
swiftenv Documentation

Release 1.4.0

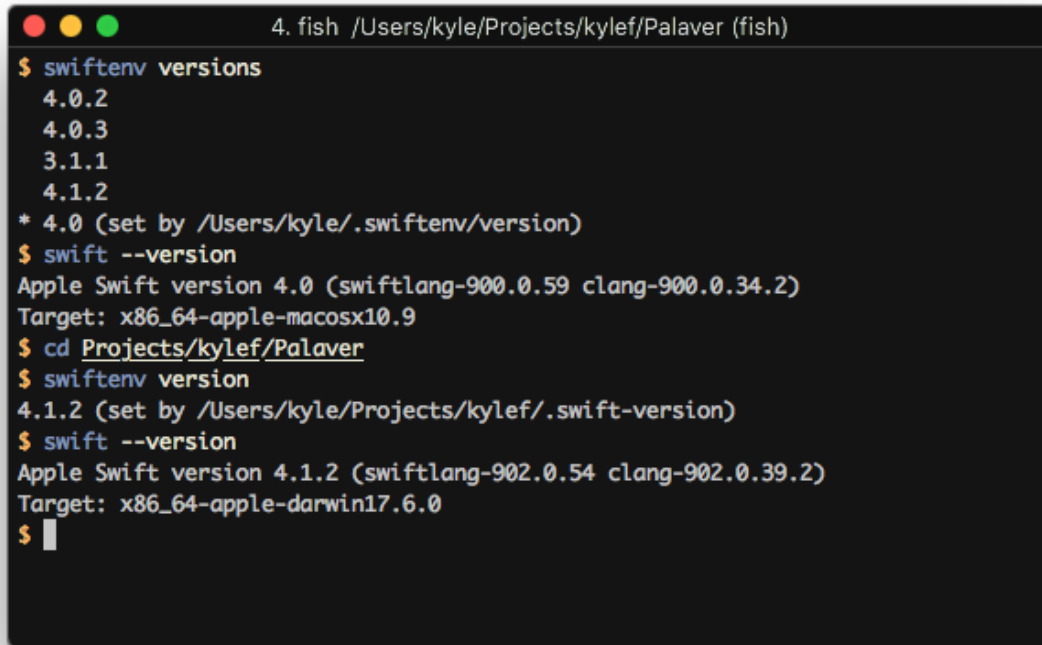
Kyle Fuller

Apr 14, 2020

Contents

1	The User Guide	3
1.1	Installation	3
1.2	Getting Started	4
1.3	Building Swift from Source	6
1.4	Command Reference	7
1.5	Changelog	9
1.6	Integrations	13

swiftenv allows you to easily install, and switch between multiple versions of Swift.



```

4. fish /Users/kyle/Projects/kylef/Palaver (fish)
$ swiftenv versions
4.0.2
4.0.3
3.1.1
4.1.2
* 4.0 (set by /Users/kyle/.swiftenv/version)
$ swift --version
Apple Swift version 4.0 (swiftlang-900.0.59 clang-900.0.34.2)
Target: x86_64-apple-macosx10.9
$ cd Projects/kylef/Palaver
$ swiftenv version
4.1.2 (set by /Users/kyle/Projects/kylef/.swift-version)
$ swift --version
Apple Swift version 4.1.2 (swiftlang-902.0.54 clang-902.0.39.2)
Target: x86_64-apple-darwin17.6.0
$

```

swiftenv allows you to:

- Change the **global Swift version**, per user.
- Set a **per-project Swift version**.
- Allows you to **override the Swift version** with an environmental variable.

1.1 Installation

NOTE: If you're on macOS, consider *installing with Homebrew*.

1.1.1 Via a Git clone

1. Check out swiftenv, we recommend ~/.swiftenv (but it can be installed elsewhere as long as you set SWIFTEENV_ROOT).

```
$ git clone https://github.com/kylef/swiftenv.git ~/.swiftenv
```

2. Configure environment.

For Bash:

```
$ echo 'export SWIFTEENV_ROOT="$HOME/.swiftenv"' >> ~/.bash_profile
$ echo 'export PATH="$SWIFTEENV_ROOT/bin:$PATH"' >> ~/.bash_profile
$ echo 'eval "$(swiftenv init -)"' >> ~/.bash_profile
```

NOTE: On some platforms, you may need to modify ~/.bashrc instead of ~/.bash_profile.

