Reproducible System Configuration with Nix and Guix

Aidan Hall - 27/08/2024

About Me

- Working in CE – GPU Software
- Studying Computer Science at Warwick

Nix

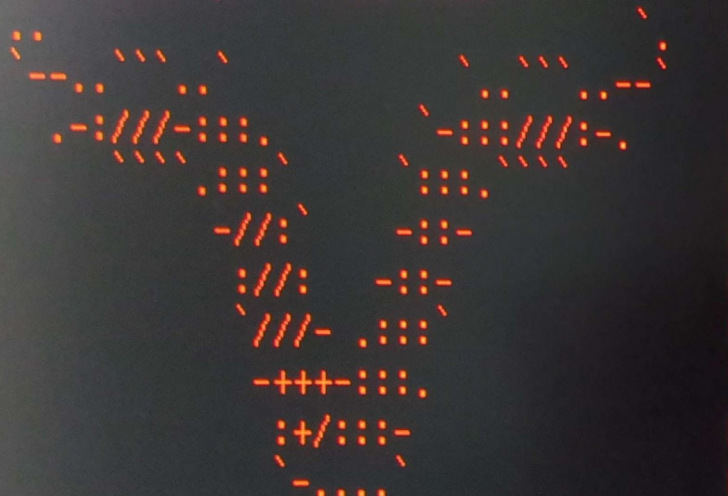
- Purely functional package manager
- Bespoke language
- No dependency hell

Guix

- Based on Nix
- A GNU project
- Scheme (Lisp)

```
This is the GNU system.  Welcome.
guix-pc login: aidan
Password:
This is the GNU operating system, welcome!

aidan@guix-pc ~$ neofetch
```



```
aidan@guix-pc
-------------
OS: Guix System x86_64
Kernel: 6.9.12
Uptime: 2 mins
Packages: 142 (guix-system), 1 (guix-user)
Shell: bash 5.1.16
Terminal: /dev/tty2
CPU: Intel i5-4670K (4) @ 3.800GHz
GPU: NVIDIA GeForce GTX 1060 3GB
Memory: 202MiB / 15935MiB
```

```
aidan@guix-pc ~$ _
```

Package Management

```
aidan@archlinux ~$ guix package -i tmux


The following package will be installed:
   tmux 3.4

aidan@archlinux ~$ which tmux
/home/aidan/.guix-profile/bin/tmux
```

Generations

```
aidan@archlinux ~$ guix package -l
Generation 1    Aug 22 2024 19:42:19
  glibc-locales 2.35

...

Generation 9    Aug 23 2024 12:50:14    (current)
 + tmux 3.4

aidan@archlinux ~$ guix package -S 1
switched from generation 9 to 1
aidan@archlinux ~$ tmux --version
bash: tmux: command not found
```

Temporary Environments

```
aidan@archlinux ~/programming/games$ clang --version
bash: clang: command not found
aidan@archlinux ~/programming/games$ guix shell clang-toolchain

aidan@archlinux ~/programming/games [env]$ clang --version
clang version 18.1.8
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /gnu/store/daql1mn64ndrql1gmv8cm6pm4s7nxgzd-profile/bin
aidan@archlinux ~/programming/games [env]$
exit
aidan@archlinux ~/programming/games$ clang --version

bash: clang: command not found
aidan@archlinux ~/programming/games$
```

Environment Manifest

```
aidan@archlinux ~/programming/games$ guix shell clang-toolchain cmake \
> --export-manifest | tee manifest.scm
;; What follows is a "manifest" equivalent to the command line you gave.
;; You can store it in a file that you may then pass to any 'guix' command
;; that accepts a '--manifest' (or '-m') option.

(specifications->manifest
  (list "clang-toolchain" "cmake"))
aidan@archlinux ~/programming/games$ guix shell -m manifest.scm
aidan@archlinux ~/programming/games [env]$ which cmake; which clang
/gnu/store/li90s400km688nn7cnq1myb0hc1xdl5n-profile/bin/cmake
/gnu/store/li90s400km688nn7cnq1myb0hc1xdl5n-profile/bin/clang
```

Home Environment

```
(home-environment
 (packages
  (specifications->packages
   (list "git" "emacs" "ripgrep")))

 (services
  (list
   (service home-syncthing-service-type)

   (service home-dotfiles-service-type
            (home-dotfiles-configuration
             (directories '("dotfiles")))))))
```

```
aidan@archlinux ~/dots$ guix home reconfigure home-configuration.scm
```

