<2020-02-19 Mi> Dr. Arne Babenhauserheide / draketo.de

Using GNU Guix Linux for software development

When I started working as a software developer at Disy, I also switched my main distribution at home to GNU Guix.

Guix enables me to run updates without fearing that losing power during the update could kill my system (I had that happen with Gentoo), it provides a stable base-system with per-user installed packages as well as environments that can install packages into the local shell much like language-specific package managers like npm. Also it aligns well with my ideals of using purely free software.

But I mostly do proprietary software development at work, and advanced features of Guix make some of that problematic. This page lists the workarounds I'm using.

Contents

IntelliJ (Ultimate)

We're using IntelliJ for Java development, because it is the best tool out there for a 1.5 Million lines Java codebase. But it requires running software which is not installed via the package manager.

IntelliJ running natively

I run IntelliJ via the file `~/.local/bin/idea.sh` which sets a custom LD_LIBRARY_PATH and the system-JDK, and then runs the idea.sh script which comes with IntelliJ.

This is sensitive to a lot of things, but it currently works. I wish it were easier.

Part of the library path is required for our starters, because of native code that has to find the libraries and libstdc++.

Update: Cleaned up finding the path to libstdc++ with Guix onboard tools.

```
#!/bin/bash
cd ~/
set -x
```

```
# HACK: use system guix to avoid glibc inconsistency problems
# PATH=/var/guix/profiles/system/profile/bin/:$PATH

JAVA17_PACKAGE="openjdk@17"
JDK17_PATH="$(guix build ${JAVA17_PACKAGE} | grep -- '-jdk$')"
# need openjdk 15, because 16 stops with module errors
JAVA15_PACKAGE="openjdk@15"
JDK15_PATH="$(guix build ${JAVA15_PACKAGE} | grep -- '-jdk$')"
# need openjdk 14, because 16 stops with module errors
JAVA14_PACKAGE="openjdk@14"
JDK14_PATH="$(guix build ${JAVA14_PACKAGE} | grep -- '-jdk$')"
# also need openjdk 11 as the official build configuration
JAVA11_PACKAGE="openjdk@11"
JDK11_PATH="$(guix build ${JAVA11_PACKAGE} | grep -- '-jdk$')"
# need to track libstdc++ in the dependencies of the GCC toolchain
GCC_TOOLCHAIN="gcc-toolchain"
GCC_LIB_PATH="$(grep -oE "[^\"]*gcc-12[^\"]*-lib" $(grep -oE "[^\"]*gcc-12[^\"]*drv"
SQLITE_PATH="$(guix build sqlite | head -n 1)"
SPATIALITE_PATH="$(guix build libspatialite)"

MAVEN_HOME=$(guix build maven)

exec -a "$0" guix environment --ad-hoc ${JAVA11_PACKAGE}:jdk ${JAVA14_PACKAGE}:jdk ${
export JDK15_PATH=\"$(guix build ${JAVA15_PACKAGE} | grep -- '-jdk$')\"
export JDK14_PATH=\"$(guix build ${JAVA14_PACKAGE} | grep -- '-jdk$')\"
# keep working in exwm
export _JAVA_AWT_WM_NONREPARENTING=1
export AWT_TOOLKIT=MToolkit
export JDK11_PATH="$(guix build ${JAVA11_PACKAGE} | grep -- '-jdk$')"

LD_LIBRARY_PATH="'${GUIX_ENVIRONMENT}'"/lib:${SQLITE_PATH}/lib:${SPATIALITE_PATH}/lib
```

For info on the (no longer necessary) trap, see http://redsymbol.net/articles/bash-exit-traps/

The fork-hack to run code after exec is from that other guy on stackoverflow.

IntelliJ with flatpak

Alternative approach: just use **flatpak**. But rmic is missing (because it was removed in openjdk@15) and IntelliJ via Flatpak has problems with unicode filenames (cannot build).