For ZSH:

```
$ echo 'export SWIFTEENV_ROOT="$HOME/.swiftenv"' >> ~/.zshenv
$ echo 'export PATH="$SWIFTEENV_ROOT/bin:$PATH"' >> ~/.zshenv
$ echo 'eval "$(swiftenv init -)"' >> ~/.zshenv
```

For Fish:

```
$ echo 'set -gx SWIFTEENV_ROOT "$HOME/.swiftenv"' >> ~/.config/fish/config.fish
$ echo 'set -gx PATH "$SWIFTEENV_ROOT/bin" $PATH' >> ~/.config/fish/config.fish
$ echo 'if which swiftenv > /dev/null; status --is-interactive; and source_
↪(swiftenv init -|psub); end' >> ~/.config/fish/config.fish
```

For other shells, please [open an issue](#) and we will visit adding support.

3. Restart your shell so the changes take effect.

1.1.2 Via Homebrew

You can install swiftenv using the [Homebrew](#) package manager on macOS.

1. Install swiftenv

```
$ brew install kylef/formulae/swiftenv
```

2. Then configure the shims and completions by adding the following to your profile.

For Bash:

```
$ echo 'if which swiftenv > /dev/null; then eval "$(swiftenv init -)"; fi' >> ~/.  
↪bash_profile
```

NOTE: *On some platforms, you may need to modify `~/.bashrc` instead of `~/.bash_profile`.*

For ZSH:

```
$ echo 'if which swiftenv > /dev/null; then eval "$(swiftenv init -)"; fi' >> ~/.  
↪zshrc
```

For Fish:

```
$ echo 'if which swiftenv > /dev/null; status --is-interactive; and source_  
↪(swiftenv init -|psub); end' >> ~/.config/fish/config.fish
```

1.1.3 Uninstalling swiftenv

1. Remove swiftenv from any `.bash_profile`, `.bashrc`, `.zshrc`, `fish.config` that you've added during installation.
2. Remove `SWIFTENV_ROOT` aka, `~/.swiftenv`.

```
$ rm -fr ~/.swiftenv
```

3. Uninstall any swiftenv packages (brew uninstall, pacman, etc).

1.2 Getting Started

Once you've [installed](#) swiftenv you can get started by checking which existing versions of Swift you have installed.

```
$ swiftenv versions  
2.2.1  
2.3  
* 3.0 (set by /Users/kyle/.swiftenv/version)
```

NOTE: *swiftenv will automatically pick up any versions of Swift installed on macOS by Xcode or Swift toolchains.*

1.2.1 Installing Swift

You can install swift using `swiftenv install`.

```
$ swiftenv install 3.0
```

Listing all versions

You can list all versions of Swift:

```
$ swiftenv install --list
```

You can also list all binary snapshots of Swift that you can install.

```
$ swiftenv install --list-snapshots
```

1.2.2 Switching Swift Versions

swiftenv allows you to switch between the installed Swift versions either globally or locally. You can configure a global Swift version that is used by default unless overridden.

Global Version

You can check the current global Swift version using `swiftenv global`.

```
$ swiftenv global
3.0
```

To change the global version:

```
$ swiftenv global 2.2.1
```

Local Version

You can override the global version within any project using a `.swift-version` file. A Swift version file will indicate the version to be used.

Setting the local Swift version:

```
$ swiftenv local 3.0
```

Now, when you're inside the current directory, the Swift version will be automatically changed to the local version.

```
$ swiftenv version
2.2.1 (set by /Users/kyle/Projects/kylef/Curassow/.swift-version)
```

When you switch to another directory without a `.swift-version` file, the global version will be used.

```
$ swiftenv version
3.0 (set by /Users/kyle/.swiftenv/version)
```

1.3 Building Swift from Source

`swiftenv install` can install Swift from source.

Listing available versions.

```
$ swiftenv install --list
2.2
2.2-dev
3.0-dev
```

NOTE: *Swift 2.2 does not include the Swift Package Manager.*

```
$ swiftenv install 2.2
```

By default, Swift will download from an Apple binary release available from swift.org. However you can use `--build` to force building the version.

```
$ swiftenv install 2.2 --build
```

1.3.1 Platforms

Below you can find a list of specific dependencies for each platform.

macOS

You will need to install the latest version of Xcode along with `cmake` and `ninja` build to build Swift on macOS.

