Practical, verifiable software freedom with GuixSD

David Thompson

Sunday, March 25th, 2018

about me

GNU Guix contributor since 2013

GNU Guile user and contributor since 2012

day job: DevOps (AWS, Ruby)

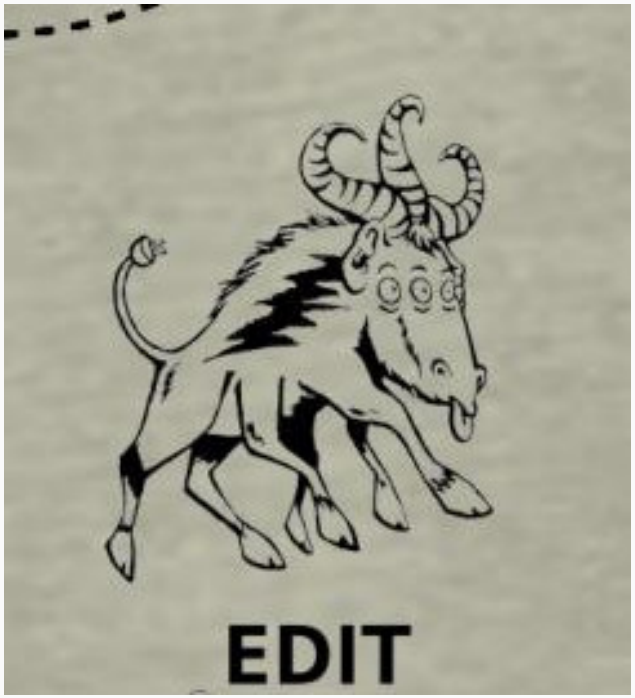
mastodon: `https://toot.cat/@dthompson`

blog: `https://dthompson.us`

the four freedoms

0: The freedom to run the program as you wish, for any purpose

the four freedoms



1: The freedom to study how the program works, and change it so it does your computing as you wish

the four freedoms

2: The freedom to redistribute copies so you can help your neighbor

the four freedoms

3: The freedom to distribute copies of your modified versions to others

the four freedoms

a wonderful set of rights, but often difficult to exercise in practice

common issues

figuring out how to view the exact source for a running program can be tricky

- source packages are good but are a bit arcane IMO

common issues

building from source is difficult or sometimes impossible

- dependency hell
- non-standard build system
- build scripts make assumptions that aren't true for your system
- need multiple package managers

common issues

sharing binaries can be tricky, too

- high barrier to entry for common package formats
- binary bundles are convenient, but problematic

common issues

major system upgrades can lead to sadness

ever upgrade your system, reboot, and find yourself in a completely broken state?

freedom: embedded

GuixSD removes many of the common barriers that prevent users from exercising their four freedoms

what is guixsd?

GuixSD is a fully-free GNU/Linux distribution with an advanced package manager and system upgrade mechanism

source code licensed under GPLv3 (shocker)

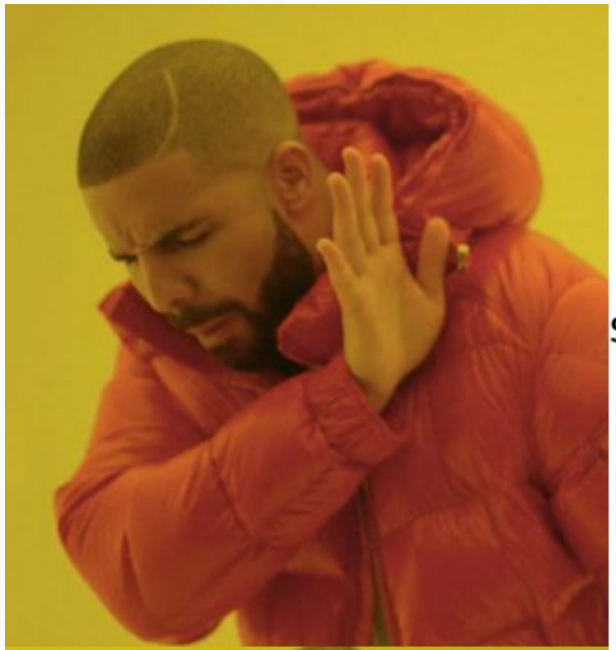
what is guix?

Guix is GuixSD's package manager (like apt, yum, pacman, etc.)