Besides flatpak this only requires setting environment variables (to make it work in exwm; see Emacs Tipps for info about exwm).

```
#!/usr/bin/env bash
export _JAVA_AWT_WM_NONREPARENTING=1
export AWT_TOOLKIT=MToolkit
exec flatpak run com.jetbrains.IntelliJ-IDEA-Ultimate
```

Maven

I run a custom version of Maven via a simple script:

```
#!/bin/sh

NOTE='\033[1;33m'
NONE='\033[0m'

echo "[${NOTE}NOTE${NONE}] Running Maven with:"
java -version

~/Disy/opt/apache-maven-3.6.1/bin/mvn "$@"
```

npm

npm is a bit of a beast: If I use what we create via maven, it dies because it does not find its libraries. Therefore I install it manually with the system-installed npm and then run `npm-cli.js` directly:

```
npm install npm@6.13.4 ; node_modules/npm/bin/npm-cli.js install; node_modules/npm/bi
```

To simplify this, I moved it into a shell-script at `~/.local/bin/npm`:

```
#!/usr/bin/env bash
# if ! echo $PWD | grep -q Disy; then
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/home/arne/.guix-profile/lib:/gnu/store/gm2
exec -a "$0" guix environment --ad-hoc gcc-toolchain@10.2 -- /home/arne/.guix-profile
```

Firefox

I actually tried to package Firefox for Guix, but packaging a few hundred Rust packages is quite some task. I stopped at more than 200 packaged ones, because the version numbers retrieved from `guix import crate NAME` did not match the version numbers needed but rather always retrieved the most recent version.

Though it ultimately failed, this is the script I used to automatically import many Rust packages:

```
for i in $(for i in $(grep -o ',[^ ()][^ ()]*' more/packages/gnuzilla.scm | sed s.,..
               grep -q "name \"$i\"" more/packages/gnuzilla.scm || echo $i;
           done | sort -u | grep rust | sed s/^rust-//  | xargs); do
    guix import crate $i >> more/packages/gnuzilla.scm || \
        (sleep 30 && guix import crate $i >> more/packages/gnuzilla.scm);
done
```

The actual solution to get Firefox was to install flatpak and get Firefox from there:

```
guix install flatpak
flatpak install --user firefox
```

Now I run Firefox via a small script at `~/.local/bin/firefox`:

```
#!/usr/bin/env bash
flatpak run org.mozilla.firefox "$@"
```

Zoom

For Zoom I use the same method as for Firefox:

```
flatpak install flathub us.zoom.Zoom
```

```
#!/run/current-system/profile/bin/bash
flatpak run us.zoom.Zoom
```

Custom Emacs profile

All my planning files and my setup to develop Javascript are in a specialized Emacs setup that I run with `disymacs`:

```
#!/usr/bin/env bash
HOME="${HOME}/Disy" emacs "$@"
```

There is a full custom setup in `~/Disy/.emacs.d` so I can more easily synchronize the config between office and home office.

Connect to VPN

I use `openconnect` and a custom script to connect to our VPN.

```
guix install openconnect
```

```
#!/run/current-system/profile/bin/bash
if [ "$EUID" -ne 0 ]; then
  echo This script needs root priviledges. 1>&2
  echo Executing sudo --login $(realpath "$0") in 3 seconds 1>&2
  for i in {1..3}; do
      echo -n .
      sleep 1
  done
  echo " " now executing sudo --login $(realpath "$0")
  exec sudo --login $(realpath "$0")
fi
# Connect via VPN to disy and ensure that the nameservers are correct
function resolv-disy-undo {
    echo "undoing VPN connection"
    cp /etc/resolv.conf.head-default /etc/resolv.conf.head
    # ensure that the manual DNS servers are available
    # let dhcp do the rest # TODO: this only worked in Gentoo. Find out how it works
    cp /etc/resolv.conf /tmp/resolv.conf && cat /etc/resolv.conf.head /tmp/resolv.con
    # /lib/dhcpcd/dhcpcd-run-hooks
}

# ensure that the change is undone when this script ends, see http://redsymbol.net/ar
trap resolv-disy-undo EXIT HUP INT QUIT ABRT TERM KILL

if test ! -e /etc/resolv.conf.head-default; then
    cp /etc/resolv.conf.head /etc/resolv.conf.head-default || exit 1
fi

# /etc/resolv.conf.head-disy contains our internal name servers,
# one of them duplicated, because dhcpcd only uses the first three nameservers.

cp /etc/resolv.conf.head-disy /etc/resolv.conf.head
# /lib/dhcpcd/dhcpcd-run-hooks # update /etc/resolv.conf

echo
echo When requested, type your PASSWORD directly followed by the AUTHENTICATOR-TOKEN
echo

(sleep 30 && sshfs -o allow_other,defer_permissions USER@NODE:/path/to/disk /path/to/

openconnect --protocol=gp gp.disy.net --csd-wrapper="$HOME/.guix-profile/libexec/open

resolv-disy-undo
reset
```

Run tomcat with Java 8

I run my system with OpenJDK 12, and we develop with Java 11, but our secondary Tomcat setup needs Java 8. So I create a local environment with Java 8 (via icedtea 3.7).

```
# download and unpack tomcat and enter the folder, then call
guix environment --ad-hoc icedtea@3.7.0
bin/startup.sh
```

Versiontracking via magit and monky

To make git usable I use magit. It gives me a good balance of efficiency and control.

