

Cross Platform Development for different distributions

Introduction

This guide describes how to build 32-bit Raspberry binaries on a Debian x86-64 host where Debian distributions or versions differ between the build host and the runtime target. A cross-development environment has three separate parts:

```
Debian x86-64 host
|
|-- native build tools
|   |-- arm-linux-gnueabihf-gcc/g++
|   `-- CMake, Ninja, pkg-config [Optional]
|
`-- target sysroot
    |-- target C library and startup files
    |-- target headers:
    |   |-- /usr/include
    |-- target libraries:
    |   |-- /lib
    |   |-- /usr/lib
    `-- target pkg-config metadata
        |
        |--> executable for Raspberry Pi OS
```

The compiler runs on x86-64 but emits ARM instructions. The sysroot supplies the headers and libraries belonging to the target operating system. The host release and target release do not have to match. A Trixie host can build for Bookworm, provided the compiler is directed to a Bookworm sysroot.

sysroot

```
Debian host
├── crossbuild-essential-armhf
└── /opt/rpi-target-sysroot/
    ├── lib/
    ├── usr/
    │   ├── include/
    │   └── lib/
```

Compiling against a sysroot

To compile against a sysroot use the `--sysroot` directive as follows:

```
arm-linux-gnueabihf-gcc --sysroot=/opt/rpi-sysroot hello.c -o hello
```

Obtaining a sysroot

There are two options to populate the sysroot with library and include files that match the target distribution and version:

1. **Copy from target:** Clone the Headers/libraries from the **actual target Raspbian Pi** you're going to run the executable on. In this case the development packages should be installed on production Raspberry. Which is usually not preferred to install these dev-packages on a production system.
2. Use **mmdebstrap**: which offers an isolated APT configuration to: Select a target architecture and download corresponding .deb packages to extract their contents into a target filesystem tree.

Clone Sysroot from target

The sysroot could be populated from the **target Raspbian Pi** you're going to run the executable on, by copying header and library files from your actual Raspbian system. You can obtain it using `rsync`, for example:

```
rsync -a --delete root@raspberrypi:/lib/ /opt/rpi-sysroot/lib/  
rsync -a --delete root@raspberrypi:/usr/include/ /opt/rpi-  
sysroot/usr/include/  
rsync -a --delete root@raspberrypi:/usr/lib/ /opt/rpi-sysroot/usr/lib/
```

There are some important details around symbolic links, `/lib`, `/usr/lib`, exclusions such as `/usr/share`, and keeping the sysroot clean, so I wouldn't blindly use those commands as-is for a production setup.

Use mmdebstrap

`mmdebstrap` can use an isolated APT configuration to:

- Select a target architecture and release.
- Resolve requested packages and all hard dependencies.
- Download the corresponding .deb packages.
- Extract their contents into a target filesystem tree.

With `--variant=extract`, package maintainer scripts and ARM programs are not executed. A sysroot can therefore normally be created without QEMU and without adding `armhf` as a foreign architecture on the host.

Install host tools

On the Debian x86-64 host:

```
# apt update
# apt install \
    build-essential \
    crossbuild-essential-armhf \
    mmdebstrap \
    fakechroot \
    debian-archive-keyring \
    raspbian-archive-keyring \
    ca-certificates \
    curl \
    file \
    binutils
Optional:
# apt install \
    gnupg \
    cmake \
    ninja-build \
    pkg-config \
```

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. Since we are using a sysroot it is **not needed** to install:

```
# dpkg --add-architecture armhf
```

That command is needed when installing :armhf packages into the host's normal APT-managed filesystem. This guide instead keeps target packages isolated in a sysroot. Verify the compiler:

```
# arm-linux-gnueabi-gcc --version
# arm-linux-gnueabi-g++ --version
```

Distribution vs. Revision Strategy

Knowing the target distribution (e.g. Bookworm) is enough to select the correct release, but not necessarily the exact package revisions. A production Pi may not yet have the latest Bookworm security updates available from its repositories. Choose one of these policies:

1. **Current release policy:** Build against the latest packages currently available for the target release and keep production systems updated to that level.
2. **Pinned baseline policy:** Pin package versions or repository snapshots to a tested production baseline.