Via Homebrew

```
$ brew install cmake ninja
```

Via Mac Ports

```
$ sudo port install cmake ninja
```

Arch Linux

You will need to install the following dependencies for Arch Linux:

```
$ pacman -S perl libbsd icu git libedit python2 clang cmake ninja
```

Ubuntu

You will need to install the following dependencies on Ubuntu:

```
$ sudo apt-get install git cmake ninja-build clang python uuid-dev libicu-dev icu-
↳ devtools libbsd-dev libedit-dev libxml2-dev libsqlite3-dev swig libpython-dev
↳ libncurses5-dev pkg-config
```

(continues on next page)

(continued from previous page)

If you are building on Ubuntu 14.04 LTS, you'll need to upgrade your clang compiler for C++14 support and create a symlink:

```
$ sudo apt-get install clang-3.6
$ sudo update-alternatives --install /usr/bin/clang clang /usr/bin/clang-3.6 100
$ sudo update-alternatives --install /usr/bin/clang++ clang++ /usr/bin/clang++-3.6 100
```

FreeBSD

You will need to install the following dependencies on FreeBSD:

```
$ pkg install binutils git python ninja cmake pkgconf e2fsprogs-libuuid
```

Your platform here

If you have successfully build Swift via swiftenv on other platforms, feel free to [update this list with a pull request](#).

1.4 Command Reference

1.4.1 version

Displays the current active Swift version and why it was chosen.

```
$ swiftenv version
2.2 (set by /home/kyle/.swiftenv/version)
```

1.4.2 versions

Lists all installed Swift versions, showing an asterisk next to the currently active version.

```
$ swiftenv versions
2.1.1
* 2.2 (set by /home/kyle/.swiftenv/version)
DEVELOPMENT-SNAPSHOT-2016-03-01-a
```

1.4.3 global

Sets the global version of Swift to be used by writing to the `~/.swiftenv/version` file. This version can be overridden by application-specific `.swift-version` file, or by setting the `SWIFT_VERSION` environment variable.

```
$ swiftenv global 2.2
$ swiftenv global
2.2
```

1.4.4 local

Sets the local application-specific Swift version by writing the version to a `.swift-version` file in the current directory. This version overrides the global version and can also be overridden further by the `SWIFT_VERSION` environment variable.

Setting a local Swift version

```
$ swiftenv local 3.1.1
```

Setting the local swift version will write the version to the `.swift-version` file in the current working directory.

Checking the local Swift version

```
$ swiftenv local
3.1.1
```

1.4.5 install

Installs a version of Swift. This supports both binary releases provided by Apple, along with all open source Swift releases.

You may use `--build` or `--no-build` to force a building from source, or installing from a binary release. Otherwise `swiftenv` will prefer installing from a binary release if available.

Please see [Building Swift from source](#) for more information.

```
$ swiftenv install 2.2
```

You may also install from a user supplied URLs to a Swift Binary package URL from [Swift Snapshots](#) as a parameter

Installing Swift from a URL

You may pass a URL of a binary Swift release directly to `swiftenv install`.

```
$ swiftenv install https://swift.org/builds/development/xcode/swift-DEVELOPMENT-
↳SNAPSHOT-2016-03-01-a/swift-DEVELOPMENT-SNAPSHOT-2016-03-01-a-osx.pkg
Downloading https://swift.org/builds/development/xcode/swift-DEVELOPMENT-SNAPSHOT-
↳2016-03-01-a/swift-DEVELOPMENT-SNAPSHOT-2016-03-01-a-osx.pkg
```

Custom Installation

You may also manually install Swift and make it accessible to `swiftenv`. Custom Swift installations can either be placed in a directory using the correct version number at `~/.swiftenv/versions/VERSION`, or can be symbolic linked into the version directory.

It is expected that all dependencies are already installed for running Swift, please consult the [Swift website](#) for more information.

NOTE: After manually installing a version of Swift, it's recommended that you run `swiftenv rehash` to update the shims.