- unpriviliged package management
- per-user profiles
- atomic updates and rollbacks
- reproducible builds
- source-based with transparent binary downloads

unprivileged package management

users can build and install software without root privileges



sudo apt-get install emacs

guix package -i emacs

per-user profiles

each user may have one or more “profiles”, a union of many packages, without clobbering another user’s environment

use cases:

- Alyssa and Ben use different versions of Emacs
- Alyssa hacks on 2 Ruby projects that require different versions

transactional upgrades and rollbacks

experiment without fear!

```
guix package --upgrade emacs
```

oh no, the new version of Emacs is broken!


```
guix package --roll-back
```

a note about binaries

there is no central point of trust for receiving pre-built binaries (we call them substitutes)

Guix is a source-based package manager, but will transparently download pre-built binaries from a trusted third party, if available.

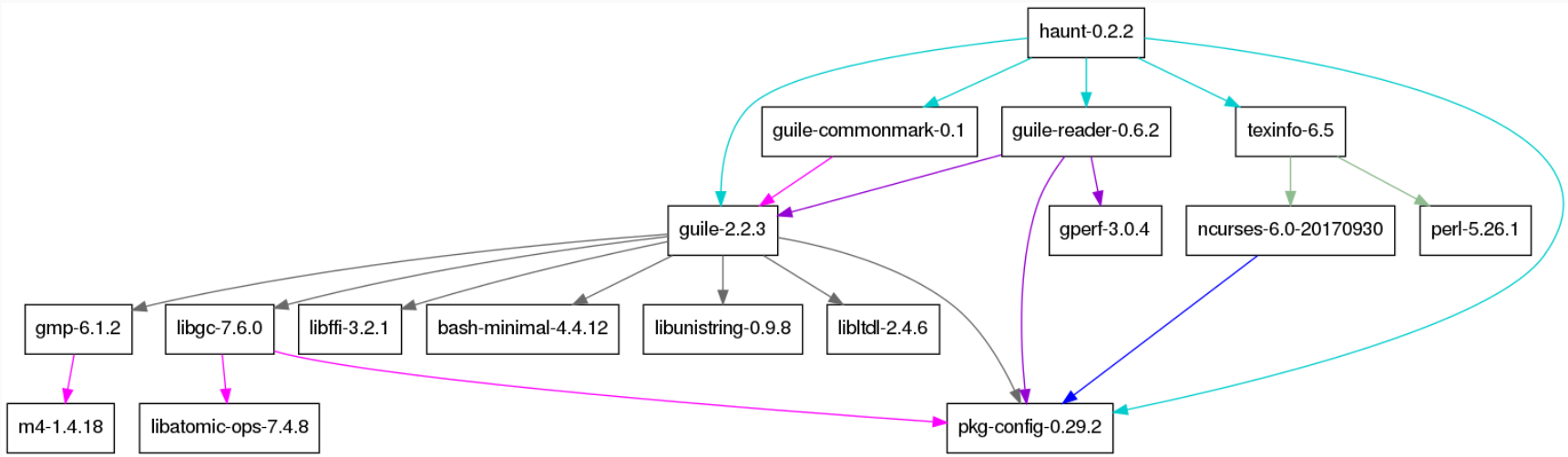
inspecting source code

quickly grab the source code for a package:

```
tar xf $(guix build --source gimp)
```

visualizing dependencies

```
guix graph haunt | dot -Tpng > graph.png
```



development environments

`guix environment` is like Python's virtualenv, Ruby's rvm, Node's nvm, etc. but for everything

quick example: play with a Ruby REPL without installing Ruby

```
guix environment --ad-hoc ruby -- irb
```

sharing development environments

```
(use-modules (guix profiles)
             (gnu packages base)
             (gnu packages guile))

(packages->manifest
 (list gnu-make
       guile-2.2
       guile-syntax-highlight
       haunt))
```

use it:

```
guix environment --manifest=guix.scm
```

containerized environments

experiment in an environment that is isolated from the rest of the system

example: a relatively constrained web browser

```
guix environment --ad-hoc icecat \
     --container \
     --network \
     --share=$HOME/.mozilla \
     --share=$HOME/Downloads \
     --expose=/tmp/.X11-unix

$ DISPLAY=:0.0 icecat
```

