UNIX BUILD NOTES

Some notes on how to build Bitcoin Core in Unix.

(For BSD specific instructions, see `build-*bsd.md` in this directory.)

Note

Always use absolute paths to configure and compile Bitcoin Core and the dependencies. For example, when specifying the path of the dependency:

```
../dist/configure --enable-cxx --disable-shared --with-pic --prefix=$BDB_PREFIX
```

Here BDB_PREFIX must be an absolute path - it is defined using $(pwd) which ensures the usage of the absolute path.

To Build

```
./autogen.sh
./configure
make # use "-j N" for N parallel jobs
make install # optional
```

This will build bitcoin-qt as well, if the dependencies are met.

See dependencies.md for a complete overview.

Memory Requirements

C++ compilers are memory-hungry. It is recommended to have at least 1.5 GB of memory available when compiling Bitcoin Core. On systems with less, gcc can be tuned to conserve memory with additional CXXFLAGS:

```
./configure CXXFLAGS="--param ggc-min-expand=1 --param ggc-min-heapsize=32768"
```

Alternatively, or in addition, debugging information can be skipped for compilation. The default compile flags are `-g -O2`, and can be changed with:

```
./configure CXXFLAGS="-O2"
```

Finally, clang (often less resource hungry) can be used instead of gcc, which is used by default:

```
./configure CXX=clang++ CC=clang
```

Linux Distribution Specific Instructions

Ubuntu & Debian

Dependency Build Instructions Build requirements:

```
sudo apt-get install build-essential libtool autotools-dev automake pkg-config bsdmainutils
```

Now, you can either build from self-compiled depends or install the required dependencies:

```
sudo apt-get install libevent-dev libboost-dev
```

SQLite is required for the descriptor wallet:

```
sudo apt install libsqlite3-dev
```

Berkeley DB is required for the legacy wallet. Ubuntu and Debian have their own `libdb-dev` and `libdb++-dev` packages, but these will install Berkeley DB 5.1 or later. This will break binary wallet compatibility with the distributed executables, which are based on BerkeleyDB 4.8. If you do not care about wallet compatibility, pass `--with-incompatible-bdb` to configure. Otherwise, you can build Berkeley DB yourself.

To build Bitcoin Core without wallet, see *Disable-wallet mode*

Optional port mapping libraries (see: `--with-miniupnpc`, `--enable-upnp-default`, and `--with-natpmp`, `--enable-natpmp-default`):

```
sudo apt install libminiupnpc-dev libnatpmp-dev
```

ZMQ dependencies (provides ZMQ API):

```
sudo apt-get install libzmq3-dev
```

User-Space, Statically Defined Tracing (USDT) dependencies:

```
sudo apt install systemtap-sdt-dev
```

GUI dependencies:

If you want to build bitcoin-qt, make sure that the required packages for Qt development are installed. Qt 5 is necessary to build the GUI. To build without GUI pass `--without-gui`.

To build with Qt 5 you need the following:

```
sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5 qttools5-dev qttools5-dev-tools
```

Additionally, to support Wayland protocol for modern desktop environments:

```
sudo apt install qtwayland5
```

libqrencode (optional) can be installed with:

```
sudo apt-get install libqrencode-dev
```

Once these are installed, they will be found by configure and a bitcoin-qt executable will be built by default.

Fedora

Dependency Build Instructions Build requirements:

```
sudo dnf install gcc-c++ libtool make autoconf automake python3
```

Now, you can either build from self-compiled depends or install the required dependencies:

```
sudo dnf install libevent-devel boost-devel
```

SQLite is required for the descriptor wallet:

```
sudo dnf install sqlite-devel
```

Berkeley DB is required for the legacy wallet:

```
sudo dnf install libdb4-devel libdb4-cxx-devel
```

Newer Fedora releases, since Fedora 33, have only `libdb-devel` and `libdb-cxx-devel` packages, but these will install Berkeley DB 5.3 or later. This will break binary wallet compatibility with the distributed executables, which are based on Berkeley DB 4.8. If you do not care about wallet compatibility, pass `--with-incompatible-bdb` to configure. Otherwise, you can build Berkeley DB yourself.

To build Bitcoin Core without wallet, see *Disable-wallet mode*

Optional port mapping libraries (see: `--with-miniupnpc`, `--enable-upnp-default`, and `--with-natpmp`, `--enable-natpmp-default`):

```
sudo dnf install miniupnpc-devel libnatpmp-devel
```

ZMQ dependencies (provides ZMQ API):

```
sudo dnf install zeromq-devel
```

User-Space, Statically Defined Tracing (USDT) dependencies:

```
sudo dnf install systemtap
```

GUI dependencies:

If you want to build bitcoin-qt, make sure that the required packages for Qt development are installed. Qt 5 is necessary to build the GUI. To build without GUI pass `--without-gui`.