Operating System

```
(operating-system
 (locale "en_GB.utf8")
 (timezone "Europe/London")
 (keyboard-layout (keyboard-layout "gb"))
 (host-name "guix-pc")
 (kernel linux)
```

```
 (users (cons* (user-account
                (name "aidan")
                (comment "Aidan Hall")
                (group "users")
                (home-directory "/home/aidan")
                (supplementary-groups
                 '("wheel" "netdev" "audio" "video")))
               %base-user-accounts))
```

```
(packages (append
           (specifications->packages
            (list "emacs" "man-db" "gnupg" "firefox" "tmux"))
           %base-packages))
```

```
(services
 (append (list
          (service openssh-service-type)
          (service bluetooth-service-type)
          (service gnome-desktop-service-type)
          (service tlp-service-type))
         %desktop-services))
```

```
(bootloader (bootloader-configuration
             (bootloader grub-bootloader)
             (targets (list "/dev/sdb"))
             (keyboard-layout keyboard-layout)))
```

```
(file-systems (cons* (file-system
                      (mount-point "/")
                      (device (file-system-label "guix-root"))
                      (type "ext4")) %base-file-systems)))
```

```
aidan@guix-pc ~/dots $ sudo guix home reconfigure home-configuration.scm
```

Closing Remarks

- A new paradigm for software deployment
- One universal standard that covers everyone’s use cases

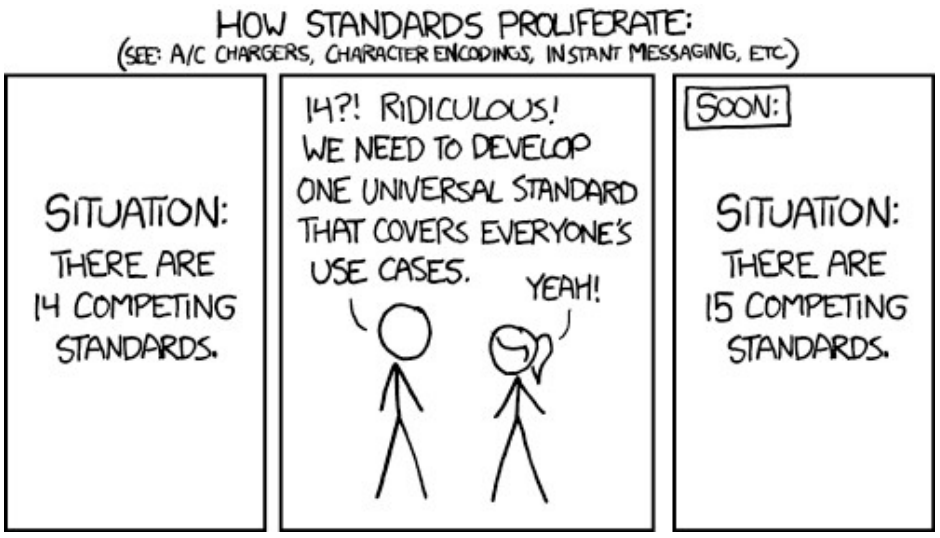


Image Sources

- https://github.com/NixOS/nixos-artwork/blob/master/logo/nixos.svg

  - Tim Cuthbertson

- http://git.savannah.gnu.org/cgit/guix/guix-artwork.git - Luis Felipe López Acevedo

- https://xkcd.com/927/

Any Questions?

Reproducible System Configuration with Nix and Guix

Aidan Hall - 27/08/2024

Nix

- Purely functional package manager
- Bespoke language
- No dependency hell

Purely functional: Package installations don’t have side-effects on other parts of the environment.

Nix expressions are used to specify the configuration options and dependencies for each package.

Every version of every package is handled in isolation: you can have multiple versions of the same package installed at the same time, if different packages depend on different ones.