containerized environments (advanced)

```
# Create a Guix container that shares the host's network devices,
# GnuPG config, SSH config, and MySQL socket directory.  The container
# includes all of the software that is needed to build the gem set
# with Bundler.
guix environment --container --network \
     --share=$HOME/.gnupg --share=$HOME/.ssh --share=/run/mysqld --share=$HOME/Code \
     --ad-hoc ruby@2.2 mariadb imagemagick libxml2 libxslt gcc-toolchain@4.9 \
     gcc@4.9:lib make git coreutils openssh libffi pkg-config which sed gawk \
     openssl grep findutils procps nss-certs sqlite inetutils rsync gnupg \
     pinentry-tty
# Tweak the environment such that Ruby gems end up in the right place
# and their binaries can be found.
export GEM_HOME=$PWD/.gems
export PATH=$GEM_HOME/bin:$PATH
export LD_LIBRARY_PATH=$LIBRARY_PATH
export SSH_AUTH_SOCK=$HOME/.gnupg/S.gpg-agent.ssh
gpg-agent --daemon --enable-ssh-support --default-cache-ttl=10800 \
          --pinentry-program=$(which pinentry-tty)
# Create gem directory.
mkdir -p .gems
# Create /usr/bin/env so Ruby scripts work.
mkdir -p /usr/bin && ln -s $(which env) /usr/bin/env
# Bundle!
gem install bundler
bundle config build.nokogiri --use-system-libraries --with-xml2-include=$C_INCLUDE_PATH/libxml2
bundle
# Start the server!
rails server
```

system configurations

system configuration files fully describe the resulting operating system

since they are just text files, they can be easily backed up, stored in a version control system, and shared with other people

sharing system configurations

```
(operating-system
  (host-name "izanagi")
  (timezone "America/New_York")
  (locale "en_US.UTF-8")
  (bootloader (grub-configuration (target "/dev/sda")))
  (file-systems (cons (file-system
                        (device "root")
                        (title 'label)
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (list (user-account
                (name "dave")
                (comment "David Thompson")
                (group "users")
                (supplementary-groups '("wheel" "netdev" "audio" "video"
                                        "cdrom" "kvm" "input" "dialout")
                (home-directory "/home/dave"))))
  (packages (cons* arc-theme arc-icon-theme
                   htop less man-db ncurses nss-certs openssh unzip rsync
                   gnome-shell-extensions gnome-tweak-tool
                   %base-packages))
  (services (cons* (gnome-desktop-service)
                   %desktop-services))
  (name-service-switch %mdns-host-lookup-nss))
```

transactional upgrades and rollbacks redux

system upgrades are transactional, too!

```
sudo guix system reconfigure my-machine.scm
```

oh no, the latest GuixSD updates broke my system!

no worries, just reboot and select the previous, working version from the bootloader menu

sharing binaries

start a server to share your builds:

```
guix publish
```

have a friend download them:

```
guix build \
     --substitute-urls=http://guix.example.com:8080 \
     hello
```

host your own Guix LAN party!

(okay that sounds kinda boring)

reproducible builds

reproducible builds produce bit-identical binaries when performed multiple times under the same conditions.

when builds are reproducible, we gain the ability to detect when binaries are compromised

requires fixing issues in upstream build systems that are nondeterministic

reproducible builds

this is a cross-distro effort, but Guix was built to facilitate reproducibility from the beginning

see Chris Lamb's talk *"You think you're not a target? A tale of three developers..."* from yesterday for a deeper dive

https://reproducible-builds.org

reproducible builds

is this build reproducible on my machine?

```
guix build --rounds=3 python
```

challenge authority

is this build reproducible on many machines?

is this build compromised?

```
guix challenge emacs \
     --substitute-urls="https://mirror.hydra.gnu.org \
       https://bobs-questionable-binaries.biz"
```

reasons for mismatched binaries

innocent build nondeterminism:

- timestamps
- hardware-specific optimizations (looking at you, ATLAS)
- build directories
- bad parallelism

or maybe...

