Bootstrappable Bitcoin Core Builds

This directory contains the files necessary to perform bootstrappable Bitcoin Core builds.

Bootstrappability furthers our binary security guarantees by allowing us to *audit and reproduce* our toolchain instead of blindly *trusting* binary downloads.

We achieve bootstrappability by using Guix as a functional package manager.

Requirements

Conservatively, you will need an x86_64 machine with:

- 16GB of free disk space on the partition that /gnu/store will reside in
- 8GB of free disk space **per platform triple** you're planning on building (see the `HOSTS` environment variable description)

Installation and Setup

If you don't have Guix installed and set up, please follow the instructions in INSTALL.md

Usage

If you haven't considered your security model yet, please read the relevant section before proceeding to perform a build.

Making the Xcode SDK available for macOS cross-compilation

In order to perform a build for macOS (which is included in the default set of platform triples to build), you'll need to extract the macOS SDK tarball using tools found in the `macdeploy` directory.

You can then either point to the SDK using the `SDK_PATH` environment variable:

```
# Extract the SDK tarball to /path/to/parent/dir/of/extracted/SDK/Xcode-<foo>-<bar>-extracte
tar -C /path/to/parent/dir/of/extracted/SDK -xaf /path/to/Xcode-<foo>-<bar>-extracted-SDK-wi

# Indicate where to locate the SDK tarball
export SDK_PATH=/path/to/parent/dir/of/extracted/SDK
```

or extract it into `depends/SDKs`:

```
mkdir -p depends/SDKs
tar -C depends/SDKs -xaf /path/to/SDK/tarball
```

Building

The author highly recommends at least reading over the common usage patterns and examples section below before starting a build. For a full list of customization options, see the recognized environment variables section.

To build Bitcoin Core reproducibly with all default options, invoke the following from the top of a clean repository:

```
./contrib/guix/guix-build
```

Codesigning build outputs

The `guix-codesign` command attaches codesignatures (produced by codesigners) to existing non-codesigned outputs. Please see the release process documentation for more context.

It respects many of the same environment variable flags as `guix-build`, with 2 crucial differences:

1. Since only Windows and macOS build outputs require codesigning, the `HOSTS` environment variable will have a sane default value of `x86_64-w64-mingw32 x86_64-apple-darwin arm64-apple-darwin` instead of all the platforms.
2. The `guix-codesign` command ***requires*** a `DETACHED_SIGS_REPO` flag.
   - ***DETACHED_SIGS_REPO***
     Set the directory where detached codesignatures can be found for the current Bitcoin Core version being built.
     REQUIRED environment variable

An invocation with all default options would look like:

```
env DETACHED_SIGS_REPO=<path/to/bitcoin-detached-sigs> ./contrib/guix/guix-codesign
```

Cleaning intermediate work directories

By default, `guix-build` leaves all intermediate files or "work directories" (e.g. `depends/work`, `guix-build-*/distsrc-*`) intact at the end of a build so that they are available to the user (to aid in debugging, etc.). However, these directories usually take up a large amount of disk space. Therefore, a `guix-clean` convenience script is provided which cleans the current `git` worktree to save disk space:

```
./contrib/guix/guix-clean
```

Attesting to build outputs

Much like how Gitian build outputs are attested to in a `gitian.sigs` repository, Guix build outputs are attested to in the `guix.sigs` repository.

After you've cloned the `guix.sigs` repository, to attest to the current worktree's commit/tag:

```
env GUIX_SIGS_REPO=<path/to/guix.sigs> SIGNER=<gpg-key-name> ./contrib/guix/guix-attest
```

See `./contrib/guix/guix-attest --help` for more information on the various ways `guix-attest` can be invoked.

Verifying build output attestations

After at least one other signer has uploaded their signatures to the `guix.sigs` repository:

```
git -C <path/to/guix.sigs> pull
env GUIX_SIGS_REPO=<path/to/guix.sigs> ./contrib/guix/guix-verify
```

Common `guix-build` invocation patterns and examples

Keeping caches and SDKs outside of the worktree

If you perform a lot of builds and have a bunch of worktrees, you may find it more efficient to keep the depends tree's download cache, build cache, and SDKs outside of the worktrees to avoid duplicate downloads and unnecessary builds. To help with this situation, the `guix-build` script honours the `SOURCES_PATH`, `BASE_CACHE`, and `SDK_PATH` environment variables and will pass them on to the depends tree so that you can do something like:

```
env SOURCES_PATH="$HOME/depends-SOURCES_PATH" BASE_CACHE="$HOME/depends-BASE_CACHE" SDK_PATH
```

Note that the paths that these environment variables point to **must be directories**, and **NOT symlinks to directories**.

See the recognized environment variables section for more details.