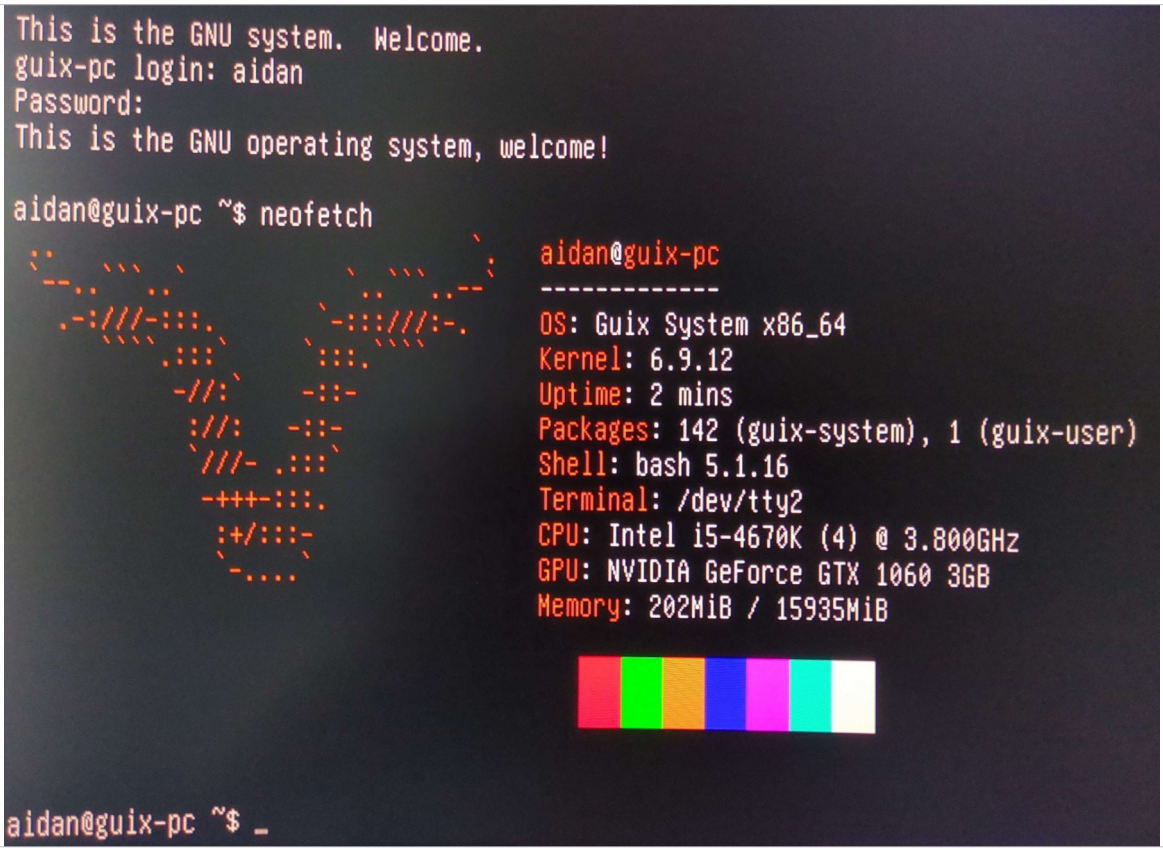
- Based on Nix
- A GNU project
- Scheme (Lisp)

A lot of directly correspondent functionality.

Focus on preserving the user’s freedom: all free software.

Not just configured, but mostly written in Scheme.

I’ll focus on Guix because it’s the one I’m more familiar with, but Nix can do pretty much everything you’ll see in this talk.



The package description languages aren’t limited to individual package management.

We can also use them to define reproducible configurations for everything from software development environments to the entire operating system!

Lets explore how they work in more detail.

Package Management

```
aidan@archlinux ~$ guix package -i tmux


The following package will be installed:
   tmux 3.4

aidan@archlinux ~$ which tmux
/home/aidan/.guix-profile/bin/tmux
```

This feels familiar.

Guix can be installed within another distribution, like flatpak.

I didn’t need to install as root: packages are installed in a per-user profile.

Generations

```
aidan@archlinux ~$ guix package -l
Generation 1    Aug 22 2024 19:42:19
  glibc-locales 2.35

...

Generation 9    Aug 23 2024 12:50:14    (current)
 + tmux 3.4

aidan@archlinux ~$ guix package -S 1
switched from generation 9 to 1
aidan@archlinux ~$ tmux --version
bash: tmux: command not found
```

Purely functional package management: updates are non-destructive.

Some changes do modify a persistent environment; to get around this, Guix uses generations to keep old versions of the environment around.

We can roll back to any previous version of the environment if we messed something up.

Temporary Environments

```
aidan@archlinux ~/programming/games$ clang --version
bash: clang: command not found
aidan@archlinux ~/programming/games$ guix shell clang-toolchain

aidan@archlinux ~/programming/games [env]$ clang --version
clang version 18.1.8
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /gnu/store/daql1mn64ndrql1gmv8cm6pm4s7nxgzd-profile/bin
aidan@archlinux ~/programming/games [env]$
exit
aidan@archlinux ~/programming/games$ clang --version

bash: clang: command not found
aidan@archlinux ~/programming/games$
```

We often have programs or sets of programs that we use together in certain contexts.

A familiar example would be the build system for a given project: every developer working on the project needs the same set of programs, on every machine they work on.

Arm already uses Docker containers for this, but we can achieve a similar effect much more simply with guix shell.

We use guix shell to set up a temporary environment with clang available.

When we exit the environment, it is no longer there. This does not permanently change the environment, so generations are unnecessary.

