Tribler Documentation

Release 8.0.0

Tribler

May 20, 2026

BASICS:

> **Note**
>
> Looking for user documentation? Check out https://www.tribler.org/howto.html

Tribler aims to give anonymous access to content. We are trying to make privacy, strong cryptography, and authentication the Internet norm.

These documents specify how to run Tribler from the latest source code and how to build Tribler distributions.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Running Tribler from Source

In order to run Tribler from its source you will need to perform some setup. We assume you have `git` and `python` installed. If you want to run a GUI for Tribler, you will need `npm` installed too.

1.1.1 Steps

1. Clone the Tribler repo:

```
git clone --recursive https://github.com/tribler/tribler
```

> **Warning**
>
> Tribler uses submodules. If you (1) download the ZIP or (2) forget to recursively clone, your `pyipv8` folder will be empty. Repair the former by downloading the IPv8 zip and extracting it in the `pyipv8` folder and repair the latter by running `git submodule update --init`.

2. Install the python dependencies:

```
python -m pip install --upgrade -r tribler/requirements.txt
```

3. Build the GUI:

```
cd src/tribler/ui/
npm install
npm run build
```

4. Add the IPv8 submodule to your `PYTHONPATH`. For example, (Windows) `set PYTHONPATH=%PYTHONPATH%;pyipv8`, (Linux) `export PYTHONPATH="${PYTHONPATH}:pyipv8"` or (PyCharm) right click the `pyipv8` folder and `Mark Directory as/Sources Root`.

5. Run Tribler:

```
cd src
python run_tribler.py
```

Sometimes, you may run into platform-specific issues. If this is your case, continue reading for your appropriate platform.

1.1.2 MacOS

You may need to install other packages separately:

```
brew install gmp mpfr libmpc libsodium
```

The security system on MacOS can prevent `libsodium.dylib` from being dynamically linked into Tribler when running Python. If this library cannot be loaded, it gives an error that libsodium could not be found. You can link or copy `libsodium.dylib` into the Tribler root directory:

```
cd tribler  # Wherever you have Tribler installed
cp /usr/local/lib/libsodium.dylib ./ || cp /opt/local/lib/libsodium.dylib ./
```

1.1.3 Apple Silicon

There are currently no python bindings available to install from pip. Therefore you need to build them from source.

To do this, please install openssl and boost first:

```
brew install openssl boost boost-build boost-python3
```

And then follow the instruction.

1.1.4 Windows

You may need to install the following packages separately:

- OpenSSL
- Libsodium

1.2 Running Tribler in Docker

In order to run Tribler from docker you only need a few steps. We assume you have `docker` installed.

Currently, only running the (unstable!) `latest` release is supported.

1.2.1 Preparation

Fetch the docker image from `docker.io`:

```
docker pull tribler/tribler:latest
```

Or, from `ghcr.io`:

```
docker pull ghcr.io/tribler/tribler:latest
```

1.2.2 Running

Choose a SECRET key for the background communication with Tribler. In the following example, we use the key `changeme` (don't use this yourself). To run the docker image:

```
docker run -e CORE_API_PORT=8085 -e CORE_API_KEY="changeme" \
           -e LANG=C.UTF-8 \
           -v ~/.Tribler:/home/ubuntu/.Tribler \
           -v ~/downloads/TriblerDownloads:/home/ubuntu/Downloads \
```

(continues on next page)

(continued from previous page)

```
           -v $XDG_CACHE_HOME/tmp/:$XDG_CACHE_HOME/tmp/ -v /run:/run \
           --security-opt "apparmor:unconfined" \
           --net="host" \
           -e XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR -e DISPLAY=$DISPLAY \
           -e XDG_CACHE_HOME=$XDG_CACHE_HOME -e XAUTHORITY=$XAUTHORITY \
           -e DBUS_SESSION_BUS_ADDRESS="$DBUS_SESSION_BUS_ADDRESS" \
           --user $(id -u):$(id -g) -e BROWSER="x-www-browser" \
           -it ghcr.io/tribler/tribler:latest
```

