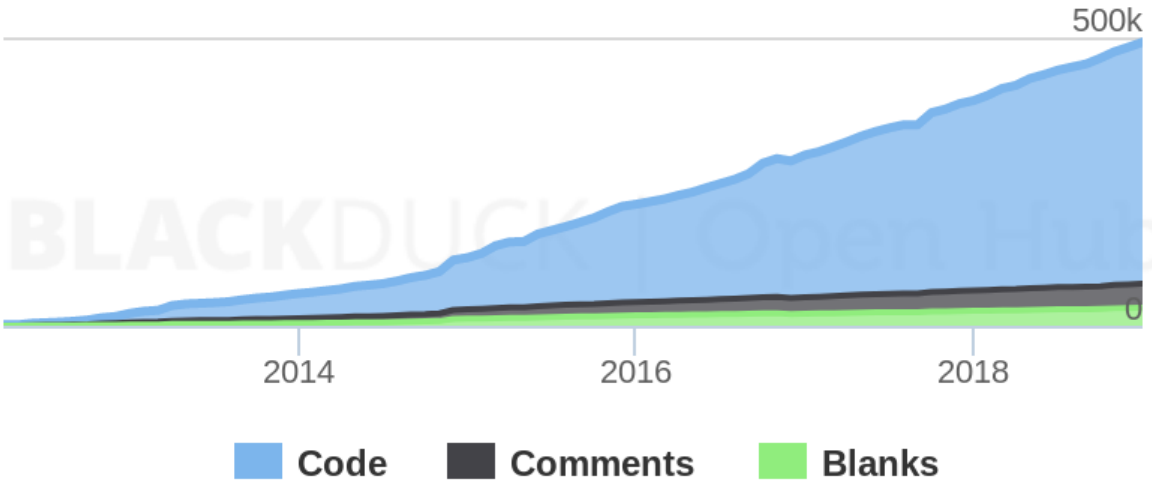
Ludovic Courtès

FOSDEM, 2 February 2019


the GNU/Linux potluck


Guix

Lines of Code



GNU Guix Reference Card

for version 0.16.1
https://gnu.org/software/guix/

Getting Started

To read the on-line documentation run `info guix`
or visit *https://gnu.org/s/guix/manual/en/html_node*. See
https://emacs-guix.gitlab.io/website/ for an Emacs interface to Guix.

Specifying Packages

Most commands take a "package specification" denoted *spec* in the sequel. Here are some examples:

`emacs`	Emacs package, latest version
`gcc-toolchain@7`	GCC toolchain, version 7.x
`gcc-toolchain:debug`	latest GCC toolchain, debugging symbols

Managing Packages

`guix package -s` *regexp* …	search for packages
`guix package --show=`*spec*	show package info
`guix package -i` *spec*…	install packages
`guix package -u` [*regexp*]	upgrade packages
`guix package -r` *name*…	remove packages
`guix package -m` *file*	instantiate from manifest
`guix package --roll-back`	roll back
`guix package -l`	list profile generations
`guix package --search-paths`	display search paths
`guix package -p` *profile* …	use a different profile

Manifests

`guix package -m` and other commands take a "manifest" file listing packages of interest, along these lines:

```
(specifications->manifest
 '("gcc-toolchain@7" "gcc-toolchain@7:debug"
   "openmpi"))
```

One-Off Environments

`guix environment --ad-hoc` *spec*…
: environment containing *spec*…

`guix environment python`
: environment to develop Python itself

`guix environment --ad-hoc python -C -- python3`
: run Python in a container

`guix environment -m` *file*
: create an environment for the packages in manifest *file*

Updating Guix

`guix describe`	describe current Guix
`guix pull`	update Guix
`guix pull -l`	view history
`guix pull --commit=`*commit*	update to *commit*
`guix pull --branch=`*branch*	update to *branch*
`guix pull -C` *file*	update the given channels

Channel Specifications

Channels specify Git repositories where `guix pull` looks for updates to Guix and external package repositories. By default `guix pull` reads `~/.config/guix/channels.scm`; with `-C` it can take channel specifications from a user-supplied file that looks like this:

```
(cons (channel
       (name 'guix-hpc)
       (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
       (branch "master"))
      %default-channels)
```