For Mercurial, my preferred version tracking tool, I now use monky to have similar integration into Emacs as with magit.

Full update of installed packages

`guix pull && guix update -k` can be slow if substitutes cannot be found, so everything has to be built locally. I still don't know why that happens, but I now have a way to avoid it: I just re-install all installed packages.

```
#!/usr/bin/env bash
# re-install all guix packages
guix pull && guix package -I | cut -f 1,3 | sed 's/\t/:/' | xargs guix install
```

I saved this as `~/.local/bin/update-guix-full`.

Join Alfaview meeting from the commandline

For lectures with so many people that jitsi and nextcloud meet their limits. European alternative to Zoom.

Add a package for guix (in `~/Dokumente/Guix/guix`) that adds untested gl-support to gst-plugins-base (the tests failed and I did not get them to run):

```
diff --git a/gnu/packages/gstreamer.scm b/gnu/packages/gstreamer.scm
index bb991789da..324af10ee0 100644
--- a/gnu/packages/gstreamer.scm
+++ b/gnu/packages/gstreamer.scm
@@ -581,6 +581,17 @@ (define-public gst-plugins-base
 for the GStreamer multimedia library.")
```

```
     (license license:lgpl2.0+)))

+(define-public gst-plugins-base-gl
+  (package (inherit gst-plugins-base)
+    (name "gst-plugins-base-gl")
+    (inputs
+     `(("mesa" ,mesa) ;; required for libgstgl
+       ,@(package-inputs gst-plugins-base)))
+    (arguments
+     `(,@(package-arguments gst-plugins-base)
+       #:tests? #f ;; check fails with gl
+       #:configure-flags '("-Dgl=enabled"))))) ;; requires mesa)
+
 (define-public gst-plugins-good
   (package
     (name "gst-plugins-good")
```

First get the debian image and unpack it. Then unpack the data tarball.

```
#!/usr/bin/env bash
cd /home/arne/Downloads/alfaview/data/opt/alfaview || exit 1
export GCC_TOOLCHAIN="gcc-toolchain@11.2"
export GCC_LIB_PATH="$(grep -oE "[^\"]*gcc-11[^\"]*-lib" $(grep -oE "[^\"]*gcc-11[^\"
exec -a "$0" ~/Dokumente/Guix/guix/pre-inst-env guix environment --ad-hoc \
  gst-plugins-base-gl gst-plugins-good gst-plugins-bad alsa-plugins alsa-lib pulseaud
  libxcb xcb-util-wm xcb-util-renderutil xcb-util-image xcb-util-keysyms libxkbcommon
  fontconfig libinput bash openssl@1.1 dbus libsecret "${GCC_TOOLCHAIN}" -- bash -c \
  "LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${GCC_LIB_PATH}/lib:${OPENSSL_LIB_PATH}/lib:"'$
```

Adapt the paths, then create this file as `~/.local/bin/alfaview` and make it executable with `chmod +x ~/.local/bin/alfaview`.

To connect to a session, open the browser URL until you get to screen with the "accept AGB/TOS"-checkbox. Check that, then right-click and copy the url and use it like this:

```
alfaview --url 'alfaview://...'
```

Unlock cores

In my Guix config I throttle my cores to reduce power consumption. But for work I often need full performance in short bursts — for example when IntelliJ re-indexes sources or when I rebuild after changes. For this, I have a small core-unlocker script saved at `~/.local/bin/boost-cpu`:

```
#!/usr/bin/env bash
```

```
# boost CPU for MIN minutes
if [[ x"$1" == x"" ]]; then
    echo "Error: missing argument."
    echo "usage: $0 <minutes>"
    exit 1
fi
if ! test $1 -eq $1 2>/dev/null; then
    echo "Error: argument '$1' is not a number."
    echo "usage: $0 <minutes>"
    exit 1
fi
if [[ x"$1" == x"--help" ]]; then
    echo "$0 <MIN>: boost CPU speed for MIN minutes."
    exit 0
fi

SEC="$((($1 * 60)))"

for i in $(seq 1 $SEC); do sudo cpupower frequency-set -u 6000000; sleep 1; done
```

