# WINDOWS BUILD NOTES

Below are some notes on how to build Bitcoin Core for Windows.

The options known to work for building Bitcoin Core on Windows are:

- On Linux, using the Mingw-w64 cross compiler tool chain.
- On Windows, using Windows Subsystem for Linux (WSL) and Mingw-w64.
- On Windows, using Microsoft Visual Studio. See README.md.

Other options which may work, but which have not been extensively tested are (please contribute instructions):

- On Windows, using a POSIX compatibility layer application such as cygwin or msys2.

## Installing Windows Subsystem for Linux

Follow the upstream installation instructions, available here.

## Cross-compilation for Ubuntu and Windows Subsystem for Linux

The steps below can be performed on Ubuntu or WSL. The depends system will also work on other Linux distributions, however the commands for installing the toolchain will be different.

First, install the general dependencies:

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential libtool autotools-dev automake pkg-config bsdmainutils curl
```

A host toolchain (`build-essential`) is necessary because some dependency packages need to build host utilities that are used in the build process.

See dependencies.md for a complete overview.

If you want to build the windows installer with `make deploy` you need NSIS:

```
sudo apt install nsis
```

Acquire the source in the usual way:

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

## Building for 64-bit Windows

The first step is to install the mingw-w64 cross-compilation tool chain: - on modern systems (Ubuntu 21.04 Hirsute Hippo or newer, Debian 11 Bullseye or newer):

```
sudo apt install g++-mingw-w64-x86-64-posix
```

- on older systems:

```
sudo apt install g++-mingw-w64-x86-64
```

Once the toolchain is installed the build steps are common:

Note that for WSL the Bitcoin Core source path MUST be somewhere in the default mount file system, for example /usr/src/bitcoin, AND not under /mnt/d/. If this is not the case the dependency autoconf scripts will fail. This means you cannot use a directory that is located directly on the host Windows file system to perform the build.

Additional WSL Note: WSL support for launching Win32 applications results in `Autoconf` configure scripts being able to execute Windows Portable Executable files. This can cause unexpected behaviour during the build, such as Win32 error dialogs for missing libraries. The recommended approach is to temporarily disable WSL support for Win32 applications.

Build using:

```
PATH=$(echo "$PATH" | sed -e 's/:\/mnt.*//g') # strip out problematic Windows %PATH% importe
sudo bash -c "echo 0 > /proc/sys/fs/binfmt_misc/status" # Disable WSL support for Win32 appl
cd depends
make HOST=x86_64-w64-mingw32
cd ..
./autogen.sh
CONFIG_SITE=$PWD/depends/x86_64-w64-mingw32/share/config.site ./configure --prefix=/
make # use "-j N" for N parallel jobs
sudo bash -c "echo 1 > /proc/sys/fs/binfmt_misc/status" # Enable WSL support for Win32 appli
```

## Depends system

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

## Installation

After building using the Windows subsystem it can be useful to copy the compiled executables to a directory on the Windows drive in the same directory structure as they appear in the release `.zip` archive. This can be done in the following way. This will install to `c:\workspace\bitcoin`, for example:

```
make install DESTDIR=/mnt/c/workspace/bitcoin
```

You can also create an installer using:

```
make deploy
```