Environment Manifest

```
aidan@archlinux ~/programming/games$ guix shell clang-toolchain cmake \
> --export-manifest | tee manifest.scm
;; What follows is a "manifest" equivalent to the command line you gave.
;; You can store it in a file that you may then pass to any 'guix' command
;; that accepts a '--manifest' (or '-m') option.

(specifications->manifest
  (list "clang-toolchain" "cmake"))
aidan@archlinux ~/programming/games$ guix shell -m manifest.scm
aidan@archlinux ~/programming/games [env]$ which cmake; which clang
/gnu/store/li90s400km688nn7cnq1myb0hc1xdl5n-profile/bin/cmake
/gnu/store/li90s400km688nn7cnq1myb0hc1xdl5n-profile/bin/clang
```

If we want to use the same environment again, especially if it has a lot of packages in it, which can create a manifest file, which stores the list of packages.

This is our first bit of Scheme-based configuration. We could copy manifest.scm to another computer, and use it to get the same environment there.

Manifests do the same job as Dockerfiles, and you can actually generate Docker containers using them.

Home Environment

```
(home-environment
 (packages
  (specifications->packages
   (list "git" "emacs" "ripgrep")))

 (services
  (list
   (service home-syncthing-service-type)

   (service home-dotfiles-service-type
            (home-dotfiles-configuration
             (directories '("dotfiles")))))))
```

```
aidan@archlinux ~/dots$ guix home reconfigure home-configuration.scm
```

There are some packages that you'll always want available. You can add them to your home environment.

You can also add services to the home environment, which will start up automatically when you log in.
Here I've got Syncthing, a file synchronisation program.

You can also use Guix to manage your configuration files.
This is useful because Guix also tracks generations of your home configuration.
This could let you roll back a breaking change to your configuration.

Operating System

```
(operating-system
 (locale "en_GB.utf8")
 (timezone "Europe/London")
 (keyboard-layout (keyboard-layout "gb"))
 (host-name "guix-pc")
 (kernel linux)


 (users (cons* (user-account
                (name "aidan")
                (comment "Aidan Hall")
                (group "users")
                (home-directory "/home/aidan")
                (supplementary-groups
                 '("wheel" "netdev" "audio" "video")))
               %base-user-accounts))


(packages (append
           (specifications->packages
            (list "emacs" "man-db" "gnupg" "firefox" "tmux"))
           %base-packages))
```

```
(services
 (append (list
           (service openssh-service-type)
           (service bluetooth-service-type)
           (service gnome-desktop-service-type)
           (service tlp-service-type))
          %desktop-services))


(bootloader (bootloader-configuration
             (bootloader grub-bootloader)
             (targets (list "/dev/sdb"))
             (keyboard-layout keyboard-layout)))

(file-systems (cons* (file-system
                      (mount-point "/")
                      (device (file-system-label "guix-root"))
                      (type "ext4")) %base-file-systems)))
```

```
aidan@guix-pc ~/dots $ sudo guix home reconfigure home-configuration.scm
```

We can use Guix to specify a whole operating system configuration.

We can specify all aspects of the system this way, including file systems and the desktop environment.

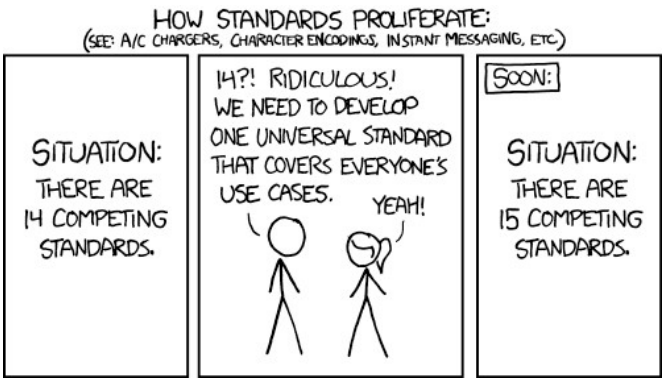
Guix also tracks generations of the operating system, and will create an entry in the GRUB bootloader for each one.

This allows you to recover the system, even if the latest kernel can’t be loaded.

You can also use these operating system manifests to create images that can run in a VM, and ISO files that you can flash to a USB stick.

Closing Remarks

- A new paradigm for software deployment
- One universal standard that covers everyone’s use cases



Guix and Nix present a new paradigm for software distribution and deployment.

They can replace a wide range of package management, system configuration, and containerisation tools that are in use today.

Whether you want to create a consistent development environment for a project, configure the OS on you PC, or deploy the same setup to every computer on a network, Guix and Nix can handle it.

Finally, both, though Nix especially, have the potential to become a universal standard that covers everyone’s use cases.

Image Sources

- https://github.com/NixOS/nixos-artwork/blob/master/logo/nixos.svg
  - Tim Cuthbertson
- http://git.savannah.gnu.org/cgit/guix/guix-artwork.git - Luis Felipe López Acevedo
- https://xkcd.com/927/

Any Questions?

Nix > Guix if you don’t care about software freedom or Lisp.