Alternatively, if you want to run *without opening the web GUI* and *without a tray icon*:

```
docker run -e CORE_API_PORT=8085 -e CORE_API_KEY="changeme" \
           -e LANG=C.UTF-8 \
           -v ~/.Tribler:/home/ubuntu/.Tribler \
           -v ~/downloads/TriblerDownloads:/home/ubuntu/Downloads \
           --net="host" -it ghcr.io/tribler/tribler:latest -s
```

You can then open Tribler in your web browser at the URL:

```
localhost:8085/ui/#/downloads/all?key=changeme
```

1.2.3 Notes

This script binds the local "state" directory (this is where Tribler puts its internal files) to `~/.Tribler` and your downloads directory (where your downloads end up) to `~/downloads/TriblerDownloads`. You can change these directories if you want to.

If you're planning on manipulating Tribler's internal REST API, you can access it through `http://localhost:8085/docs`. By default, the REST API is bound to localhost inside the container so to access the APIs, network needs to be set to host (–net="host").

1.2.4 Stopping

To stop Tribler, you should get the container id of your process and then stop it. You can view all active docker containers using `docker ps` and you can stop a container id using `docker stop`. For most UNIX systems, the following command will stop your Tribler Docker container:

```
docker stop $(docker ps -aqf ancestor="ghcr.io/tribler/tribler:latest")
```

1.3 Building on Linux

We assume you've set up your environment to run Tribler. Don't forget to build the GUI using NPM! Run the following commands in your terminal (assuming you are in the Tribler's repository root folder).

First, install additional requirements:

```
sudo apt-get -y install alien devscripts fakeroot gir1.2-gtk-3.0 libgirepository1.0-dev␣
↪rpm libcairo2-dev patchelf
python -m pip install --upgrade -r build/requirements.txt
```

Second, create the `.deb` file in the `dist` directory. You can set the `GITHUB_TAG` to whatever you want to have your version set as.

```
export GITHUB_TAG="1.2.3"

./build/debian/makedist_debian.sh
```

1.4 Building on Mac

We assume you've set up your environment to run Tribler. Don't forget to build the GUI using NPM! Run the following commands in your terminal (assuming you are in the Tribler's repository root folder).

First, install additional requirements:

```
python -m pip install -r build/requirements.txt
```

Second, create the `.dmg` file in the `dist` directory. You can set the `GITHUB_TAG` to whatever you want to have your version set as.

```
export GITHUB_TAG="1.2.3"

./build/mac/makedist_macos.sh
```

1.5 Building on Windows

We assume you've set up your environment to run Tribler. Don't forget to build the GUI using NPM! Additionally, you will need to install:

- `NSIS`, the `SimpleFC` plugin, and the `nsProcess` plugin.
- The latest `libsodium.dll` release.
- `Microsoft Visual Studio 2022 Enterprise`. `2022 Community` will also work, but you need to edit *tribler.nsi* in the appropriate place.
- `Windows Kits 10.0.26100.0`.
- `OpenSSL`.

Note

If you install any of these applications to non-default folders, you will need to modify the build scripts.

Run the following commands in your command prompt (assuming you are in the Tribler's repository root folder).

First, install additional requirements:

```
python -m pip install -r build/requirements.txt
```

Second, create the `.exe` file in the `dist` directory. You can set the `GITHUB_TAG` to whatever you want to have your version set as.

```
set GITHUB_TAG="1.2.3"

build\win\makedist_win.bat
```

1.6 Software Architecture

This reference provides a high-level overview of Tribler's software architecture.

The main entry point of Tribler is the `run_tribler.py` script and it has two main modes. The first mode launches Tribler with default settings. This causes a new browser tab to open that displays a web page generated with TypeScript. The second "headless" mode, does not open a browser tab. However, in both modes the core Tribler Python process runs.