Test against IE11 in a virtual machine

First get one of the IE11 VMware images from Microsoft: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

```
wget https://az792536.vo.msecnd.net/vms/VMBuild_20190311/VirtualBox/MSEdge/MSEdge.Win
```

Unzip and untar it:

```
unzip MSEdge.Win10.VirtualBox.zip
tar xvf MSEdge\ -\ Win10.ova
```

Then convert if for use with qemu:

```
qemu-img convert -f vmdk IE11-Win81-VMWare-disk1.vmdk -O qcow2 W81.qcow2 -p
```

Finally run the VM and access your local service over your internal (NAT-) IP (use `ifconfig` to get that):

```
qemu-system-x86_64 -accel kvm -smp 4 -m 8000 -hda W81.qcow2
```

If you organize with org-mode, you can create a link that runs Windows when you open it (with `C-c C-o` or by clicking it) like this:

```
[[shell:cd /mnt/blattwerk/downloads && qemu-system-x86_64 -accel kvm -smp 4 -m 8000 -
```

Get the (OWA) Exchange-Calendar in the org-agenda via ICS

In OWA open the **Options**. Then select under Calendar and sharing the **publish calendar** option. Choose the calendar and **export all details**. Then copy the link to the **ICS file**.

For this example let's assume that it is http://ccc-ffm.de/wp-content/uploads/2015/02/Vortrag-FSFE-Freie_Software_in_der_Bildung.ics.

Now add the following to your `~/.emacs.d/init.el`:

```
(use-package org-agenda
  :defer 6
  :custom
  (alert-default-style 'libnotify)
  (appt-disp-window-function 'alert-for-appt)
  (org-agenda-include-diary t)
  (appt-delete-window-function (lambda ()))
  :config
  ;; Add calendar, option 3  https://www.ict4g.net/adolfo/notes/emacs/emacs-caldav.ht
  (unless (file-exists-p "~/.emacs.d/diaries/")
    (make-directory "~/.emacs.d/diaries/"))
  (setq diary-location "~/.emacs.d/diaries/")
  (setq calendars
        '(("exchangeexport" . "http://ccc-ffm.de/wp-content/uploads/2015/02/Vortrag-F
  ;; remember to add
  ;; #include "diaries/exchangeexport"
  ;; into ~/.emacs.d/diary
  ;; Rebuild reminders everytime the agenda is displayed
  (add-hook 'org-agenda-finalize-hook (lambda () (org-agenda-to-appt t)))
  ;; Run once when Emacs starts
  (org-agenda-to-appt t)
  ;; Activate appointments so we get notifications
  (appt-activate t)
  (add-hook 'diary-list-entries-hook 'diary-include-other-diary-files)
  (add-hook 'diary-mark-entries-hook 'diary-mark-included-diary-files)
  (defun getcal (url file)
    "Download ics file and add it to file"
    (let ((tmpfile (url-file-local-copy url)))
      (icalendar-import-file tmpfile file)
      ;; preserve previous calendar entries in case of network problems
      (when (string-blank-p (buffer-string)) (undo))
      (kill-buffer (car (last (split-string tmpfile "/"))))))
  (defun getcals ()
    "Load a set of ICS calendars into Emacs diary files"
    (interactive)
    (save-mark-and-excursion
      (save-window-excursion
        (with-silent-modifications
          (mapcar #'(lambda (x)
                      (let ((file (concat diary-location (car x)))
                            (url (cdr x)))
                        (message (concat "Loading " url " into " file))
                        (find-file file)
                        ;; (flush-lines "^[& ]") ;; if you import ical as non marking
                        (erase-buffer) ;; to avoid duplicating events
                        (getcal url file)))
                  calendars)))))
  ;; for work calendars, update the calendar every 3 hours
  (defun appt-reparse-diary-file ()
    "force reparsing the diary file"
    (appt-check t))
  (add-to-list 'midnight-hook 'getcals)
  (add-to-list 'midnight-hook 'appt-reparse-diary-file)
  (setq midnight-period 10800) ;; (eq (* 3 60 60) "3 hours")
  (midnight-mode t))
```

Finally include the new calendar in your diary-file, so you see the entries in `M-x diary` and the `M-x calendar`:

```
echo '#include "diaries/exchangeexport"' >> ~/.emacs.d/diary
```

For desktop notifications, look into `org-agenda-to-appt`.

