GNU Guix Reference Card

for version 1.5.0
https://guix.gnu.org/

Getting Started

To read the on-line documentation run `info guix` or visit *https://guix.gnu.org/manual/stable.*

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 search` *regexp* …	search for packages
`guix show` *spec*	show package info
`guix install` *spec…*	install packages
`guix upgrade` [*regexp*]	upgrade packages
`guix remove` *name…*	remove packages
`guix package -m` *file*	instantiate from manifest
`guix package --export-manifest`	export profile contents as manifest
`guix package --roll-back`	roll back
`guix package -I`	list installed packages
`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 shell` *spec* …
  environment containing *spec* …
`guix shell -D` *spec*
  environment to develop *spec*
`guix shell` *spec* … `-C --` *command* …
  run *command* … in a container
`guix shell --check`
  check if the shell clobbers environment variables
`guix shell -m` *file*
  create an environment for the packages in manifest *file*

Updating Guix

`guix describe`	describe current Guix
`guix describe -f channels`	produce a channel spec
`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)
```

Using a Different Version of Guix

The `guix time-machine` command provides access to other revisions of Guix, for example to install older versions of packages, or to reproduce a computation in an identical environment.

`guix time-machine -C` *file* `--` *commands* …
  Run *commands* in a version of Guix specified by the given channels in *file*

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* `--with-git-url=`*spec*`=`*URL*
  build *spec* from the given Git *URL*
`guix` *command spec* `--with-branch=`*package*`=`*branch*
  build *spec* from the given Git *branch* of *package*
`guix` *command spec* `--with-commit=`*package*`=`*commit*
  build *spec* from the given Git *commit* of *package*
`guix` *command spec* `--with-patch=`*package*`=`*file*
  build *spec* after applying the given patch *file* to *package*
`guix` *command spec* `--with-latest=`*package*
  build *spec* using the latest upstream release for *package*
`guix` *command spec* `--with-c-toolchain=`*package*`=`*toolchain*
  build *spec* using *toolchain* for *package*
`guix` *command spec* `--without-tests=`*package*
  build *spec* without running the tests for *package*
`guix` *command spec* `--with-debug-info=`*package*
  build *spec* against *package* with debugging info
`guix` *command spec* `--with-configure-flag=`*package*`=`*flag*
  build *spec* adding *flag* to *package*'s configure phase
`guix` *command spec* `--tune`[`=`*arch*]
  optimize tunable packages for *arch* or the host architecture

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*

Guix

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 -f deb` *spec* …	create a Debian package archive
`guix pack -RR` *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*

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 gc -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

Managing the Home Environment

`guix home` takes a configuration file that declares dotfiles, packages, and user services.

`guix home import` *directory*
populate *directory* with an initial Home configuration

`guix home search` *regexp*
search for services matching *regexp*

`guix home container` *file*
try the configuration in *file* in a container

`guix home reconfigure` *file*
reconfigure the home according to the configuration in *file*

`guix home delete-generations` *pattern*
delete generations matching *pattern*

`guix home roll-back`
roll back to the previous generation

Declaring an Operating System

`guix system` takes a configuration file that declares the *complete* configuration of an operating system, along these lines:

```
(use-modules (gnu))
(use-service-modules networking ssh)
(use-package-modules certs screen)

(operating-system
  (host-name "gnu")
  (timezone "Europe/Berlin")
  (locale "en_US.utf8")
  (keyboard-layout (keyboard-layout "us" "altgr-intl"))

  (bootloader (bootloader-configuration
                (bootloader grub-efi-bootloader)
                (target (list "/boot/efi"))
                (keyboard-layout keyboard-layout)))
  (file-systems (cons (file-system
                        (device (file-system-label "my-root"))
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))

  (users (cons (user-account
                (name "charlie")
                (comment "Charlie Smith")
                (group "users")
                (supplementary-groups '("wheel"
                                        "audio" "video")))
               %base-user-accounts))

  ;; Globally installed packages.
  (packages (append (list screen nss-certs)
                    %base-packages))

  ;; System services: add sshd and DHCP to the base services.
  (services (append (list (service dhcp-client-service-type)
                          (service openssh-service-type
                                   (openssh-configuration
                                    (port-number 2222))))
                    %base-services)))
```

Building Operating Systems

`guix system image` *file*
create a raw disk image for the OS declared in *file*

`guix system image --image-type=iso9660` *file*
create an ISO CD/DVD image for the OS declared in *file*

`guix system image --image-type=qcow2` *file*
produce a QCOW2 image of the OS in *file*

`guix system vm` *file*
produce a script that runs the OS declared in *file* in a VM

Managing the Operating System

`guix system search` *regexp*
search for services matching *regexp*

`guix system reconfigure` *file*
reconfigure the OS according to the configuration in *file*

`guix system list-generations` [*pattern*]
list OS generations matching *pattern*—e.g., `1m` for one month

`guix system roll-back`
roll back to the previous system generation

`guix system delete-generations` *pattern*
delete generations matching *pattern*

`guix system build` *file*
build the OS declared in *file*

Building and Running Containers

`guix system container` *file*
produce a script that runs the OS declared in *file* in a container

`guix system image -t docker` *file*
build a Docker image of the OS declared in *file*

Inspecting an Operating System

`guix system extension-graph` *file*
show the graph of services extensions for the OS in *file*

`guix system shepherd-graph` *file*
show the dependency graph of Shepherd services for *file*

Copyright © 2018–2020, 2026 Ludovic Courtès <*ludo@gnu.org*>
Copyright © 2022 Ricardo Wurmus <*rekado@elephly.net*>
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at *https://gnu.org/licenses/gfdl.html*.
The source of this document is available from *https://git.sv.gnu.org/cgit/guix/maintenance.git*.

Guix