# MacOS Deployment

The `macdeployqtplus` script should not be run manually. Instead, after building as usual:

```
make deploy
```

When complete, it will have produced `Bitcoin-Core.dmg`.

## SDK Extraction

### Step 1: Obtaining `Xcode.app`

A free Apple Developer Account is required to proceed.

Our current macOS SDK (`Xcode-12.2-12B45b-extracted-SDK-with-libcxx-headers.tar.gz`) can be extracted from Xcode_12.2.xip. Alternatively, after logging in to your account go to 'Downloads', then 'More' and search for `Xcode_12.2`. An Apple ID and cookies enabled for the hostname are needed to download this. The `sha256sum` of the archive should be `28d352f8c14a43d9b8a082ac6338dc173cb153f964c6e8fb6ba389e5be528bd0`.

After Xcode version 7.x, Apple started shipping the `Xcode.app` in a `.xip` archive. This makes the SDK less-trivial to extract on non-macOS machines. One approach (tested on Debian Buster) is outlined below:

```
# Install/clone tools needed for extracting Xcode.app
apt install cpio
git clone https://github.com/bitcoin-core/apple-sdk-tools.git

# Unpack Xcode_12.2.xip and place the resulting Xcode.app in your current
# working directory
python3 apple-sdk-tools/extract_xcode.py -f Xcode_12.2.xip | cpio -d -i
```

On macOS the process is more straightforward:

```
xip -x Xcode_12.2.xip
```

### Step 2: Generating `Xcode-12.2-12B45b-extracted-SDK-with-libcxx-headers.tar.gz` from `Xcode.app`

To generate `Xcode-12.2-12B45b-extracted-SDK-with-libcxx-headers.tar.gz`, run the script `gen-sdk` with the path to `Xcode.app` (extracted in the previous stage) as the first argument.

```
# Generate a Xcode-12.2-12B45b-extracted-SDK-with-libcxx-headers.tar.gz from
# the supplied Xcode.app
./contrib/macdeploy/gen-sdk '/path/to/Xcode.app'
```

## Deterministic macOS DMG Notes

Working macOS DMGs are created in Linux by combining a recent `clang`, the Apple `binutils` (`ld`, `ar`, etc) and DMG authoring tools.

Apple uses `clang` extensively for development and has upstreamed the necessary functionality so that a vanilla clang can take advantage. It supports the use of `-F`, `-target`, `-mmacosx-version-min`, and `-isysroot`, which are all necessary when building for macOS.

Apple's version of `binutils` (called `cctools`) contains lots of functionality missing in the FSF's `binutils`. In addition to extra linker options for frameworks and sysroots, several other tools are needed as well such as `install_name_tool`, `lipo`, and `nmedit`. These do not build under Linux, so they have been patched to do so. The work here was used as a starting point: mingwandroid/toolchain4.

In order to build a working toolchain, the following source packages are needed from Apple: `cctools`, `dyld`, and `ld64`.

These tools inject timestamps by default, which produce non-deterministic binaries. The `ZERO_AR_DATE` environment variable is used to disable that.

This version of `cctools` has been patched to use the current version of `clang`'s headers and its `libLTO.so` rather than those from `llvmgcc`, as it was originally done in `toolchain4`.

To complicate things further, all builds must target an Apple SDK. These SDKs are free to download, but not redistributable. See the SDK Extraction notes above for how to obtain it.

The Guix process builds 2 sets of files: Linux tools, then Apple binaries which are created using these tools. The build process has been designed to avoid including the SDK's files in Guix's outputs. All interim tarballs are fully deterministic and may be freely redistributed.

`xorrisofs` is used to create the DMG.

`xorrisofs` cannot compress DMGs, so afterwards, the DMG tool from the `libdmg-hfsplus` project is used to compress it. There are several bugs in this tool and its maintainer has seemingly abandoned the project.

The DMG tool has the ability to create DMGs from scratch as well, but this functionality is broken. Only the compression feature is currently used. Ideally, the creation could be fixed and `xorrisofs` would no longer be necessary.

Background images and other features can be added to DMG files by inserting a `.DS_Store` during creation.

As of OS X 10.9 Mavericks, using an Apple-blessed key to sign binaries is a requirement in order to satisfy the new Gatekeeper requirements. Because this private key cannot be shared, we'll have to be a bit creative in order for the build process to remain somewhat deterministic. Here's how it works:

- Builders use Guix to create an unsigned release. This outputs an unsigned DMG which users may choose to bless and run. It also outputs an unsigned app structure in the form of a tarball, which also contains all of the tools that have been previously (deterministically) built in order to create a final DMG.
- The Apple keyholder uses this unsigned app to create a detached signature, using the script that is also included there. Detached signatures are available from this repository.
- Builders feed the unsigned app + detached signature back into Guix. It uses the pre-built tools to recombine the pieces into a deterministic DMG.