Fix font support in Chromium

```
xset +fp $(dirname $(readlink -f ~/.guix-profile/share/fonts/truetype/fonts.dir))
fc-cache -rv
```

Eslint

Save the following as `~/.local/bin/eslint`:

```
#!/usr/bin/env bash
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/home/arne/.guix-profile/lib:/gnu/store/rgj
exec -a "$0" guix environment --ad-hoc gcc-toolchain@10.3 -- /home/arne/.guix-profile
```

Eslint requires libc in the path. Since I did not find how to get only the gcc library as package, I search for the gcc package used by the gcc-toolchain via

```
ls -d /gnu/store/*gcc-10.3.0-lib/lib/ &
```

Note the version 10.3.

run npm node binary

```
GCC_TOOLCHAIN="gcc-toolchain@10.3"
GCC_LIB_PATH="$(grep -oE "[^\"]*gcc-10[^\"]*-lib" $(grep -oE "[^\"]*gcc-10[^\"]*drv"
guix environment --ad-hoc nss node ${GCC_TOOLCHAIN} sqlite libspatialite node -- bash
  "LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${GCC_LIB_PATH}/lib:$(realpath ~/.guix-profile/
```

signal-cli

Signal Cli enables interaction with Signal from the commandline (you only need a mobile phone to receive an SMS). It is built and run with gradle, so it needs some tooling to become nice. This is what I built.

Build signal-cli

```
git clone https://github.com/AsamK/signal-cli.git
cd signal-cli
guix shell openjdk:jdk -- ./gradlew build
```

Run signal-cli

I created a small commandline-tool and put it to `~/.local/bin/signal-cli`:

```
#!/usr/bin/env bash
cd /path/to/signal-cli && \
if [[ x"$4" != x"" ]]; then
exec -a "$0" guix shell openjdk:jdk -- ./gradlew -q run --args="-a +YOUR_PHONENUMBER
elif [[ x"$3" != x"" ]]; then
exec -a "$0" guix shell openjdk:jdk -- ./gradlew -q run --args="-a +YOUR_PHONENUMBER
elif [[ x"$2" != x"" ]]; then
exec -a "$0" guix shell openjdk:jdk -- ./gradlew -q run --args="-a +YOUR_PHONENUMBER
elif [[ x"$1" != x"" ]]; then
exec -a "$0" guix shell openjdk:jdk -- ./gradlew -q run --args="-a +YOUR_PHONENUMBER
else
echo Missing argument: Received arguments "$@"
```

```
fi
```

For setup, see signal-cli#usage.

Start shepherd for user-level services

Put the following into `~/.bash_profile` or `~/.profile` to ensure that your user-level shepherd services run:

```
# shart shepherd for this user, if it is not running yet
pgrep -u $(whoami) shepherd  -l | grep -q shepherd || shepherd 2>/dev/null >/dev/null
```

(this feels like a hack and there likely is a better way, but with this I have something working)

Running jest with playwright

Put this to `~/.local/bin/jest-e2e`.

```
#!/usr/bin/env bash
exec -a "$0" guix shell samba gcc-toolchain icecat alsa-lib glib nss nss:bin \
     nss-certs nspr atk cups gnome libdrm dbus expat libxcb libxkbcommon \
     glibc sqlite maven zlib e2fsprogs bash node -- bash -c \
     "cd ~/path/to/src/;
  LD_LIBRARY_PATH="'${GUIX_ENVIRONMENT}'"/lib:"'${GUIX_ENVIRONMENT}'"/lib/icecat:$(gr
  -oE "[^\"]*gcc-12[^\"]*-lib" $(grep
    -oE "[^\"]*gcc-12[^\"]*drv" $(guix
      build -d gcc-toolchain)) | head -n 1)/lib \
  JEST_PLAYWRIGHT_CONFIG=playwright-configs/chrome-playwright.config.js \
  node node_modules/jest/bin/jest.js $@ || ls -l "'${GUIX_ENVIRONMENT}'
```

(you might need to remove some linebreaks in the command-string)

Run gephi from Guix

```
wget https://github.com/gephi/gephi/releases/download/v0.10.1/gephi-0.10.1-linux-x64.
tar xf gephi-0.10.1-linux-x64.tar.gz
cd gephi-0.10.1
# keep working in exwm
export _JAVA_AWT_WM_NONREPARENTING=1
export AWT_TOOLKIT=MToolkit
export LIBGL_ALWAYS_SOFTWARE=1
# this pulls in mesa, but OpenGL still doesn't work.
```

```
guix shell mesa mesa-opencl mesa-headers libdrm libglvnd freeglut openjdk@11:jdk \
     -- bash -x -c 'export PATH=$GUIX_ENVIRONMENT/bin:$PATH;
bin/gephi --jdkhome "$(dirname $(dirname $(which java)))" 2>&1'
```

Update Guix Grub EFI Bootloader when updating from a fallback-system

My new Ryzen onboard graphics card does not support CMS (Compatibility Mode), thank you for those 10 hours you shaved off from my life.