An overview of the most important functional components of Tribler is given in the following graph. We will provide a short description of each of these components.

1.6.1 Session

The `Session` object (in `session.py`) manages the core managers of Tribler:

- The `Notifier` allows components to publish and subscribe to events. In some special cases, this is preferred over class inheritance or composition. Architectural spaghetti can sometimes be avoided by using `Notifier` events.
- The `TriblerConfigManager` manages writing to and reading from user configuration files. This class manages defaults and the regeneration of (partially) missing configurations.
- The `DownloadManager` allows management of torrent downloads. Functionality involves fetching torrent meta information, adding and removing downloads, and changing their anonymity level.
- The `RESTManager` sets up a REST API. The API is used by Tribler's GUI, but can also be used programmatically by users in "headless" mode.
- `IPv8` is used to manage peer-to-peer network overlays. All peer-to-peer traffic is handled by `IPv8` with the exception of non-anonymized torrent downloads. Functionality involves decentralized search and sharing popular torrents.
- The `IPv8CommunityLoader` is used to launch all optional parts of Tribler. This includes launching `IPv8` overlays and registering REST API endpoints.

The `TriblerDatabase`, `MetadataStore` and the `TorrentChecker` are optionally available through the `Session`. If requested, they are launched through the `IPv8CommunityLoader`.

1.6.2 Components

The optional components of Tribler are given in `components.py`. They can be divided into two groups: those that require a network overlay (inheriting from `ComponentLauncher`) and those that do not (inheriting from `ComponentLauncher`). Furthermore, the components can have complex relationships, like only launching after other components.

The following list contains a high-level description of the components:

- The `DatabaseComponent` is responsible for the two main databases. The first is the `MetadataStore`, which stores all torrent and torrent tracker meta data and "health" (seeders and leechers). The second is the `TriblerDatabase`, which stores torrent tags.
- The `TorrentCheckerComponent` is responsible for retrieving the "health" information of torrents and trackers. This component may contact trackers for random torrents to determine whether they are reachable. The component also periodically updates the health information for known torrents based on tracker and DHT information.
- The `VersioningComponent` is responsible for Tribler version management. Functionality includes upgrading data from local old Tribler versions to new Tribler versions and determining whether a new Tribler version has been published online.

- The `TunnelComponent` allows for anonymization of torrent downloads. This component manages routing downloads over Tor-like circuits, including circuit construction and destruction.
- The `DHTDiscoveryComponent` allows `IPv8` to decentrally search for overlays and peers with given public keys.
- The `ContentDiscoveryComment` serves remote user searches and content popularity. Both outgoing and incoming user searches are served through this component.
- The `KnowledgeComponent` handles extraction and search for torrent tags.
- The `RendezvousComponent` keeps track of peers that have been connected to in the past. This is an experimental component that aims to serve as a source of decentral reputation.
- The `RecommenderComponent` keeps track of local user searches and the preferred download for the search query. This is an experimental component that aims to serve as a source for Artificial Intelligence based personalized recommendations.

1.6.3 Web UI

The Web UI serves to view and control the state of the Tribler process. Essentially, the Web UI uses three main classes:

- The `IPv8Service` binds to the REST API of the `IPv8` component in the Tribler process.
- The `TriblerService` binds to the REST API of anything that is not in the `IPv8` service.
- The `App` is the React application entrypoint.

The `App` contains a component to route information, the `RouterProvider`, and a means to report errors that have been sent from the Tribler process. The architecture closely resembles the information in the GUI. Essentially, the `RouterProvider` mirrors the visible pages:

- `src/pages/Debug` queries the Tribler process for debug information.
- `src/pages/Downloads` includes components to render torrent downloads.
- `src/pages/Popular` queries for popular torrents.
- `src/pages/Search` manages pending searches, combining local and remote results.
- `src/pages/Settings` reads and writes the Tribler configuration (through the REST API).

CHAPTER

TWO

SEARCH

- genindex
- modindex
- search