Installing a Program on Guix

Dr. Arne Babenhauserheide

<2021-09-14 Di>

Usually if you want to buid a program with non-standard libraries on a GNU Linux system, you install the libraries in your HOME or in a library-folder one by one and then build the final program. With Guix this can be done cleaner and more elegantly. Let me show you the example of FMS, the Freenet Message System.

Start the Guix

Start by creating a `guix.scm` file in my program directory. That's the package format of Guix. The simplest file just defines a module:

```
(define-module (gnu packages freenet))
```

I call the package-file freenet, because I might package more later. Guix puts related packages in the same file and allows selecting from them.

Now I could continue with how Guix works in detail and with the file format and so on. But that is not how I *actually* package something for guix.

What I actually do is to copy the header of a package file with programs similar to the one I want to install. FMS needs a `C++` library (poco), so I start with gnu/packages/cpp.scm. This gives me a big header and far too many imports (that I will not repeat here, just click the link to `cpp.scm` above).

The Library/-ies: provide poco

To package poco, I first search for a program that is similar. Poco uses cmake and requires openssl. Searching in `cpp.scm` I find rct that also uses cmake and requires openssl. Its definition looks like this:

```
(define-public rct
  (let* ((commit "b3e6f41d9844ef64420e628e0c65ed98278a843a")
         (revision "2"))
```

```
    (package
      (name "rct")
      (version (git-version "0.0.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://github.com/Andersbakken/rct")
                      (commit commit)))
                (sha256
                 (base32
                  "1m2931jacka27ghnpgf1z1plkkr64z0pga4r4zdrfpp2d7xnrdvb"))
                (patches (search-patches "rct-add-missing-headers.patch"))
                (file-name (git-file-name name version))))
      (build-system cmake-build-system)
      (arguments
       '(#:configure-flags
         '("-DWITH_TESTS=ON"            ; To run the test suite
           "-DRCT_RTTI_ENABLED=ON")))
      (native-inputs
       `(("cppunit" ,cppunit)
         ("pkg-config" ,pkg-config)))
      (inputs
       `(("openssl" ,openssl)
         ("zlib" ,zlib)))
      (home-page "https://github.com/Andersbakken/rct")
      (synopsis "C++ library providing Qt-like APIs on top of the STL")
      (description "Rct is a set of C++ tools that provide nicer (more Qt-like)
 APIs on top of Standard Template Library (@dfn{STL}) classes.")
      (license (list license:expat        ; cJSON
                     license:bsd-4)))))   ; everything else (LICENSE.txt)
```

As you can see, there is more than I need. So I strip it down and replace urls and descriptions. Finally I add `poco` to the end of the file. That is the return value. That way I get this:

```
;;; GNU Guix --- Functional package management for GNU
;;; Copyright © ...
;;; --- many more lines ---

(define-module (gnu packages freenet)
  #:use-module ((guix licenses) #:prefix license:)
  ;; many more use-module lines
  )

(define-public poco
```

```
  (let* ((commit "poco-1.11.0-release")
         (revision "2"))
    (package
      (name "poco")
      (version (git-version "1.11.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://github.com/pocoproject/poco")
                      (commit commit)))
                (sha256
                 (base32
                  "1m2931jacka27ghnpgf1z1plkkr64z0pga4r4zdrfpp2d7xnrdvb"))
                (file-name (git-file-name name version))))
      (build-system cmake-build-system)
      (native-inputs
       `(("cppunit" ,cppunit)
         ("pkg-config" ,pkg-config)))
      (inputs
       `(("openssl" ,openssl)))
      (home-page "https://pocoproject.org/")
      (synopsis "cross-platform C++ libraries for building network- and internet-base
      (description "The POCO C++ Libraries are powerful cross-platform C++ libraries
      (license (list license:boost1.0)))))

;; return value: this is the package that will be built
poco
```

If you look closely, you'll see that I did not change the base32 hash. I did not, because I don't know it yet. To fix that, I add `poco` to the end of the file and run:

```
LANG=C guix build -f guix.scm
```

I receive an error (LANG=C forces the output in English — the actual error I see is in German):

```
r:sha256 hash mismatch for /gnu/store/wf153m4q9q7sm8r01wxdwlanq3s4zwv0-poco-1.11.0-2.
  expected hash: 1m2931jacka27ghnpgf1z1plkkr64z0pga4r4zdrfpp2d7xnrdvb
  actual hash:   052lhlr9mb6ahinjj647lf18dzgjp3rbj97q8xm6fv07yr7vzn94
hash mismatch for store item '/gnu/store/wf153m4q9q7sm8r01wxdwlanq3s4zwv0-poco-1.11.0
```

And this gives me what I need. I replace the hash by the actual hash and run `guix build` again.

Again I receive an error, now because there is no test target for make:

```
starting phase `check'
```

```
make: *** No rule to make target 'test'.  Stop.
```

So I search in cpp.scm again and find a configure-flag to enable tests (you can also find this in the CMakeLists.txt file):

```
(arguments
 '(#:configure-flags
   '("-DENABLE_TESTS=ON")))
```

To be able to inspect the build tree if something goes wrong, I now build with –keep-failed:

```
guix build -f guix.scm --keep-failed
```

The proper Guix packager would now investigate, why there are test failures. But if you just need something running right-away, you can also go the risky route and disable tests:

```
(arguments '(#:tests? #f))
```

I went into the source folder left there due to –keep-failed and ran the tests with `make test`. There were failures in NET and redis and postgres and a segfault in mongo. But I have no more time, so I go the risky route for now and disable tests. Sorry for anyone wanting to simply copy this package definition.

