GUFI Quickstart Guide

GUFI Developers

May 18, 2026

Contents

1 License

This file is part of GUFI, which is part of MarFS, which is released
under the BSD license.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

From Los Alamos National Security, LLC:
LA-CC-15-039

2 Quickstart

This guide assumes that the reader has some awareness of the purpose of GUFI, but does not know how to set it up or use it. This guide provides a full run through of the basics of building, installing, setting up, and running GUFI within a short period of time.

What this guide does not do is provide information on every single executable, flag, design decision, and implementation detail. For that, see the other documentation.

3 Download

Obtain a copy of the source code from GitHub:

```
git clone https://github.com/mar-file-system/GUFI.git
```

3.1 RPMs

Alternatively, three pre-built RPMs are attached to the assets of each release on GitHub. If proceeding with the RPMs, only download and install the RPM without `server` or `client` in its name. The Python3 library files required by the user facing tools (Section 6.2.2) will be installed into the site-packages directory.

Then, skip to Section 6.

Note that the RPMs on GitHub are specific to those releases/commits and do not get updated. To get the latest changes into RPM(s), configure and build GUFI from source. However, instead of running `make install`, install `rpmbuild`, rerun CMake so that `rpmbuild` is picked up, and run `make package`. The RPM(s) will be placed the build directory.

4 Dependencies

GUFI is known to build on Linux, macOS, and Windows (via cygwin). Most of the dependencies that GUFI requires are part of standard packages, such as `coreutils`, a C compiler, CMake, and bash. However, even these dependencies on different systems are known under different names. In `contrib/CI`, there are a series of shell scripts that can be run on the systems they are named for to download dependencies. Windows dependencies are listed in the GitHub Actions YAML file located in `.github/workflows/test.yml`.

5 Build and Install

5.1 Build Directory

Create a directory to configure and build GUFI. The recommended build directory name is `build` under the source tree root. It can be anywhere that is not

the source tree root itself.

```
mkdir -p build && cd build
```

5.2 Configure Build

GUFI uses CMake to generate Makefiles. The method of running CMake recommended here does not use the extensive list of flags available in `cmake(1)`:

```
cmake ..  <flags>
```

GUFI defines many CMake variables. Run `ccmake ..` to see the latest variables. Some particularly useful ones are listed here:

CMake Variable (prepend with `-D` to set)	Explanation
`CMAKE_INSTALL_PREFIX=<directory path>`	Where to install GUFI when running `make install`. Useful if not installing to system directories.
`DEP_AI=<ON|OFF>`	Enable or disable building sqlite-vec, sqlite-lembed, llama.cpp, and other AI related files. Note that OpenMP is left enabled in llama.cpp and thus required by GUFI when enabled.
`DEP_BUILD_THREADS=<thread count>`	Build dependencies with this many threads to speed up builds.
`DEP_INSTALL_PREFIX=<directory path>`	Place compiled bundled dependencies here. Used to keep the dependencies when the build directory is deleted.
`SERVER_CONFIG=<file path>`	Name of file used to configure user facing tools on the server side (explained later). A copy of `server.example` will be placed into the parent directory of the provided path. Copy/rename this file to the provided configuration file name. This is the GUFI configuration file that is used in this guide. There is an additional configuration file that is not necessary and not described here.

There are also a few CMake variables that are not normally modified, but are used in later text, and so are defined here:

CMake Variable (prepend with `-D` to set)	Explanation
`SERVER_BIN=<dir>`	Directory under `CMAKE_INSTALL_PREFIX` to place executables. (Default: `bin`)
`SERVER_LIB=<dir>`	Directory under `CMAKE_INSTALL_PREFIX` to place library files. (Default: `lib`)

Configuring CMake on macOS is trickier than building on Linux, especially with `DEP_AI=On`. See the latest macOS CMake configuration in `.github/workflows/test.yml`.

Configuring CMake on cygwin has a few caveats as well. See the latest cygwin CMake configuration in `.github/workflows/test.yml`.

5.3 Build

```
make
```

5.4 Install

```
(sudo) make install
```

Due to how GUFI was planned to be deployed, this installs what is known as the server side code. The server contains all of the actual implementations of GUFI functionality. The implied client side code does exist, but is not explained here.

At this point, the parent directory of `SERVER_CONFIG` should have a file called `server.example` that should be copied or renamed to `SERVER_CONFIG` and updated before using the user facing tools (Sec 6.2.2).

5.5 Environment Variables

If GUFI was installed, add `${CMAKE_INSTALL_PREFIX}/${SERVER_BIN}` to `PATH` and `${CMAKE_INSTALL_PREFIX}/${SERVER_LIB}` to both `LD_LIBRARY_PATH` and `PYTHONPATH` if they are not already in there.

Alternatively, if the files in the build directory are being used, the absolute path of `build/src` should be added to `PATH` and `LD_LIBRARY_PATH`. Similarly, the absolute path of `build/scripts` should be added to `PATH` and `PYTHONPATH`. Note that the server configuration file still needs to be found at `SERVER_CONFIG`.

Either way, **DO NOT** use the original files from the repository. Many are modified when they are placed in the build directory. **Always** use the files in the build or install directory.

6 Usage

With GUFI installed, usage is grouped into 2 phases. The first, indexing, is normally only done by an administrator. Querying can be done by both administrators and users, and has been split into 2 subsections (there is a third way, but has been intentionally left out here because it is not necessary to get GUFI running).