- malicious tampering

view the source

show me how Ruby is built:

```
export EDITOR=emacs
guix edit ruby
```

customize packages

build Ruby using different source code:

```
guix build ruby --with-source=ruby-2.5.0.tar.gz
```

customize packages in Guix itself

let's make some changes to the source code itself!

```
git clone https://git.savannah.gnu.org/git/guix.git
cd guix
guix environment guix
./configure
make
./pre-inst-env guix edit ruby
guix build ruby
```

now make a patch and send it to us!

sharing custom packages

```
(define-public openfst
  (let ((commit "58983d37849a24ad80cf908098e2af7c4863941d"))
    (package
      (name "openfst")
      (version (string-append "1.4.1-1." (string-take commit 7)))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://github.com/cobaltspeech/stable-openfst.git")
                      (commit commit)))
                (file-name (string-append name "-" version))
                (sha256
                 (base32
                  "0yikm03d82j6rpzqkg41yhs91lg4s9k03zhiqx7cndw9xqdsnbg1"))))
      (build-system gnu-build-system)
      (arguments
       '(#:configure-flags '("--with-pic"
                             "--enable-shared"
                             "--enable-static")))
      (synopsis "Finite-state transducer library")
      (description "OpenFst is a library for constructing, combining,
optimizing, and searching weighted finite-state
transducers (FSTs).")
      (home-page "https://github.com/cobaltspeech/stable-openfst")
      (license license:asl2.0))))
```

```
guix build --load-path=$HOME/my-packages openfst
```

interoperate with other systems

need a Docker image?

```
guix pack --format=docker guile emacs geiser
```

(tangent: see *Solving the deployment crisis with GNU Guix* from LibrePlanet 2016 for reasons why Docker may not be so great)

interoperate with other systems

or maybe you want something similar to snap or flatpak?

make a tarball bundle that anyone can extract on their GNU/Linux system:

```
guix pack guile emacs geiser
```

import foreign packages

or maybe you want assistance translating foreign packages into Guix packages:

```
guix import pypi flask
guix import gem pry
guix import elpa magit
```

and many more (CRAN, CPAN, Crate, etc.)

literally: embedded

fun fact: GuixSD now runs on the Beaglebone Black single-board computer!

```
(operating-system
  (bootloader (bootloader-configuration
               (bootloader u-boot-beaglebone-black-bootloader)
               (target "/dev/mmcblk1")))
  (initrd-modules (cons "omap_hsmmc" %base-initrd-modules))
  (services (cons* (dhcp-client-service)
                   (agetty-service
                    (agetty-configuration
                     (extra-options '("-L"))
                     (baud-rate "115200")
                     (term "vt100")
                     (tty "ttyO0")))
                   %base-services))
  ...)
```

hopefully more ARM systems coming soon!

extending guix

GuixSD is essentially a big Scheme library

with a little Scheme know-how its easy to write new packages, services, and tools that use the exact same APIs that the core Guix tools use

extending guix (silly example)

```
> (use-modules (guix packages) (gnu packages emacs))
> (for-each (lambda (name)
              (display (string-append "hey, " name "! You're an Emacs dependency!\n")))
            (sort (map car (package-inputs emacs)) string<))

hey, acl! You're an Emacs dependency!
hey, alsa-lib! You're an Emacs dependency!
hey, dbus! You're an Emacs dependency!
hey, giflib! You're an Emacs dependency!
hey, gnutls! You're an Emacs dependency!
hey, gtk+! You're an Emacs dependency!
hey, imagemagick! You're an Emacs dependency!
hey, libice! You're an Emacs dependency!
hey, libjpeg! You're an Emacs dependency!
hey, libotf! You're an Emacs dependency!
hey, libpng! You're an Emacs dependency!
hey, librsvg! You're an Emacs dependency!
hey, libsm! You're an Emacs dependency!
hey, libtiff! You're an Emacs dependency!
hey, libx11! You're an Emacs dependency!
hey, libxft! You're an Emacs dependency!
hey, libxml2! You're an Emacs dependency!
hey, libxpm! You're an Emacs dependency!
hey, m17n-lib! You're an Emacs dependency!
hey, ncurses! You're an Emacs dependency!
hey, zlib! You're an Emacs dependency!
```

the stack

core components written in Scheme:

- initial RAM disk
- init system (GNU Shepherd)
- package manager

lots of code reuse and opportunities for extension

challenges

- usabliity
- the npm problem
- self-hosting compilers
- cluster deployments

the freedom to contribute

The GNU Guix project has a welcoming community:

- code of conduct
- Outreachy
- Google Summer of Code
- oh, and no copyright assignment (in case you were wondering)

we need your help to bring GuixSD to a wider audience!

join us!

thanks!

docs, past talks, source code, mailing list/IRC info, etc.:

https://gnu.org/s/guix

credits

GNU run, edit, share, contribute images: `https://shop.fsf.org/tshirts-hoodies/4-gnus-4-freedoms-t-shirt`