Verifying Linux Binary Packages

When downloading a pre-built binary package, swiftenv can also download the corresponding signature and verify it with gpg. This option assumes gpg is installed on the system, and the [Swift public keys](#) already exist on the public gpg keyring. If verification fails, the version will not be installed. Signatures are currently only checked in this way for Linux builds.

```
$ swiftenv install 2.2 --verify
```

1.4.6 uninstall

Uninstalls a specific Swift version.

```
$ swiftenv uninstall 2.2
```

1.4.7 rehash

Installs shims for the Swift binaries. This command should be ran after you manually install new versions of Swift.

```
$ swiftenv rehash
```

1.4.8 which

Displays the full path to the executable that would be invoked for the selected version for the given command.

```
$ swiftenv which swift
/home/kyle/.swiftenv/versions/2.2/usr/bin/swift

$ swiftenv which lldb
/home/kyle/.swiftenv/versions/2.2/usr/bin/lldb
```

1.5 Changelog

1.5.1 1.4.0

Enhancements

- You can now instruct `swiftenv install` to both locally and globally set the installed swift version. `--set-local` and `--set-global` respectively will set the current Swift version.

The default behaviour will set the global version by default when `swiftenv install` was provided an explicit version. When installing with the `SWIFT_VERSION` environment value or the `.swift-version` file present, then the default behaviour is to not set the global or local version.

- When installing Swift from binary, swiftenv will now detect Ubuntu-based Linux distributions such as Elementary OS and use the appropriate binary image from swift.org.
- On macOS, `swiftenv install` now accepts `--user` command which allows you to install Swift into your home directory instead of requiring root.

Bug Fixes

- On macOS, `swiftenv uninstall` would fail to uninstall some installed binary toolchains due to `-RELEASE` being after the version in some paths that was unexpected.
- When using `swiftenv uninstall`, the command would fail if there was no global set version of swift. The command can now handle missing global version.

1.5.2 1.3.0

Enhancements

- New `--verify` option to `swiftenv install` to verify binary snapshots using GPG. This option expects that GPG is setup and configured to accept the Swift master keys. Verify can be forced with the environment variable `SWIFTENV_VERIFY`.
- Added local cache for Swift binaries for 3.0.2, 3.1, 3.1.1.
- `swiftenv install` will now resume any failed downloads instead of restarting the download process when restarting an install.
- `swiftenv install --verbose` will now include verbose build output while compiling swift.
- Adds build instructions for Swift 3.0, 3.0.1, 3.0.2, 3.1, 3.1.1, 3.0-dev, 3.1-dev and 4.0-dev.
- When building Swift from source, `swiftenv` will download tarballs instead of git cloning the repository resulting in faster download speed.
- Adds support for Fish 2.6.

1.5.3 1.2.1

Enhancements

- Adds usage and summaries when using `swiftenv --help` with a subcommand.
- Adds a manpage for `swiftenv` and `swiftenv-install`.

```
$ man swiftenv
$ man swiftenv-install
```

Bug Fixes

- Fixes detecting Swift release toolchains on macOS.
- Fixes an issue where `swiftenv install` wouldn't emit an error if it couldn't find instructions to install the given version.

1.5.4 1.2.0

Enhancements

- Only create shims for `swift*` and `lldb*` binaries found within Xcode installs. Before we created shims for all executable tools found in Xcode and created shims for tools like `ctags`, `cc`, `clang`, etc.

- Adds support for installing binary GM releases.

Bug Fixes

- Expose not found errors when using `swiftenv exec` against unknown commands.
- Swift preview versions such as `3.0-preview-1` will be detected as binary versions when using `swiftenv install`.

1.5.5 1.1.0

Enhancements

- Add a `--skip-existing/-s` flag to `swiftenv install` to skip installation if version is already installed.
- Adds support for Swift toolchains installed into `~/Library/Developer/Toolchains/` on OS X.

1.5.6 1.0.2

Bug Fixes

- Adds support for installing preview snapshots such as `3.0-preview-1-SNAPSHOT-2016-05-31-a`.
- `swiftenv init` will now cause a rehash if the version of `swiftenv` has changed.

1.5.7 1.0.1

Enhancements

- Added `swiftenv install --list-snapshots` which shows you a list of snapshots for your platform.

