Continuous Integration for Swift

Table of Contents

Introduction

This page is designed to assist in the understanding of proper practices for testing for the Swift project.

Pull Request Testing

In order for the Swift project to be able to advance quickly, it is important that we maintain a green build [1]. In order to help maintain this green build, the Swift project heavily uses pull request (PR) testing. Specifically, an important general rule is that **all** non-trivial checkins to any Swift Project repository should at least perform a smoke test if simulators will not be impacted *or* a full validation test if simulators may be impacted. If in addition one is attempting to make a source breaking change across multiple repositories, one should follow the cross repo source breaking changes workflow. We now continue by describing the Swift system for Pull Request testing, @swift-ci:

@swift-ci

Users with commit access can trigger pull request testing by writing a comment on a PR addressed to the GitHub user @swift-ci. Different tests will run depending on the specific comment used. The current test types are:

1. Smoke Testing
2. Validation Testing
3. Benchmarking.
4. Linting
5. Source Compatibility Testing
6. Specific Preset Testing
7. Testing Compiler Performance

We describe each in detail below:

Smoke Testing

Platform	Comment	Check Status
All supported platforms	@swift-ci Please smoke test	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test)
All supported platforms	@swift-ci Please clean smoke test	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test)
All supported platforms	@swift-ci Please smoke test and merge	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test)
All supported platforms	@swift-ci Please clean smoke test and merge	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test)
macOS platform	@swift-ci Please smoke test macOS platform	Swift Test macOS Platform (smoke test)
macOS platform	@swift-ci Please clean smoke test macOS platform	Swift Test macOS Platform (smoke test)
Linux platform	@swift-ci Please smoke test Linux platform	Swift Test Linux Platform (smoke test)
Linux platform	@swift-ci Please clean smoke test Linux platform	Swift Test Linux Platform (smoke test)

A smoke test on macOS does the following:

1. Builds LLVM/Clang incrementally.
2. Builds Swift clean.
3. Builds the standard library clean only for macOS. Simulator standard libraries and device standard libraries are not built.
4. lldb is not built.
5. The test and validation-test targets are run only for macOS. The optimized version of these tests are not run.

A smoke test on Linux does the following:

1. Builds LLVM/Clang incrementally.
2. Builds Swift clean.
3. Builds the standard library clean.
4. lldb is built incrementally.
5. Foundation, SwiftPM, LLBuild, XCTest are built.
6. The swift test and validation-test targets are run. The optimized version of these tests are not run.
7. lldb is tested.
8. Foundation, SwiftPM, LLBuild, XCTest are tested.

Validation Testing

Platform	Comment	Check Status
All supported platforms	@swift-ci Please test	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test) Swift Test Linux Platform Swift Test macOS Platform
All supported platforms	@swift-ci Please clean test	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test) Swift Test Linux Platform Swift Test macOS Platform
All supported platforms	@swift-ci Please test and merge	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test) Swift Test Linux Platform Swift Test macOS Platform
All supported platforms	@swift-ci Please clean test and merge	Swift Test Linux Platform (smoke test) Swift Test macOS Platform (smoke test) Swift Test Linux Platform Swift Test macOS Platform
macOS platform	@swift-ci Please test macOS platform	Swift Test macOS Platform (smoke test) Swift Test macOS Platform
macOS platform	@swift-ci Please clean test macOS platform	Swift Test macOS Platform (smoke test) Swift Test macOS Platform
macOS platform	@swift-ci Please benchmark	Swift Benchmark on macOS Platform (many runs - rigorous)
macOS platform	@swift-ci Please smoke benchmark	Swift Benchmark macOS Platform (few runs - sanity)
Linux platform	@swift-ci Please test Linux platform	Swift Test Linux Platform (smoke test) Swift Test Linux Platform
Linux platform	@swift-ci Please clean test Linux platform	Swift Test Linux Platform (smoke test) Swift Test Linux Platform
macOS platform	@swift-ci Please ASAN test	Swift ASAN Test macOS Platform
Ubuntu 18.04	@swift-ci Please test Ubuntu 18.04 platform	Swift Test Ubuntu 18.04 Platform
Ubuntu 20.04	@swift-ci Please test Ubuntu 20.04 platform	Swift Test Ubuntu 20.04 Platform
CentOS 7	@swift-ci Please test CentOS 7 platform	Swift Test CentOS 7 Platform
CentOS 8	@swift-ci Please test CentOS 8 platform	Swift Test CentOS 8 Platform
Amazon Linux 2	@swift-ci Please test Amazon Linux	Swift Test Amazon Linux 2 Platform