Managing Storage Space

`guix gc`	collect all garbage
`guix gc -C` *n*`G`	collect *n* GB of garbage
`guix gc -F` *n*`G`	ensure *n* GB are available
`guix package -d` *duration*	delete generations older than *duration*—e.g., `1m` for one month
`guix size` *spec* …	view package size
`guix gc -R /gnu/store/`*…*	list run-time dependencies
`guix graph -t references` *spec* …	view run-time dependencies

Customizing Packages

`guix` *command name* `--with-source=`*name*`=`*source*
: build *name* with a different source URL

`guix` *command spec* `--with-input=`*spec1*`=`*spec2*
: replace *spec1* with *spec2* in the dependency graph of *spec*

`guix` *command spec* `--with-graft=`*spec1*`=`*spec2*
: graft *spec2* in lieu of *spec1* in *spec*

`guix` *command spec* `--with-branch=`*branch*
: build *spec* from the given Git *branch*

`guix` *command spec* `--with-commit=`*commit*
: build *spec* from the given Git *commit*

Developing Packages

`guix edit` *spec*	view the definition
`guix build` *spec* …	build packages
`guix build --log-file` *spec*	view the build log
`guix build -K` *spec* …	build packages, keep build trees on failure
`guix build -S` *spec*	obtain the source of *spec*
`guix build --check` *spec*	rebuild a package
`guix build --target=`*triplet* …	cross-compile to *triplet*—e.g., `arm-linux-gnueabihf`
`guix download` *URL*	download from *URL* and print its SHA256 hash
`guix hash` *file*	print the hash of *file*
`guix graph` *spec* `| dot -Tpdf` …	view dependencies
`guix refresh` *spec*	update package definition
`guix import` *repo name*	import *name* from *repo*

Creating Application Bundles

`guix pack` *spec* …	create a tarball
`guix pack -f docker` *spec* …	create a Docker image
`guix pack -f squashfs` *spec* …	create a Singularity image
`guix pack --relocatable` *spec* …	create a relocatable tarball
`guix pack -S /bin=bin` *spec* …	make `/bin` a symlink to the packages' `bin` directory
`guix pack -m` *file*	bundle the packages from the manifest in *file*

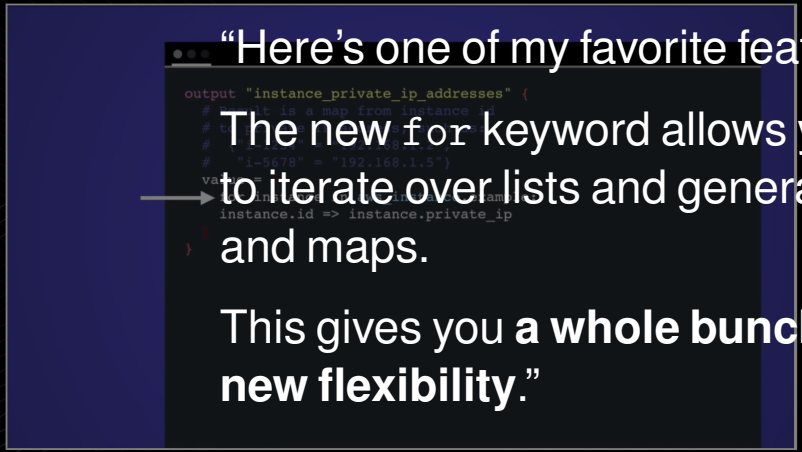
```
(operating-system
  (host-name "schememachine")
  (timezone "Europe/Brussels")
  (locale "fr_BE.utf8")
  (bootloader (bootloader-configuration
               (bootloader grub-efi-bootloader)
               (target "/boot/efi")))
  (file-systems (cons (file-system
                        (device (file-system-label "my-root"))
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (cons (user-account
                (name "charlie")
                (group "users")
                (home-directory "/home/charlie"))
               %base-user-accounts))
  (services (cons* (service dhcp-client-service-type)
                   (service openssh-service-type)
                   %base-services)))
```

```
$ guix system build config.scm
…

$ guix system vm config.scm
…

$ guix system container config.scm
…

$ guix system reconfigure config.scm
…
```

HashiCorp

```
output "instance_private_ip_addresses" {
  # Result is a map from instance id
  # to private IP address, such as:
  #  {"i-1234" = "192.168.1.2",
  #   "i-5678" = "192.168.1.5"}
  value = {
    for instance in aws_instance.example:
    instance.id => instance.private_ip
  }
}
```

HashiCorp



“Here’s one of my favorite features.

