Swift

Swift Programming Language

	Architecture	main	Package
macOS	x86_64	Build Status	Build Status
Ubuntu 18.04	x86_64	Build Status	Build Status
Ubuntu 20.04	x86_64	Build Status	Build Status
CentOS 8	x86_64	Build Status	Build Status
CentOS 7	x86_64	Build Status	Build Status
Amazon Linux 2	x86_64	Build Status	Build Status

Swift Community-Hosted CI Platforms

OS	Architecture	Build
Ubuntu 16.04	PPC64LE	Build Status
Ubuntu 18.04	AArch64	Build Status
Ubuntu 20.04	AArch64	Build Status
Ubuntu 20.04	wasm32	Build Status
Android	ARMv7	Build Status
Android	AArch64	Build Status
Windows 2019 (VS 2017)	x86_64	Build Status

Windows 2019 (VS 2019)	x86_64	Build Status

Welcome to Swift

Swift is a high-performance system programming language. It has a clean and modern syntax, offers seamless access to existing C and Objective-C code and frameworks, and is memory safe by default.

Although inspired by Objective-C and many other languages, Swift is not itself a C-derived language. As a complete and independent language, Swift packages core features like flow control, data structures, and functions, with high-level constructs like objects, protocols, closures, and generics. Swift embraces modules, eliminating the need for headers and the code duplication they entail.

To learn more about the programming language, visit swift.org.

Contributing to Swift

Contributions to Swift are welcomed and encouraged! Please see the Contributing to Swift guide.

To be a truly great community, Swift.org needs to welcome developers from all walks of life, with different backgrounds, and with a wide range of experience. A diverse and friendly community will have more great ideas, more unique perspectives, and produce more great code. We will work diligently to make the Swift community welcoming to everyone.

To give clarity of what is expected of our members, Swift has adopted the code of conduct defined by the Contributor Covenant. This document is used across many open source communities, and we think it articulates our values well. For more, see the Code of Conduct.

Getting Started

If you are interested in:

- Contributing fixes and features to the compiler: See our How to Submit Your First Pull Request guide.
- Building the compiler as a one-off: See our Getting Started guide.
- Building a toolchain as a one-off: Follow the Getting Started guide up until the "Building the project" section. After that, follow the instructions in the Swift Toolchains section below.

We also have an FAQ that answers common questions.

Swift Toolchains

Building

Swift toolchains are created using the script build-toolchain. This script is used by swift.org's CI to produce snapshots and can allow for one to locally reproduce such builds for development or distribution purposes. A typical invocation looks like the following:

```
$ ./swift/utils/build-toolchain $BUNDLE_PREFIX
```

where `$BUNDLE_PREFIX` is a string that will be prepended to the build date to give the bundle identifier of the toolchain's `Info.plist`. For instance, if `$BUNDLE_PREFIX` was `com.example`, the toolchain produced will have the bundle identifier `com.example.YYYYMMDD`. It will be created in the directory you run the script with a filename of the form: `swift-LOCAL-YYYY-MM-DD-a-osx.tar.gz`.

Beyond building the toolchain, `build-toolchain` also supports the following (non-exhaustive) set of useful options::

- `--dry-run`: Perform a dry run build. This is off by default.
- `--test`: Test the toolchain after it has been compiled. This is off by default.
- `--distcc`: Use distcc to speed up the build by distributing the c++ part of the swift build. This is off by default.
- `--sccache`: Use sccache to speed up subsequent builds of the compiler by caching more c++ build artifacts. This is off by default.

More options may be added over time. Please pass `--help` to `build-toolchain` to see the full set of options.

Installing into Xcode

On macOS if one wants to install such a toolchain into Xcode:

1. Untar and copy the toolchain to one of `/Library/Developer/Toolchains/` or `~/Library/Developer/Toolchains/`. E.x.:

```
$ sudo tar -xzf swift-LOCAL-YYYY-MM-DD-a-osx.tar.gz -C /
$ tar -xzf swift-LOCAL-YYYY-MM-DD-a-osx.tar.gz -C ~/
```

The script also generates an archive containing debug symbols which can be installed over the main archive allowing symbolication of any compiler crashes.

```
$ sudo tar -xzf swift-LOCAL-YYYY-MM-DD-a-osx-symbols.tar.gz -C /
$ tar -xzf swift-LOCAL-YYYY-MM-DD-a-osx-symbols.tar.gz -C ~/
```

2. Specify the local toolchain for Xcode's use via `Xcode->Toolchains`.

Build Failures

Try the suggestions in Troubleshooting build issues.

Make sure you are using the correct release of Xcode.

If you have changed Xcode versions but still encounter errors that appear to be related to the Xcode version, try passing `--clean` to `build-script`.

When a new version of Xcode is released, you can update your build without recompiling the entire project by passing `--reconfigure` to `build-script`.

Learning More

Be sure to look at the documentation index for a bird's eye view of the available documentation. In particular, the documents titled Debugging the Swift Compiler and Continuous Integration for Swift are very helpful to understand before submitting your first PR.