	2 platform	

The core principles of validation testing is that:

1. A validation test should build and run tests for /all/ platforms and all architectures supported by the CI.
2. A validation test should not be incremental. We want there to be a definitiveness to a validation test. If one uses a validation test, one should be sure that there is no nook or cranny in the code base that has not been tested.

With that being said, a validation test on macOS does the following:

1. Removes the workspace.
2. Builds the compiler.
3. Builds the standard library for macOS and the simulators for all platforms.
4. lldb is /not/ built/tested [2]
5. The tests, validation-tests are run for iOS simulator, watchOS simulator and macOS both with and without optimizations enabled.

A validation test on Linux does the following:

1. Removes the workspace.
2. Builds the compiler.
3. Builds the standard library.
4. lldb is built.
5. Builds Foundation, SwiftPM, LLBuild, XCTest
6. Run the swift test and validation-test targets with and without optimization.
7. lldb is tested.
8. Foundation, SwiftPM, LLBuild, XCTest are tested.

Benchmarking

Platform	Comment	Check Status
macOS platform	@swift-ci Please benchmark	Swift Benchmark on macOS Platform (many runs - rigorous)
macOS platform	@swift-ci Please smoke benchmark	Swift Benchmark on macOS Platform (few runs - sanity)

Linting

Language	Comment	What it Does	Corresponding Local Command
Python	@swift-ci Please Python lint	Lints Python sources	`./utils/python_lint.py`

Source Compatibility Testing

Platform	Comment	Check Status
macOS platform	@swift-ci Please Test Source Compatibility	Swift Source Compatibility Suite on macOS Platform (Release and Debug)
macOS platform	@swift-ci Please Test Source Compatibility Release	Swift Source Compatibility Suite on macOS Platform (Release)

macOS platform	@swift-ci Please Test Source Compatibility Debug	Swift Source Compatibility Suite on macOS Platform (Debug)

Sourcekit Stress Testing

Platform	Comment	Check Status
macOS platform	@swift-ci Please Sourcekit Stress test	Swift Sourcekit Stress Tester on macOS Platform

Build Swift Toolchain

Platform	Comment	Check Status
macOS platform	@swift-ci Please Build Toolchain macOS Platform	Swift Build Toolchain macOS Platform
Linux platform	@swift-ci Please Build Toolchain Linux Platform	Swift Build Toolchain Linux Platform

Build and Test Stdlib against Snapshot Toolchain

To test/build the stdlib for a branch that changes only the stdlib using a last known good snapshot toolchain:

Platform	Comment	Check Status
macOS platform	@swift-ci Please test stdlib with toolchain	Swift Test stdlib with toolchain macOS Platform

Specific Preset Testing

Platform	Comment	Check Status
macOS platform	preset= @swift-ci Please test with preset macOS Platform	Swift Test macOS Platform with preset
Linux platform	preset= @swift-ci Please test with preset Linux Platform	Swift Test Linux Platform with preset

For example:

```
preset=buildbot_incremental,tools=RA,stdlib=RD,smoketest=macosx,single-thread
@swift-ci Please test with preset macOS
```

Specific Preset Testing against a Snapshot Toolchain

One can also run an arbitrary preset against a snapshot toolchain

Platform	Comment	Check Status
macOS platform	preset= @swift-ci Please test with toolchain and preset	Swift Test stdlib with toolchain macOS Platform (Preset)