To build with Qt 5 you need the following:

```
sudo dnf install qt5-qttools-devel qt5-qtbase-devel
```

Additionally, to support Wayland protocol for modern desktop environments:

```
sudo dnf install qt5-qtwayland
```

libqrencode (optional) can be installed with:

```
sudo dnf install qrencode-devel
```

Once these are installed, they will be found by configure and a bitcoin-qt executable will be built by default.

Notes

The release is built with GCC and then "strip bitcoind" to strip the debug symbols, which reduces the executable size by about 90%.

miniupnpc

miniupnpc may be used for UPnP port mapping. It can be downloaded from here. UPnP support is compiled in and turned off by default. See the configure options for UPnP behavior desired:

```
--without-miniupnpc      No UPnP support, miniupnp not required
--disable-upnp-default   (the default) UPnP support turned off by default at runtime
--enable-upnp-default    UPnP support turned on by default at runtime
```

libnatpmp

libnatpmp may be used for NAT-PMP port mapping. It can be downloaded from here. NAT-PMP support is compiled in and turned off by default. See the configure options for NAT-PMP behavior desired:

```
--without-natpmp          No NAT-PMP support, libnatpmp not required
--disable-natpmp-default  (the default) NAT-PMP support turned off by default at runtime
--enable-natpmp-default   NAT-PMP support turned on by default at runtime
```

Berkeley DB

The legacy wallet uses Berkeley DB. To ensure backwards compatibility it is recommended to use Berkeley DB 4.8. If you have to build it yourself, you can use the installation script included in contrib/ like so:

```
./contrib/install_db4.sh `pwd`
```

from the root of the repository.

Otherwise, you can build Bitcoin Core from self-compiled depends.

Note: You only need Berkeley DB if the legacy wallet is enabled (see *Disable-wallet mode*).

Security

To help make your Bitcoin Core installation more secure by making certain attacks impossible to exploit even if a vulnerability is found, binaries are hardened by default. This can be disabled with:

Hardening Flags:

```
./configure --enable-hardening
./configure --disable-hardening
```

Hardening enables the following features: * *Position Independent Executable*: Build position independent code to take advantage of Address Space Layout Randomization offered by some kernels. Attackers who can cause execution of code at an arbitrary memory location are thwarted if they don't know where anything useful is located. The stack and heap are randomly located by default, but this allows the code section to be randomly located as well.

```
On an AMD64 processor where a library was not compiled with -fPIC, this will cause an error
such as: "relocation R_X86_64_32 against `......' can not be used when making a shared objec

To test that you have built PIE executable, install scanelf, part of paxutils, and use:

    scanelf -e ./bitcoin

The output should contain:

 TYPE
ET_DYN
```

- *Non-executable Stack*: If the stack is executable then trivial stack-based buffer overflow exploits are possible if vulnerable buffers are found. By default, Bitcoin Core should be built with a non-executable stack, but if one of the libraries it uses asks for an executable stack or someone makes a mistake and uses a compiler extension which requires an executable stack, it will silently build an executable without the non-executable stack protection.

  To verify that the stack is non-executable after compiling use: `scanelf -e ./bitcoin`

  The output should contain: STK/REL/PTL RW- R– RW-

  The STK RW- means that the stack is readable and writeable but not executable.

Disable-wallet mode

When the intention is to only run a P2P node, without a wallet, Bitcoin Core can be compiled in disable-wallet mode with:

```
./configure --disable-wallet
```

In this case there is no dependency on SQLite or Berkeley DB.

Mining is also possible in disable-wallet mode using the `getblocktemplate` RPC call.

Additional Configure Flags

A list of additional configure flags can be displayed with:

```
./configure --help
```

Setup and Build Example: Arch Linux

This example lists the steps necessary to setup and build a command line only, non-wallet distribution of the latest changes on Arch Linux:

```
pacman -S git base-devel boost libevent python
git clone https://github.com/bitcoin/bitcoin.git
cd bitcoin/
./autogen.sh
./configure --disable-wallet --without-gui --without-miniupnpc
make check
```

Note: Enabling wallet support requires either compiling against a Berkeley DB newer than 4.8 (package `db`) using `--with-incompatible-bdb`, or building and depending on a local version of Berkeley DB 4.8. The readily available Arch Linux packages are currently built using `--with-incompatible-bdb` according to the PKGBUILD. As mentioned above, when maintaining portability of the wallet between the standard Bitcoin Core distributions and independently built node software is desired, Berkeley DB 4.8 must be used.

ARM Cross-compilation

These steps can be performed on, for example, an Ubuntu VM. The depends system will also work on other Linux distributions, however the commands for installing the toolchain will be different.

Make sure you install the build requirements mentioned above. Then, install the toolchain and curl:

```
sudo apt-get install g++-arm-linux-gnueabihf curl
```

To build executables for ARM:

```
cd depends
make HOST=arm-linux-gnueabihf NO_QT=1
cd ..
./autogen.sh
CONFIG_SITE=$PWD/depends/arm-linux-gnueabihf/share/config.site ./configure --enable-reduce-e
make
```

For further documentation on the depends system see README.md in the depends directory.