6.1 Phase 1: Index a Filesystem Tree

There are 4 executables that can be used to index a filesystem tree. Only `gufi_dir2index` is described here. The common inputs are

```
gufi_dir2index --threads <n> --index-xattrs tree GUFI_tree_parent
```

where `tree` is the root of a filesystem tree to be indexed. The basename of the directory path will be used to create the index. `GUFI_tree_parent` is the directory above where the index for this tree should be found. For example, passing in `tree=a/b/c` and `GUFI_tree_parent=search` will result in the index being generated in `search/c`. `GUFI_tree_parent` was designed this way so that all calls to `gufi_dir2index` can point to a unified index location.

6.1.1 Post-Indexing Optimizations

There are a few operations that can be done after indexing a tree to help increase search performance. Applying these operations is optional (there is one exception - see Section 6.2.2). The executables used to do them are listed here, but are not elaborated upon: `gufi_treesummary`, `gufi_treesummary_all`, and `gufi_rollup`.

6.2 Phase 2: Querying

Querying a GUFI tree is is done through either `gufi_query` or the user facing tools.

6.2.1 `gufi_query`

The primary command line tool for querying a GUFI tree is `gufi_query`. There are others, but this is the one to know about.

Using `gufi_query` requires knowledge of the internals of GUFI (such as the schema of GUFI databases), and thus only "advanced" users are expected to use it. The 3 primary flags used for passing in SQL, `-T`, `-S`, and `-E`, correspond to the tables/views that should be queried by them: the (optional) `treesummary` table, the (`directory`)`summary` table, and the `entries` table. Because the `treesummary` table is optional, it will not be elaborated on here. The (`directory`)`summary` table contains `stat(2)` information about the directory, as well as a series of columns that summarize the current directory, such as total number of files and the total size of all files. The `entries` table contains `stat(2)` information on the individual files and links.

`gufi_query` does **NOT** use `SERVER_CONFIG`.

There are several things to note that affect how `gufi_query` should be called (explainations can be found in the other LaTeXdocumentation).

- The `entries` table should not be used directly; instead, use the `vrpentries` view.
- The `summary` table should not be used directly; instead, use the `vrsummary` view.

- `-S` is run before `-E`, and will cause `-E` to not run if `-S` does not output at least one row of results (say, due to a `WHERE` clause).
- Nothing stops a user from querying an incorrect table from a flag e.g. querying `vrpentries` from `-S`.
- Many SQLite3 User Defined Functions have been defined for GUFI. The one to know is `rpath(sname, sroll, [entry name])` (not usable with `summary` and `entries`/`pentries`).

A simple `gufi_query` call might look something like:

```
gufi_query --threads <n> \
   -S "SELECT ...  FROM vrsummary ...  ;" \
   -E "SELECT ...  FROM vrpentries ...  ;" \
   GUFI_tree
```

where `GUFI_tree` can be any directory generated by indexing.

A basic `find(1)` call such as

```
find a/b/c
```

can be replicated in GUFI like so:

```
gufi_query --threads <n> \
   -S "SELECT rpath(sname, sroll) FROM vrsummary;" \
   -E "SELECT rpath(sname, sroll, name) FROM vrpentries;" \
   search/c
```

6.2.1.1 `gufi_vt` `gufi_vt` is a SQLite3 Run-Time Loadable Extension that wraps `gufi_query` and makes it accessible via the SQlite3 command line as well as via any language's SQLite3 library. It can be found in `build/src` and, if GUFI was installed, `${CMAKE_INSTALL_PREFIX}/${SERVER_LIB}` (`make install`) or `/usr/lib` (RPM).

6.2.2 User Facing Tools

Because `gufi_query` is very complex and requires detailed knowledge of the internals of GUFI, it is not the expected way most people will use to query indexes. They will instead use Python scripts that wrap `gufi_query` (or one of the other binaries) that partially replicate common filesystem search functionality with GUFI. The tools are: `gufi_du`, `gufi_find`, `gufi_getfattr`, `gufi_ls`, and `gufi_stat`. There is also another tool called `gufi_stats` that does not replicate the functionality of any standard tool.

Before running these tools, the server configuration must be set up. Look in the `SERVER_CONFIG` parent directory. There should be a file called `server.example`.

Copy this file to `SERVER_CONFIG` and modify any values that need changing. At minimum, `IndexRoot` will likely need to be changed to point to `GUFI_tree_parent`[1] or a subdirectory. An absolute path is recommended.

With the configuration file set up, the user facing tools will prefix every path passed to them with `IndexRoot`:

```
In the configuration file:
IndexRoot=search
... # other configuration values
```

Then, running

```
gufi_find c/d                                   # d is some path under c
```

will return all paths accessible by the caller under `search/c/d`. Note that `gufi_find` has the same quirk as `find(1)` where the input path(s) must come first in the argument list. However, flag ordering does not matter.

Similarly,

```
gufi_ls -al c/d
```

will print all directories, files, and links immediately under `search/c/d` formatted similarly to `ls(1) -al`.

Note that while all of the other user facing tools listed above will work immediately after indexing (and setting up the configuration file), `gufi_du` requires that the post-indexing optimization `gufi_treesummary_all` be applied to the index in order to function properly.

[1] These need to be renamed to the same variable name at some point.