Bug Fixes

- Adds support for building Swift 2.2.1 from source, and installing 2.2.1 development snapshots.
- `swiftenv uninstall` will now uninstall Swift toolchains on OS X.
- `swiftenv uninstall` will now inform you if you're trying to uninstall a version of Swift bundled with Xcode.

1.5.8 1.0.0

Enhancements

- Supports installing final Swift releases such as `2.2`.

Bug Fixes

- Swift toolchains ‘latest’ version is no longer shown in `swiftenv versions` on OS X.
- Fixes a problem where `swiftenv install` on Linux will incorrectly determine URL for the Swift binaries.
- Adds a `--verbose` mode to `swiftenv versions` to show where the version was installed.

1.5.9 0.5.0

Enhancements

- The `swift-` prefix for versions is now optional.
- `swiftenv install` now has a `--list` option:

```
$ swiftenv install --list
```

- `swiftenv install` is capable of building Swift 2.2-dev from source.
- `swiftenv install` now takes URLs to a Swift binary package.
- `swiftenv install` was updated to use the new binary swift.org release URLs.

Bug Fixes

- Fixes an issue where using shims would suppress error messages when the configured version was not installed.
- Allows the completion to work when using `swiftenv` installed from Homebrew.

1.5.10 0.4.0

Enhancements

- Adds support for command and argument completions.

1.5.11 0.3.2

Bug Fixes

- Performance improvement when running on OS X. In previous versions, during initialisation `swiftenv` with rehash the environment, unfortunately once we added support for Xcode’s Swift there was a huge negative performance impact due to Xcode tools taking large amount of time due to the underlying commands being tremendously slow.

1.5.12 0.3.1

Bug Fixes

- Improved error reporting when trying to install a non-existent Swift version.
- When a shim command isn’t found in version, search `PATH` too. This fixes a problem when using `swiftenv` on OS X with Xcode installed while your Swift version is configured to a snapshot from swift.org.

1.5.13 0.3.0

Enhancements

- `swiftenv install` can now install Swift on OS X.

1.5.14 0.2.1

Bug Fixes

- Fixes an issue when installing via Homebrew and the `$SWIFTENV_ROOT` directory didn't exist.

1.5.15 0.2.0

Enhancements

- Adds support for versions of Swift included in Xcode.
- Added `swiftenv --help`.

1.5.16 0.1.0

Initial release.

1.6 Integrations

1.6.1 Heroku

The [Swift buildpack for Heroku](#) automatically makes use of `swiftenv` and will automatically install the local version of Swift you've specified in your `.swift-version` file.

Usage

Example usage:

```
$ ls
Procfile Package.swift Sources .swift-version

$ heroku create --buildpack https://github.com/kylef/heroku-buildpack-swift.git

$ git push heroku master
remote: ----> Swift app detected
remote: ----> Installing Swift DEVELOPMENT-SNAPSHOT-2016-02-08-a
remote: ----> Installing clang-3.7.0
remote: ----> Building Package
remote: ----> Copying binaries to 'bin'
```

You can also add it to upcoming builds of an existing application:

```
$ heroku buildpacks:set https://github.com/kylef/heroku-buildpack-swift.git
```

The buildpack will detect your app as Swift if it has a `Package.swift` file in the root.

Procfile

Using the Procfile, you can set the process to run for your web server. Any binaries built from your Swift source using swift package manager will be placed in your `$PATH`.

```
web: HelloWorld --workers 3 --bind 0.0.0.0:$PORT
```

1.6.2 Travis CI

You can use swiftenv to both install Swift, and to manage multiple versions of Swift on [Travis CI](#).

Using the following `install` phase, you can install both swiftenv and the Swift version found in the `.swift-version` file or the `SWIFT_VERSION` environment variable.

```
install:
- eval "$(curl -sL https://swiftenv.fuller.li/install.sh)"
```

Operating Systems

macOS

For macOS support on Travis, you will need to enable a version of Xcode which contains a support for the desired Swift version. See [Swift Documentation for Requirements on Apple platforms](#).

For example, Swift 5.1 requires macOS 10.14.6 and Xcode 11 or later:

```
osx_image: xcode11.3
```

Linux

For Linux, Travis needs to be configured to use a compatible version of Ubuntu and the surrounding tools.