Building a subset of platform triples

Sometimes you only want to build a subset of the supported platform triples, in which case you can override the default list by setting the space-separated `HOSTS` environment variable:

```
env HOSTS='x86_64-w64-mingw32 x86_64-apple-darwin' ./contrib/guix/guix-build
```

See the recognized environment variables section for more details.

Controlling the number of threads used by `guix build` commands

Depending on your system's RAM capacity, you may want to decrease the number of threads used to decrease RAM usage or vice versa.

By default, the scripts under `./contrib/guix` will invoke all `guix` build commands with `--cores="$JOBS"`. Note that `$JOBS` defaults to `$(nproc)` if not

specified. However, astute manual readers will also notice that `guix` build commands also accept a `--max-jobs=` flag (which defaults to 1 if unspecified).

Here is the difference between `--cores=` and `--max-jobs=`:

> Note: When I say "derivation," think "package"

`--cores=`

- controls the number of CPU cores to build each derivation. This is the value passed to `make`'s `--jobs=` flag.

`--max-jobs=`

- controls how many derivations can be built in parallel
- defaults to 1

Therefore, the default is for `guix` build commands to build one derivation at a time, utilizing `$JOBS` threads.

Specifying the `$JOBS` environment variable will only modify `--cores=`, but you can also modify the value for `--max-jobs=` by specifying `$ADDITIONAL_GUIX_COMMON_FLAGS`. For example, if you have a LOT of memory, you may want to set:

```
export ADDITIONAL_GUIX_COMMON_FLAGS='--max-jobs=8'
```

Which allows for a maximum of 8 derivations to be built at the same time, each utilizing `$JOBS` threads.

Or, if you'd like to avoid spurious build failures caused by issues with parallelism within a single package, but would still like to build multiple packages when the dependency graph allows for it, you may want to try:

```
export JOBS=1 ADDITIONAL_GUIX_COMMON_FLAGS='--max-jobs=8'
```

See the recognized environment variables section for more details.

Recognized environment variables

- ***HOSTS***

  Override the space-separated list of platform triples for which to perform a bootstrappable build.

  (defaults to "x86_64-linux-gnu arm-linux-gnueabihf aarch64-linux-gnu riscv64-linux-gnu powerpc64-linux-gnu powerpc64le-linux-gnu x86_64-w64-mingw32 x86_64-apple-darwin arm64-apple-darwin")

- ***SOURCES_PATH***

  Set the depends tree download cache for sources. This is passed through to the depends tree. Setting this to the same directory across multiple builds

of the depends tree can eliminate unnecessary redownloading of package sources.

The path that this environment variable points to **must be a directory**, and **NOT a symlink to a directory**.

- ***BASE_CACHE***

  Set the depends tree cache for built packages. This is passed through to the depends tree. Setting this to the same directory across multiple builds of the depends tree can eliminate unnecessary building of packages.

  The path that this environment variable points to **must be a directory**, and **NOT a symlink to a directory**.

- ***SDK_PATH***

  Set the path where *extracted* SDKs can be found. This is passed through to the depends tree. Note that this is should be set to the *parent* directory of the actual SDK (e.g. `SDK_PATH=$HOME/Downloads/macOS-SDKs` instead of `$HOME/Downloads/macOS-SDKs/Xcode-12.2-12B45b-extracted-SDK-with-libcxx-headers`).

  The path that this environment variable points to **must be a directory**, and **NOT a symlink to a directory**.

- ***JOBS***

  Override the number of jobs to run simultaneously, you might want to do so on a memory-limited machine. This may be passed to:

  - `guix` build commands as in `guix environment --cores="$JOBS"`
  - `make` as in `make --jobs="$JOBS"`
  - `xargs` as in `xargs -P"$JOBS"`

  See here for more details.

  (defaults to the value of `nproc` outside the container)

- ***SOURCE_DATE_EPOCH***

  Override the reference UNIX timestamp used for bit-for-bit reproducibility, the variable name conforms to standard.

  (defaults to the output of `$(git log --format=%at -1)`)

- ***V***

  If non-empty, will pass `V=1` to all `make` invocations, making `make` output verbose.

  Note that any given value is ignored. The variable is only checked for emptiness. More concretely, this means that `V=` (setting `V` to the empty string) is interpreted the same way as not setting `V` at all, and that `V=0` has the same effect as `V=1`.

- ***SUBSTITUTE_URLS***

  A whitespace-delimited list of URLs from which to download pre-built packages. A URL is only used if its signing key is authorized (refer to the substitute servers section for more details).

- ***ADDITIONAL_GUIX_COMMON_FLAGS***

  Additional flags to be passed to all `guix` commands.

- ***ADDITIONAL_GUIX_TIMEMACHINE_FLAGS***

  Additional flags to be passed to `guix time-machine`.