When you update from a non-EFI system to an EFI-system, you have no grub in the EFI partition, so the BIOS recognizes **no bootable disk**.

To fix this, you need to boot from a recent USB stick, changeroot into your system, and reconfigure.

This is what I had to do on my system for that:

```
sudo su -
# thanks to the ever awesome Gentoo Wiki; https://wiki.gentoo.org/wiki/Chroot/de
mkdir /mnt/nvme2n1p2
mount /dev/nvme2n1p2 /mnt/nvme2n1p2
mount /dev/nvme0n1p1 /mnt/nvme2n1p2/home
mount --rbind /dev /mnt/nvme2n1p2/dev
mount --make-rslave /mnt/nvme2n1p2/dev
mount -t proc /proc /mnt/nvme2n1p2/proc
mount --rbind /sys /mnt/nvme2n1p2/sys
mount --make-rslave /mnt/nvme2n1p2/sys
mount --rbind /tmp /mnt/nvme2n1p2/tmp
cp /etc/resolv.conf /mnt/nvme2n1p2/etc
# thanks to Thorsten Wilms:
# https://lists.gnu.org/archive/html/help-guix/2018-03/msg00116.html
# give your sacrifice to systemd, took me over 1h to find
unshare -m
# started a new shell?
mount /dev/nvme2n1p1 /mnt/nvme2n1p2/boot/efi/
chroot /mnt/nvme2n1p2/ /run/current-system/profile/bin/bash
# started a new shell
source /etc/profile
guix-daemon --build-users-group=guixbuild &
# install the bootloader from the USB’s EFI system so it gets installed correctly.
guix system reconfigure -M8 -c6 -k /etc/config.scm
```

You might also need this if your EFI partition breaks.

You might also need to delete efi logs from the firmware to make space:

```
# rm /sys/fs/efi/efivars/dump-*
```

Compile and Run FMS (Freenet Message System)

Building Software with glibc dependency can need some additional steps on Guix. The following builds and runs FMS (replace `/path/to/fms`):

```
cd /path/to/fms/fms-src-0.3.85/ \
  && guix shell gcc-toolchain cmake glibc -- \
  bash -c 'make clean ; \
    cmake -D I_HAVE_READ_THE_README=ON -D CMAKE_C_COMPILER=$(which gcc) \
      -D CMAKE_CXX_COMPILER=$(which g++) . \
    && make' \
  && guix shell gcc-toolchain gcc cmake glibc -- \
    bash -c 'cd /path/to/fms/ \
    && LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GUIX_ENVIRONMENT/lib \
    ./fms-src-0.3.85/fms'
```

[2023-04-30 So]

A container with X11

To run a subshell as Container with access to X11, you can use

```
~/Dokumente/Guix/guix/pre-inst-env guix shell -CN \
  -E DISPLAY -E XAUTHORITY -E XAUTHLOCALHOSTNAME \
  --share=/tmp/.X11-unix --share=/run/user/1000/gdm \
  <packages> -- <program call>
```

You may have to allow access to the docker IP you find with `ifconfig` in the container via

```
xhost <IP> # in the host
```

[2023-08-11 Fr]

Generate a full software bill of materials

In GNU Guix I can always understand completely which dependencies are included - or have been included - in a build. A truly complete software bill of materials is therefore a simple shell command:

```
guix graph --type=bag <package> | grep -o label.* | sort -u
```

Some results:

- (ungoogled) Chromium: 859 dependencies
- icecat (Firefox): 739
- openjdk: 422
- Python: 129
- httpd (Apache): 99
- nginx: 107
- Guile: 91 (but this is because the bootstrap path was minimized for Guile on Guix - including projects like mes: Maxwell's Equations of Software, which is based on a Scheme boostrapped from assembly — because it's part of the system stack, so they've been working for years on using as few dependencies as possible for it).
- Perl: 91.

And this then is the true software bill of materials due to the dependency.

[2023-09-24 So]