Modern versions of Swift provided binary releases for Ubuntu 18.04 (also known as bionic). Use the `dist` key in Travis CI to enable the [Ubuntu 18.04 bionic runtime](#).

```
language: generic
sudo: required
dist: bionic
```

Multi-OS

swiftenv can be used on both macOS and Linux, you can use Travis [multiple operating system](#) support by adding both platforms to the `os` key:

```
os:
- linux
- osx
```

You can mix this together with the above steps required for macOS and Linux to have a complete `.travis.yml` file as follows:

```
os:
- linux
- osx
language: generic
sudo: required
dist: buster
osx_image: xcode11.3
install:
- eval "$(curl -sL https://swiftenv.fuller.li/install.sh)"
script:
- swift build
```

Testing against multiple Swift versions

You can use build matrix on Travis CI to set the `SWIFT_VERSION` environment variable to different values. Travis will now run against multiple versions of Swift.

```
env:
- SWIFT_VERSION=4.2
- SWIFT_VERSION=5.2.1
```

1.6.3 GitLab CI

Using the `swiftenv` docker image, you can install a version of Swift and test against it using GitLabs docker based CI runners.

This is what the `.gitlab-ci.yml` file looks like which can install `swiftenv` and Swift 3:

```
image: kylef/swiftenv

before_script:
- swiftenv install 3.0

test:
  script:
  - swift test
```

1.6.4 CircleCI

You can use `swiftenv` in conjunction with Docker on [CircleCI](#) to easily test your Swift project on CircleCI.

NOTE: *These instructions only cover using CircleCI on Linux and do not apply to CircleCI macOS containers.*

Dockerfile

A Dockerfile may contain the instructions to build a docker container containing swiftenv, Swift and your source code.

The following Dockerfile shows an example of setting up swiftenv and installing the version of Swift found in `.swift-version`.

You may also base your image on top of `kylef/swiftenv:swift3` or `kylef/swiftenv:swift` to use pre-installed Swift versions.

```
FROM kylef/swiftenv

RUN mkdir -p /code
WORKDIR /code
ADD . /code

RUN swiftenv install
```

circle.yml

Using a `circle.yml` file we can instruct CircleCI to build and run `swift test` inside our docker container.

```
machine:
  services: docker

dependencies:
  override: docker build -t myapp .

test:
  override: docker run myapp swift test
```

1.6.5 Docker

We provide a [swiftenv image for Docker](#). You can pull it down to use Swiftenv and Swift in Docker or base your own images from the swiftenv image.

Swiftenv provides multiple base docker images:

- `latest` - Image with swiftenv and all runtime dependencies to use Swift binaries.
- `build` - Image with swiftenv and all build dependencies to be able to build Swift from source.
- `swift3` - Image with swiftenv and the latest stable version of Swift 3.
- `swift` - Image with swiftenv and the latest stable version of Swift.

All of the docker images are based on top of Ubuntu 16.04 LTS (Xenial).

Running the swiftenv image directly

You can pull down the `kylef/swiftenv` docker image and run it.

```
$ docker pull kylef/swiftenv
$ docker run -i -t --entrypoint /bin/sh kylef/swiftenv
# swiftenv --version
swiftenv 1.4.0
```

Or for swiftenv with latest Swift:

```
$ docker pull kylef/swiftenv:swift
$ docker run -i -t --entrypoint /bin/sh kylef/swiftenv
# swift --version
swift 3.0.1
```

Building a docker image using swiftenv

You may base your own Docker image from the swiftenv image, you may then install any Swift version you desire in your container.

```
FROM kylef/swiftenv
RUN swiftenv install 3.0
```

docker-compose

Docker compose allows you to setup and run your project easier. It's a wrapper around `docker`.

For example, we can create a service called `commander` on top of the `swift3` swiftenv image which maps the source files into the docker container.

```
version: '2.0'

services:
  commander:
    image: kylef/swiftenv:swift3
    volumes:
      - './Sources:/code/Sources'
      - './Tests:/code/Tests'
      - './Packages:/code/Packages'
      - './Package.swift:/code/Package.swift'
    working_dir: /code
    command: swift build
```

We can then use `docker-compose` to run commands such as `swift test` inside our container.

```
$ docker-compose run commander swift test
```

You can switch out the `image` line of your service for `build: .` to build a `Dockerfile` found in your repository instead of going straight from the `swift3` image. This allows you to pin to a specific version of Swift.