The new `for` keyword allows you to iterate over lists and generate lists and maps.

This gives you **a whole bunch of new flexibility**.”

HashiConf ‘18

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.

– Abelson & Sussman, 1987

```
(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 "An example GNU package")
    (description "Produce a friendly greeting.")
    (home-page "https://gnu.org/software/hello/")
    (license gpl3+)))

;; Yields: /gnu/store/...-hello-2.8
```

```
(service openssh-service-type)
```

```
(service openssh-service-type
         (openssh-configuration
           (x11-forwarding? #true)
           (permit-root-login 'without-password)))
```

```
;; Sets up cgit + nginx + fcgiwrap.
(service cgit-service-type
         (cgit-configuration
           (repository-directory "/srv/git")
           (project-list (list "foo" "bar"))))
```

```
(operating-system
  ;; …
  (services (remove (lambda (service)
                      (eq? ntp-service-type
                           (service-kind service)))
                    %desktop-services)))
```

```
(define %my-services
  ;; My very own list of services.
  (modify-services %desktop-services
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            (motd (plain-file "motd"
                                              "Howdy FOSDEM!"))))
    (upower-service-type config =>
                         (upower-configuration
                          (inherit config)
                          (ignore-lid? #true)
                          (percentage-critical 5.)))))
```

- Emacs and Web user interfaces
- `guix refresh` package auto-updater
- `guix lint` package checker
- `guix graph` dependency graph viewer
- `guix system extension-graph` service composition viewer
- ...

Unification
beyond the “distro”.

Nix & string interpolation

```
{  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

function call

Nix & string interpolation

```
{ fetchurl, stdenv }:

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

Bash snippet

NixOS / **nixpkgs**

Code | Issues 1,215 | Pull requests 317 | Projects 0 | Pulse

lib: Make escapeShellArg more robust

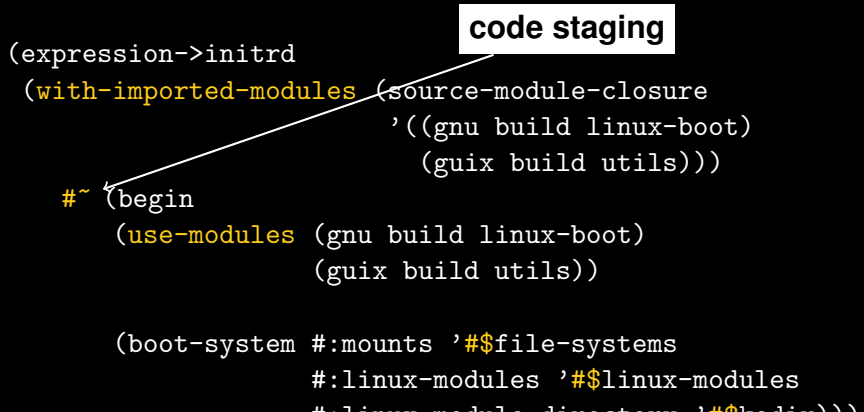
```
Quoting various characters that the shell *may* interpret specially is a
very fragile thing to do.

I've used something more robust all over the place in various Nix
expression I've written just because I didn't trust escapeShellArg.

Here is a proof of concept showing that I was indeed right in
distrusting escapeShellArg:
```

