Frequently Asked Questions

Build System and CMake Configuration

How do I add a new file to CMake?

If you forget to add a new file to the CMake configuration, you may end up with undefined symbol errors at link time.

There should be a `CMakeLists.txt` in the directory where you added the new file, which has the list of different .h/.cpp files to be included in the build. Add your new file to that list.

How do I speed up iterating on changes to the build system?

The general idea is to build as little as you can.

- Use `sccache` or `ccache` if you aren't doing so already.
- Use `build-script` 's various `--skip-*` flags to skip configuring for platforms that you do not care about.
- If you're on macOS, use `--swift-darwin-supported-archs="x86_64"` .
- Use a release build without assertions (`--release --no-assertions`). While debuginfo and assertions are valuable to enable when working on the toolchain itself, they are not so useful if you are working only on changes to the build system.

Using a Locally Built Toolchain

How do I use a locally built compiler to build X?

You can use the `SWIFT_EXEC` environment variable to use a locally built compiler to compile both packages and Xcode projects.

1. For SwiftPM packages, pass the environment variable when invoking SwiftPM.

```
# Assuming the current working directory contains the package, build the
# package using a custom compiler.
SWIFT_EXEC=/path/to/swiftc swift build
```

2. For Xcode projects, select the project in the Project Navigator. In the Build Settings tab, click '+' and then 'Add User-Defined Setting'. Create a build setting `SWIFT_EXEC` with the value set to `/path/to/swiftc` . If you now do a clean build, your locally built compiler will be used.

   At the time of writing, in the latest Xcode 12.2 beta 3, `SWIFT_EXEC` does not work for SwiftPM integration inside Xcode, so this will not work for Xcode projects that depend on SwiftPM packages.

Note: Even thought the variable says 'SWIFT', it needs to point to 'swift**c**', not 'swift'. The extra 'c' is not a typo.

Testing and CI

How do I reproduce/fix a test that fails in CI but passes for me locally?

TODO: Write some tips here, point to Testing.md for simulator setup.

Documentation

Where can I find documentation on X?

This very depends on what X is, but some broad guidelines are:

1. Do a case-insensitive recursive string search.
    - Use a specialized tool like ripgrep or ag.
    - Use `git grep --ignore-case "mypattern"` . `git grep` also supports helpful flags which provide more context:
        - `--show-function` : Tries to print the function name that a match was found in.
        - `--function-context` : Tries to print the entire surrounding function containing the match.
    - Use 'Find in Workspace' in Xcode (⌘+⇧+F).
    - Use `grep --ignore-case --recursive "mypattern" .` .
2. Go through the Documentation Index.

How do I build the documentation as HTML?

You can build the ReST formatted documentation as HTML using Sphinx. Follow Sphinx's installation instructions and check that `sphinx-build` is available on your `PATH` :

```
sphinx-build --version
```

If that succeeds, you can build the documentation using `make`

```
make -C docs
```

(Tested with `sphinx-build` version 3.2.1.)

This compiles the `.rst` files in the `docs` directory into HTML in the `docs/_build/html` directory.

For the Markdown documentation, you can view the rendered HTML directly on GitHub. For example, this file is rendered on GitHub at https://github.com/apple/swift/tree/main/docs/HowToGuides/FAQ.md .

HTML documentation for the standard library on Darwin platforms is hosted on the Apple Developer website.

Pull Request Workflow

How do I format my changes?

First, install `clang-format` using your system's package manager. This should also install the `git-clang-format` script (try `git-clang-format --help`). In case it doesn't, you can replace `git-clang-format` in the following commands with `../llvm-project/clang/tools/clang-format/git-clang-format` .

Start out at the tip of the branch where you want to reformat the commits.

```
# If there is only one commit that needs to be reformatted.
git-clang-format HEAD~1
git add .
git commit --amend --no-edit
```

```
# Say the last N commits need to be reformatted.
# Mark them as 'edit' instead of 'pick'.
git rebase -i HEAD~N
# Re-run N times, reformatting each commit.
git-clang-format HEAD~1
git add .
git commit --amend --no-edit
git rebase --continue
```

How do I clean up my git history?

TODO: Link to a beginner-friendly external resource, or (less preferably) describe basic usage of rebase here.