macOS Build Guide

Updated for MacOS 11.2

This guide describes how to build bitcoind, command-line utilities, and GUI on macOS

Preparation

The commands in this guide should be executed in a Terminal application. macOS comes with a built-in Terminal located in:

`/Applications/Utilities/Terminal.app`

1. Xcode Command Line Tools

The Xcode Command Line Tools are a collection of build tools for macOS. These tools must be installed in order to build Bitcoin Core from source.

To install, run the following command from your terminal:

```
xcode-select --install
```

Upon running the command, you should see a popup appear. Click on `Install` to continue the installation process.

2. Homebrew Package Manager

Homebrew is a package manager for macOS that allows one to install packages from the command line easily. While several package managers are available for macOS, this guide will focus on Homebrew as it is the most popular. Since the examples in this guide which walk through the installation of a package will use Homebrew, it is recommended that you install it to follow along. Otherwise, you can adapt the commands to your package manager of choice.

To install the Homebrew package manager, see: https://brew.sh

Note: If you run into issues while installing Homebrew or pulling packages, refer to Homebrew's troubleshooting page.

3. Install Required Dependencies

The first step is to download the required dependencies. These dependencies represent the packages required to get a barebones installation up and running.

See dependencies.md for a complete overview.

To install, run the following from your terminal:

```
brew install automake libtool boost pkg-config libevent
```

4. Clone Bitcoin repository

`git` should already be installed by default on your system. Now that all the required dependencies are installed, let's clone the Bitcoin Core repository to a directory. All build scripts and commands will run from this directory.

```
git clone https://github.com/bitcoin/bitcoin.git
```

5. Install Optional Dependencies

Wallet Dependencies It is not necessary to build wallet functionality to run `bitcoind` or `bitcoin-qt`.

Descriptor Wallet Support

`sqlite` is required to support for descriptor wallets.

macOS ships with a useable `sqlite` package, meaning you don't need to install anything.

Legacy Wallet Support

`berkeley-db@4` is only required to support for legacy wallets. Skip if you don't intend to use legacy wallets.

```
brew install berkeley-db@4
```

GUI Dependencies Qt

Bitcoin Core includes a GUI built with the cross-platform Qt Framework. To compile the GUI, we need to install `qt@5`. Skip if you don't intend to use the GUI.

```
brew install qt@5
```

Ensure that the `qt@5` package is installed, not the `qt` package. If 'qt' is installed, the build process will fail. if installed, remove the `qt` package with the following command:

```
brew uninstall qt
```

Note: Building with Qt binaries downloaded from the Qt website is not officially supported. See the notes in #7714.

qrencode

The GUI can encode addresses in a QR Code. To build in QR support for the GUI, install `qrencode`. Skip if not using the GUI or don't want QR code functionality.

```
brew install qrencode
```

Port Mapping Dependencies miniupnpc

miniupnpc may be used for UPnP port mapping. Skip if you do not need this functionality.

```
brew install miniupnpc
```

libnatpmp

libnatpmp may be used for NAT-PMP port mapping. Skip if you do not need this functionality.

```
brew install libnatpmp
```

Note: UPnP and NAT-PMP support will be compiled in and disabled by default. Check out the further configuration section for more information.

ZMQ Dependencies Support for ZMQ notifications requires the following dependency. Skip if you do not need ZMQ functionality.

```
brew install zeromq
```

ZMQ is automatically compiled in and enabled if the dependency is detected. Check out the further configuration section for more information.

For more information on ZMQ, see: zmq.md

Test Suite Dependencies There is an included test suite that is useful for testing code changes when developing. To run the test suite (recommended), you will need to have Python 3 installed:

```
brew install python
```

Deploy Dependencies You can deploy a `.dmg` containing the Bitcoin Core application using `make deploy`. This command depends on a couple of python packages, so it is required that you have `python` installed.

Ensuring that `python` is installed, you can install the deploy dependencies by running the following commands in your terminal:

```
pip3 install ds_store mac_alias
```

Building Bitcoin Core

1. Configuration

There are many ways to configure Bitcoin Core, here are a few common examples:

Wallet (BDB + SQlite) Support, No GUI: If `berkeley-db@4` is installed, then legacy wallet support will be built. If `berkeley-db@4` is not installed, then this will throw an error. If `sqlite` is installed, then descriptor wallet support will also be built. Additionally, this explicitly disables the GUI.

```
./autogen.sh
./configure --with-gui=no
```

Wallet (only SQlite) and GUI Support: This explicitly enables the GUI and disables legacy wallet support. If `qt` is not installed, this will throw an error. If `sqlite` is installed then descriptor wallet functionality will be built. If `sqlite` is not installed, then wallet functionality will be disabled.

```
./autogen.sh
./configure --without-bdb --with-gui=yes
```

No Wallet or GUI

```
./autogen.sh
./configure --without-wallet --with-gui=no
```

Further Configuration You may want to dig deeper into the configuration options to achieve your desired behavior. Examine the output of the following command for a full list of configuration options:

```
./configure -help
```

2. Compile

After configuration, you are ready to compile. Run the following in your terminal to compile Bitcoin Core:

```
make        # use "-j N" here for N parallel jobs
make check  # Run tests if Python 3 is available
```

3. Deploy (optional)

You can also create a `.dmg` containing the `.app` bundle by running the following command:

```
make deploy
```

Running Bitcoin Core

Bitcoin Core should now be available at `./src/bitcoind`. If you compiled support for the GUI, it should be available at `./src/qt/bitcoin-qt`.

The first time you run `bitcoind` or `bitcoin-qt`, it will start downloading the blockchain. This process could take many hours, or even days on slower than average systems.

By default, blockchain and wallet data files will be stored in:

```
/Users/${USER}/Library/Application Support/Bitcoin/
```

Before running, you may create an empty configuration file:

```
mkdir -p "/Users/${USER}/Library/Application Support/Bitcoin"
```

```
touch "/Users/${USER}/Library/Application Support/Bitcoin/bitcoin.conf"
```

```
chmod 600 "/Users/${USER}/Library/Application Support/Bitcoin/bitcoin.conf"
```

You can monitor the download process by looking at the debug.log file:

```
tail -f $HOME/Library/Application\ Support/Bitcoin/debug.log
```

Other commands:

```
./src/bitcoind -daemon      # Starts the bitcoin daemon.
./src/bitcoin-cli --help    # Outputs a list of command-line options.
./src/bitcoin-cli help      # Outputs a list of RPC commands when the daemon is running.
./src/qt/bitcoin-qt -server # Starts the bitcoin-qt server mode, allows bitcoin-cli control
```