The Initial RAM Disk

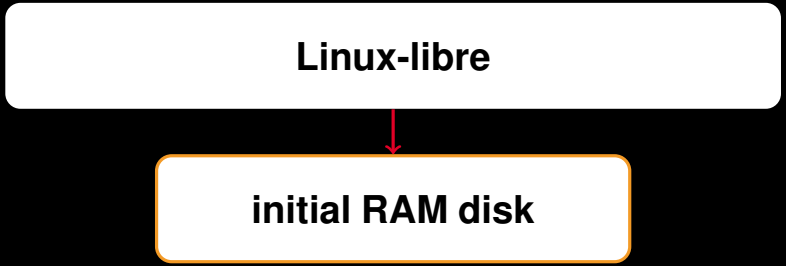
The Initial RAM Disk

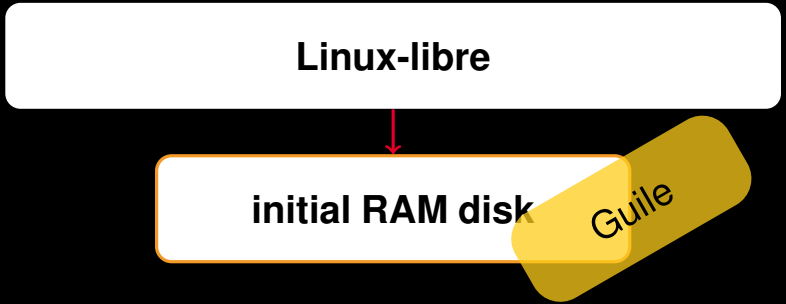
```
(expression->initrd
 (with-imported-modules (source-module-closure
                         '((gnu build linux-boot)
                           (guix build utils)))
   #~ (begin
        (use-modules (gnu build linux-boot)
                     (guix build utils))

        (boot-system #:mounts '#$file-systems
                     #:linux-modules '#$linux-modules
                     #:linux-module-directory '#$kodir)))
```