Poco is as ready as I'll get it for now (or whatever dependencies you need — just put their names at the end of the file). On to FMS itself.

The program: install FMS

The next step is to package from the local source folder. You can read the documentation and figure out how to do it, or, like me, you have an existing project lying around from procrastination that you can simply copy and adapt. Maybe you also find one on some random website. One like this one:

```
(define-public fms
  (package
    (name "fms")
    (version "0.1")
   (source (local-file "." #f #:recursive #t))
      (build-system cmake-build-system)
      (arguments
       '(#:tests? #f
         #:configure-flags
         '("-DI_HAVE_READ_THE_README=ON")))
      (native-inputs
       `(("pkg-config" ,pkg-config)))
      (inputs
```

```
       `(("poco" ,poco)))
      (home-page "USK@0npnMrqZNKRCRoGojZV93UNHCMN-6UU3rRSAmP6jNLE,~BG-edFtdCC1cSH4O3B
      (synopsis "Forums System for Freenet")
      (description "The Freenet Message System is a reference specification and imple
for newsgroup like communication inside of Freenet.")
      (license (list license:gpl2 license:bsd-3)))) ;; for dependencies

fms ;; replaced poco to make the setup build fms instead of poco
```

Now ensure that you start with a clean source tree and build this. This is how I did it:

```
mkdir -p fms-src-0.3.83 && \
  cp guix.scm fms-src-0.3.83.zip fms-src-0.3.83 && \
  cd fms-src-0.3.83 && \
  unzip fms-src-0.3.83.zip && \
  guix build -f guix.scm --keep-failed
```

First this will fail with *missing package* and suggest adding `(guix gexp)`, so just add that to the module definition:

```
(define-module (gnu packages freenet)
  ;; ...
  #:use-module (guix gexp)
  ;; ...
  )
```

Once this works, it is time to install the program cleanly (modulo ignoring tests):

```
mkdir -p fms-src-0.3.83 && \
  cp guix.scm fms-src-0.3.83.zip fms-src-0.3.83 && \
  cd fms-src-0.3.83 && \
  unzip fms-src-0.3.83.zip && \
  guix package -f guix.scm
```

That's it. FMS is installed for my user.

Environment: build in local directory

What if — like FMS — the package does not actually install but just build a local binary? Then you can use `guix environment` to provide all libraries and run the build locally:

```
mkdir -p fms-src-0.3.83 && \
  cp guix.scm fms-src-0.3.83.zip fms-src-0.3.83 && \
  cd fms-src-0.3.83 && \
  unzip fms-src-0.3.83.zip && \
```

```
guix environment -l guix.scm -- cmake -DI_HAVE_READ_THE_README=ON . && \
guix environment -l guix.scm -- make
```

That's it. Now I have an `fms` executable that works.

Cleanup

The `guix.scm` file still contains far too many imports. I simply comment out all `#:use-module` lines and re-add those guix requests

Summary

This tutorial provides full integration of arbitrary programs into the Guix package manager, including installation, de-installation, and libraries that do not interfere with other libraries.

You now have the power of virtualenv for any library and any program. Have fun!

If you do not want to install it, but just to use it for testing, have a look at guix environment.

Appendix: the full package definition

```
(define-module (gnu packages freenet)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (guix packages)
   #:use-module (guix gexp)
   #:use-module (guix git-download)
   #:use-module (guix build-system cmake)
   #:use-module (gnu packages tls))

(define-public poco
  (let* ((commit "poco-1.11.0-release")
         (revision "2"))
    (package
      (name "poco")
      (version (git-version "1.11.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://github.com/pocoproject/poco")
                      (commit commit)))
```

```
                (sha256
                 (base32
                  "052lhlr9mb6ahinjj647lf18dzgjp3rbj97q8xm6fv07yr7vzn94"))
                (file-name (git-file-name name version))))
      (build-system cmake-build-system)
      ;; (arguments '(#:tests? #f))
      (arguments
       '(#:tests? #f
         #:configure-flags
         '("-DENABLE_TESTS=ON")))
      (inputs
       `(("openssl" ,openssl)))
      (home-page "https://pocoproject.org/")
      (synopsis "cross-platform C++ libraries for building network- and internet-base
      (description "The POCO C++ Libraries are powerful cross-platform C++ libraries
      (license (list license:boost1.0)))))

(define-public fms
  (package
    (name "fms")
    (version "0.3.83")
   (source (local-file "." "src" #:recursive? #t))
   (build-system cmake-build-system)
   (arguments
    '(#:tests? #f
      #:configure-flags
      '("-DI_HAVE_READ_THE_README=ON")))
   (inputs
    `(("poco" ,poco)))
   (home-page "USK@OnpnMrqZNKRCRoGojZV93UNHCMN-6UU3rRSAmP6jNLE,~BG-edFtdCC1cSH4O3BWde
   (synopsis "Forums System for Freenet")
   (description "The Freenet Message System is a reference specification and implemen
for newsgroup like communication inside of Freenet.")
   (license (list license:gpl2 license:bsd-3)))) ;; for dependencies

fms
```