For example:

```
preset=$PRESET_NAME
@swift-ci Please test with toolchain and preset
```

Running Non-Executable Device Tests using Specific Preset Testing

Using the specific preset testing, one can run non-executable device tests by telling swift-ci:

```
preset=buildbot,tools=RA,stdlib=RD,test=non_executable
@swift-ci Please test with preset macOS
```

Build and Test the Minimal Freestanding Stdlib using Toolchain Specific Preset Testing

To test the minimal freestanding stdlib on macho, you can use the support for running a miscellaneous preset against a snapshot toolchain.

```
preset=stdlib_S_standalone_minimal_macho_x86_64,build,test
@swift-ci please test with toolchain and preset
```

Testing Compiler Performance

Platform	Comment	Check Status
macOS platform	@swift-ci Please test compiler performance	Compiles full source compatibility test suite and measures compiler performance
macOS platform	@swift-ci Please smoke test compiler performance	Compiles a subset of source compatibility test suite and measures compiler performance

These commands will:

1. Build a set of projects from the compatibility test suite
2. Collect counters and timers reported by the compiler
3. Compare the obtained data to the baseline (stored in git) and HEAD (version of a compiler built without the PR changes)
4. Report the results in a pull request comment

For the detailed explanation of how compiler performance is measured, please refer to this document.

Cross Repository Testing

Simply provide the URL from corresponding pull requests in the same comment as "@swift-ci Please test" phrase. List all of the pull requests and then provide the specific test phrase you would like to trigger. Currently, it will only merge the main pull request you requested testing from as opposed to all of the PR's.

For example:

```
Please test with following pull request:
https://github.com/apple/swift/pull/4574

@swift-ci Please test Linux platform
```

```
Please test with following PR:
https://github.com/apple/swift-lldb/pull/48
```

```
https://github.com/apple/swift-package-manager/pull/632

@swift-ci Please test macOS platform
```

```
apple/swift-lldb#48

@swift-ci Please test Linux platform
```

1. Create a separate PR for each repository that needs to be changed. Each should reference the main Swift PR and create a reference to all of the others from the main PR.

2. Gate all commits on @swift-ci smoke test and merge. As stated above, it is important that *all* checkins perform PR testing since if breakage enters the tree PR testing becomes less effective. If you have done local testing (using build-toolchain) and have made appropriate changes to the other repositories then perform a smoke test and merge should be sufficient for correctness. This is not meant to check for correctness in your commits, but rather to be sure that no one landed changes in other repositories or in swift that cause your PR to no longer be correct. If you were unable to make workarounds to the other repositories, this smoke test will break *after* Swift has built. Check the log to make sure that it is the expected failure for that platform/repository that coincides with the failure your PR is supposed to fix.

3. Merge all of the pull requests simultaneously.

4. Watch the public incremental build on ci.swift.org to make sure that you did not make any mistakes. It should complete within 30-40 minutes depending on what else was being committed in the mean time.

Swift Community Hosted CI Pull Request Testing

Currently, supported pull request testing triggers:

Platform	Comment	Check Status
Windows	@swift-ci Please test Windows platform	Swift Test Windows Platform
Linux	@swift-ci Please test Tensorflow Linux platform	Swift Test Linux Platform (TensorFlow)
Linux (GPU)	@swift-ci Please test Tensorflow Linux GPU platform	Swift Test Linux Platform with GPU (TensorFlow)
macOS	@swift-ci Please test Tensorflow macOS platform	Swift Test macOS Platform (TensorFlow)

ci.swift.org bots

FIXME: FILL ME IN!

[1] Even though it should be without saying, the reason why having a green build is important is that:

1. A full build break can prevent other developers from testing their work.
2. A test break can make it difficult for developers to know whether or not their specific commit has broken a test, requiring them to perform an initial clean build, wasting time.
3. @swift-ci pull request testing becomes less effective since one can not perform a test and merge and one must reason about the source of a given failure.

[2] This is due to unrelated issues relating to running lldb tests on macOS.