

Cross Platform Development for Raspberry

This page describes how to setup a cross platform build environment where the Debian distribution and version on build host and run-time Raspberry are identical. It will install and use the same ARM libraries (libc) and header version for building the armhf binary. I have used 2 different approaches to setup a cross build environment:

1. Debian Crossbuild packages
2. Manual setup

Debian's own cross-compilation documentation recommends `crossbuild-essential-<architecture>` as the normal way of installing the cross compiler.

GCC and Linux Architectures

GCC is also used to cross compile Linux applications. Applications can be compiled for 32-bit or 64-bit Linux systems.

Architecture	Compilers
32-bit	arm-linux-gnueabi-gcc arm-linux-gnueabi-g++
64-bit	aarch64-linux-gnu-gcc aarch64-linux-gnu-g++

Check the current target architecture by logging into the Pi: To check if system is running a 32 (Raspberry) or 64 (X86) architecture, use one of the following commands:

```
# getconf LONG_BIT
-----
32

file /lib/systemd/systemd
-----
/lib/systemd/systemd: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
(SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3,
BuildID[sha1]=1dc619f043235803cc3339a671c7f0821b733d93, for GNU/Linux 3.2.0,
stripped
```

Method 1: Debian Crossbuild packages

Build tools

Install all the required build tools on the Debian x86-64 host:

```
# apt-get update
# apt-get upgrade
# apt-get install build-essential
# apt-get install crossbuild-essential-armhf
```

crossbuild-essential-armhf installs the ARM cross C/C++ compiler and its basic runtime support. It does not install arbitrary target libraries such as OpenSSL, curl or ALSA development files.

multiarch mechanism

To setup dpkg for the armhf architecture run the `dpkg --add-architecture armhf` command. That command is needed when installing `:armhf` packages into the host's normal APT-managed filesystem. To setup dpkg for armhf architecture:

```
dpkg --add-architecture armhf
apt-get update
```

This enables Debian's **multiarch** mechanism. You can then install ARM libraries alongside your normal x86 libraries by `apt install libfoo-dev:armhf`, while retaining `libfoo-dev:amd64` on the same machine. Debian's cross-toolchains are specifically designed to work with these multiarch library/header locations. The ARM compiler knows where to find the ARM versions. For example:

```
/usr/lib/x86_64-linux-gnu/
/usr/include/...

/usr/lib/arm-linux-gnueabi/
/usr/include/...
```

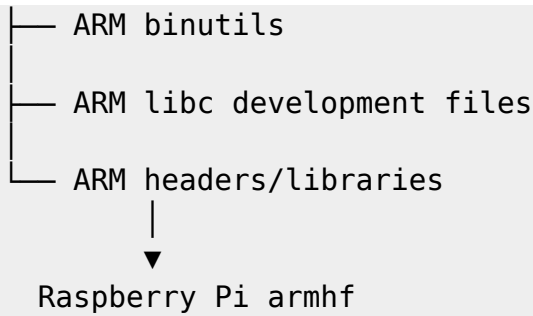
Additional Libraries

The standard `clib` libraries are already installed with the `crossbuild-essential-armhf` essential package. Installing additional development libraries for the new architecture:

```
apt-get install libssl-dev:armhf
apt-get install gnutls-dev:armhf
apt-get install libmicrohttpd-dev:armhf
apt-get install libgpiod-dev:armhf
```

This gives you a reasonably complete Debian cross-development environment looking like this:

```
x86-64 Debian
  |
  ├── native build tools
  |
  └── arm-linux-gnueabi-gcc
```



Method 2: Manual Setup

Set Up Cross Build Tools

The first step is to install the development tools on the desktop, or host system. From the command line run the following:

```
That will involve s
# apt-get update
# apt-get upgrade
# apt-get install build-essential
# apt-get install gcc-arm-linux-gnueabi
# apt-get install g++-arm-linux-gnueabi
# apt-get install gcc-aarch64-linux-gnu
# apt-get install g++-aarch64-linux-gnu
```

The first line makes sure that the system is up to date. The second instruction installs the general build tools. The third installs the C and C++ compiler and build tools for the Pi's ARM processor. The ARM architecture designation, arm-linux-gnueabi, is used as a prefix to distinguish the ARM tools from the host system tools. Note the dashThat will involve s at the end of the string. Test the installation by entering the command below. This reports the version of the G++ compiler installed and other information:

```
arm-linux-gnueabi-g++ -v
```

Get required ARM libraries

Depending on the application and required libraries, it is possible that not all libraries are available. Easiest way is to copy these over from an existing Raspberry installation. On the Raspberry these are available in: **/usr/lib/arm-linux-gnueabi/**. On the target build environment the library path is: **/usr/arm-linux-gnueabi/lib/**. We will be placing these retrieved libraries in a separate subdirectory (extra).

```
mkdir /usr/arm-linux-gnueabi/lib/extra
scp root@192.168.178.xx:/usr/lib/arm-linux-gnueabi/libxxx.* /usr/arm-
linux-gnueabi/lib/extra
```

Now we need to tell the loader (ld) to look for libraries in this extra directory

```
nano /etc/ld.so.conf.d/arm-linux-gnueabihf.conf
Insert the path "/usr/arm-linux-gnueabihf/lib/extra" into this file. Then
save and run:
ldconfig
```

Test Application

The host system should now be ready to build a Raspberry Pi program. Let's test it with a minimal test application, just to see if everything is working. Create the following test.cpp file:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "This is our first test!" << endl;
    return 0;
}
```

Build the program. The first line compiles the file test.cpp and the second links the compiler output to build the executable.

```
arm-linux-gnueabihf-g++ -O3 -g3 -Wall -c -fPIC -o test.o test.cpp
arm-linux-gnueabihf-g++ -o test test.o
```

Now let's check the architecture of the executable with the 'file' command. If this shows 'ARM' it is compiled for an ARM processor and can transfer it to the Pi.

```
file test
test: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0,
BuildID[sha1]=3a73dacec4a89fa2c6ea9751d18743ba9bba33ac, not stripped

scp test pi@192.168.178.xx:/tmp
```

From:
<https://wiki.oscardegroot.nl/> - HomeWiki

Permanent link:
<https://wiki.oscardegroot.nl/doku.php?id=raspberry:cross-platform-compile:setup-for-identical-debian-versions&rev=1788507309>

Last update: 2026/09/04 07:35