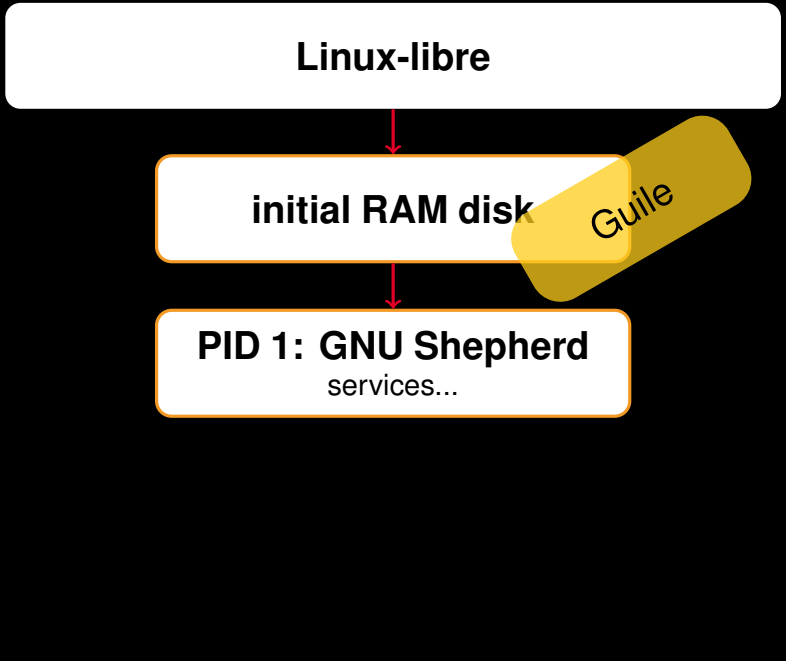
The Initial RAM Disk

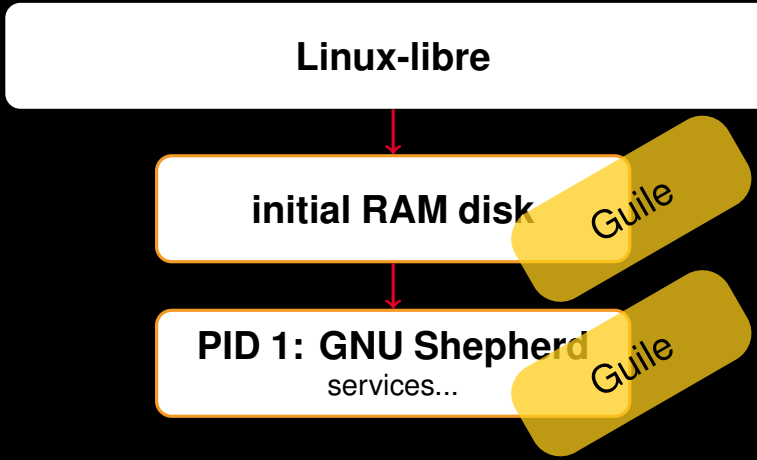


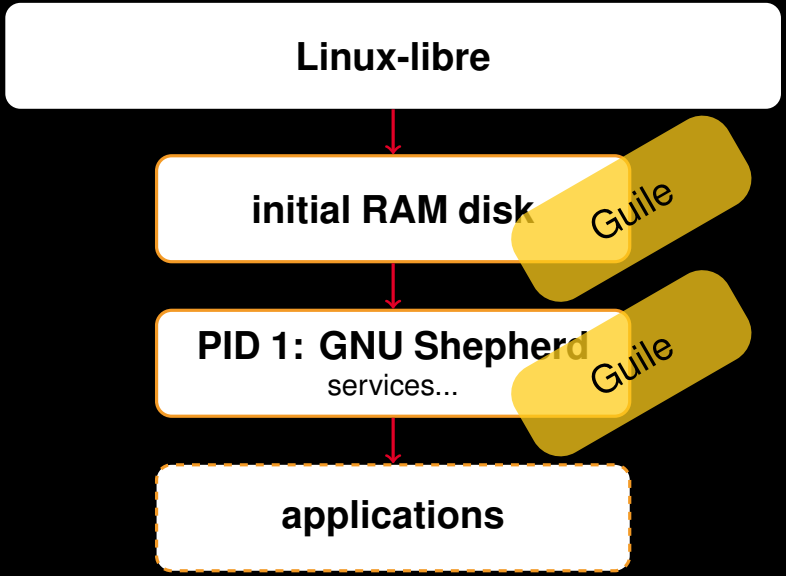
Linux-libre


Linux-libre
initial RAM disk


Linux-libre
initial RAM disk
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile
applications

System Services

```
;; Service definition for the GNU Shepherd (PID 1)
;; embedded in GuixSD.

(shepherd-service
  (provision '(mysql))
  (documentation "Run the MySQL server.")
  (start (let ((my.cnf (mysql-configuration-file config)))
           #~(make-forkexec-constructor
              (list (string-append #$mysql "/bin/mysqld")
                    (string-append "--defaults-file="
                                   #$my.cnf))
              #:user "mysql" #:group "mysql")))
  (stop #~(make-kill-destructor)))
```

```
;; Shepherd service to mount/unmount a file system.

(with-imported-modules '((gnu build file-systems))
  (shepherd-service
    (provision '(file-system-/home))
    (start #~(lambda ()
               (mount "/dev/foo" "/home" "ext4")))
    (stop #~(lambda ()
              (umount "/home")))))
```

```
;; Shepherd service for the BitlBee IRC gateway daemon.

  (shepherd-service
    (provision '(bitlbee))
    (requirement '(loopback))
    (start #~(make-forkexec-constructor
              (list #$(file-append bitlbee "/sbin/bitlbee")
                    ...)))
    (stop  #~(make-kill-destructor)))
```

```
;; Shepherd service for the BitlBee IRC gateway daemon.
(with-imported-modules '((gnu build linux-container))
  (shepherd-service
    (provision '(bitlbee))
    (requirement '(loopback))
    (start #~(make-forkexec-constructor/container
              (list #$(file-append bitlbee "/sbin/bitlbee")
                    ...)))
    (stop  #~(make-kill-destructor))))
```

Wrap-up.

- distro & tools as a **Scheme library**
- **hackability** through uniformity
- **code staging** techniques to glue it all

Join us now, share the parens!

- **install the distribution**
- **use it**, report bugs, add packages
- share your **ideas**!

Guix

ludo@gnu.org

https://gnu.org/software/guix/