- ***ADDITIONAL_GUIX_ENVIRONMENT_FLAGS***

  Additional flags to be passed to the invocation of `guix environment` inside `guix time-machine`.

Choosing your security model

No matter how you installed Guix, you need to decide on your security model for building packages with Guix.

Guix allows us to achieve better binary security by using our CPU time to build everything from scratch. However, it doesn't sacrifice user choice in pursuit of this: users can decide whether or not to use **substitutes** (pre-built packages).

Option 1: Building with substitutes

Step 1: Authorize the signing keys

Depending on the installation procedure you followed, you may have already authorized the Guix build farm key. In particular, the official shell installer script asks you if you want the key installed, and the debian distribution package authorized the key during installation.

You can check the current list of authorized keys at `/etc/guix/acl`.

At the time of writing, a `/etc/guix/acl` with just the Guix build farm key authorized looks something like:

```
(acl
 (entry
  (public-key
   (ecc
    (curve Ed25519)
    (q #8D156F295D24B0D9A86FA5741A840FF2D24F60F7B6C4134814AD55625971B394#)
    )
   )
  (tag
```

```
   (guix import)
   )
  )
 )
```

If you've determined that the official Guix build farm key hasn't been authorized, and you would like to authorize it, run the following as root:

```
guix archive --authorize < /var/guix/profiles/per-user/root/current-guix/share/guix/ci.guix
```

If `/var/guix/profiles/per-user/root/current-guix/share/guix/ci.guix.gnu.org.pub` doesn't exist, try:

```
guix archive --authorize < <PREFIX>/share/guix/ci.guix.gnu.org.pub
```

Where `<PREFIX>` is likely: - `/usr` if you installed from a distribution package - `/usr/local` if you installed Guix from source and didn't supply any prefix-modifying flags to Guix's `./configure`

For dongcarl's substitute server at https://guix.carldong.io, run as root:

```
wget -qO- 'https://guix.carldong.io/signing-key.pub' | guix archive --authorize
```

Removing authorized keys To remove previously authorized keys, simply edit `/etc/guix/acl` and remove the `(entry (public-key ...))` entry.

Step 2: Specify the substitute servers

Once its key is authorized, the official Guix build farm at https://ci.guix.gnu.org is automatically used unless the `--no-substitutes` flag is supplied. This default list of substitute servers is overridable both on a `guix-daemon` level and when you invoke `guix` commands. See examples below for the various ways of adding dongcarl's substitute server after having authorized his signing key.

Change the **default list** of substitute servers by starting `guix-daemon` with the `--substitute-urls` option (you will likely need to edit your init script):

```
guix-daemon <cmd> --substitute-urls='https://guix.carldong.io https://ci.guix.gnu.org'
```

Override the default list of substitute servers by passing the `--substitute-urls` option for invocations of `guix` commands:

```
guix <cmd> --substitute-urls='https://guix.carldong.io https://ci.guix.gnu.org'
```

For scripts under `./contrib/guix`, set the `SUBSTITUTE_URLS` environment variable:

```
export SUBSTITUTE_URLS='https://guix.carldong.io https://ci.guix.gnu.org'
```

Option 2: Disabling substitutes on an ad-hoc basis

If you prefer not to use any substitutes, make sure to supply `--no-substitutes` like in the following snippet. The first build will take a while, but the resulting packages will be cached for future builds.

For direct invocations of `guix`:

```
guix <cmd> --no-substitutes
```

For the scripts under `./contrib/guix/`:

```
export ADDITIONAL_GUIX_COMMON_FLAGS='--no-substitutes'
```

Option 3: Disabling substitutes by default

`guix-daemon` accepts a `--no-substitutes` flag, which will make sure that, unless otherwise overridden by a command line invocation, no substitutes will be used.

If you start `guix-daemon` using an init script, you can edit said script to supply this flag.

Purging/Uninstalling Guix

In the extraordinarily rare case where you messed up your Guix installation in an irreversible way, you may want to completely purge Guix from your system and start over.

1. Uninstall Guix itself according to the way you installed it (e.g. `sudo apt purge guix` for Ubuntu packaging, `sudo make uninstall` for a build from source).
2. Remove all build users and groups

   You may check for relevant users and groups using:

   ```
   getent passwd | grep guix
   getent group | grep guix
   ```

   Then, you may remove users and groups using:

   ```
   sudo userdel <user>
   sudo groupdel <group>
   ```
3. Remove all possible Guix-related directories
   - `/var/guix/`
   - `/var/log/guix/`
   - `/gnu/`
   - `/etc/guix/`
   - `/home/*/.config/guix/`

- /home/*/.cache/guix/
- /home/*/.guix-profile/
- /root/.config/guix/
- /root/.cache/guix/
- /root/.guix-profile/