Inspect the target

The following commands only read information. Run them on one representative production Pi, directly or through SSH:

```
# cat /etc/os-release
# dpkg --print-architecture
# uname -m
# getconf GNU_LIBC_VERSION
```

Confirm that the expected result is similar to:

```
VERSION_CODENAME=bookworm
armhf
armv7l
glibc 2.36
```

Display the target's configured repositories:

```
# grep -RhsE '^[[:space:]]*(deb|Types:|URIs:|Suites:|Components:|Signed-By:)' \
    /etc/apt/sources.list \
    /etc/apt/sources.list.d/
```

Raspberry Pi OS 32-bit images normally use both:

1. A Raspbian base repository.
2. The Raspberry Pi archive for Raspberry Pi-specific packages.

Record installed runtime versions relevant to the application:

```
# dpkg-query -W -f='${binary:Package}=${Version}\n' libc6 libstdc++6 libssl3
2>/dev/null
```

For a complete read-only inventory:

```
# dpkg-query -W -f='${binary:Package}=${Version}\n' | sort > raspberry-pi-
installed-packages.txt
```

The output can be copied back to the development host.

Raspberry Pi Repositories

Raspberry Pi OS requires two separate repositories because it combines standard Linux software with custom hardware-specific code created by Raspberry Pi Ltd.

Feature	archive.raspberrypi.com/debian	raspbian.raspberrypi.com/raspbian
---------	---	---

Maintained By	Raspberry Pi Trading / Foundation	Raspbian Community (Independent project)
Purpose	Hardware-specific additions and Pi custom software.	Broad Linux operating system packages rebuilt for ARM architecture.
Typical Contents	VideoCore/GPU drivers, Linux kernels, raspi-config, custom desktop theme, bootloader updates, Pi-optimized software.	Core OS tools, standard Linux utilities, programming languages, web browsers, window managers, and standard server packages.
Repository Size	Small (hundreds of packages).	Massive (tens of thousands of packages).
Config Location	/etc/apt/sources.list.d/raspi.list	/etc/apt/sources.list
GPG key location	/usr/share/keyrings/ /etc/apt/trusted.gpg.d/	/usr/share/keyrings/ /etc/apt/trusted.gpg.d/

Look at [APT - Keyrings & Sources](#) for background information. Inspecting the target Raspberry reveals the following keyrings and sources configuration:

Keyrings

```
/etc/apt/keyrings/
  empty
/usr/share/keyrings/
  raspberrypi-archive-keyring.gpg
  raspberrypi-archive-removed-keys.gpg
  raspberrypi-archive-stable.gpg
  raspbian-archive-keyring.gpg
/etc/apt/trusted.gpg
  decoded with: gpg --no-default-keyring --keyring
/etc/apt/trusted.gpg --list-keys
  Mike Thompson (Raspberry Pi Debian armhf ARMv6+VFP)
<mptompson@gmail.com>
/etc/apt/trusted.gpg.d/
  mikethompson.gpg
  raspberrypi-archive-stable.gpg
```

APT Sources

```
/etc/apt/sources.list
  deb [ arch=armhf ] http://raspbian.raspberrypi.com/raspbian/ bookworm
  main contrib non-free rpi

/etc/apt/sources.list.d/raspi.list
  deb http://archive.raspberrypi.com/debian/ bookworm main
```

Prepare repository signing keys

The following two repositories are essential for the Raspberry Pi, and we need both to be setup on the build host as well:

- The Raspbian community repo provides the generic Linux software base.
- The Raspberry Pi repo provides the hardware tweaks, drivers, and Pi-specific software that make the system actually run on Raspberry Pi hardware.

We need to obtain the keyrings on the host system and store them at the `/etc/apt/keyrings/` location.

Get Raspbian repo archive GPG Key

To assure that the gpg key is not expired, download the gpg key directly from the Raspberry Pi archive. The Raspberry Pi archive currently provides 'raspberrypi-archive-keyring_2025.1+rpt1_all.deb'. To download this current keyring package do:

```
# cd /tmp

# wget
http://archive.raspberrypi.com/debian/pool/main/r/raspberrypi-archive-keyring/raspberrypi-archive-keyring_2025.1+rpt1_all.deb
```

Then extract it without installing anything:

```
dpkg-deb -x raspberrypi-archive-keyring_2025.1+rpt1_all.deb /tmp/rpi-keyring
```

Look at what it contains. You should get something like: `/usr/share/keyrings/raspberrypi-archive-keyring.gpg`

```
# find /tmp/rpi-keyring/usr/share/keyrings -type f -ls
```

Now inspect this key:

```
# gpg --show-keys --with-fingerprint /tmp/rpi-keyring/usr/share/keyrings/raspberrypi-archive-keyring.gpg
```

If that is the new keyring, you can install/copy it into your custom location:

```
# cp /tmp/rpi-keyring/usr/share/keyrings/raspberrypi-archive-keyring.gpg \
/etc/apt/keyrings/raspberrypi-archive-for-x-build.gpg
```

Get Raspberry Pi-specific archive GPG Key

Look at the contents of the Raspbian keyring directory. You should see an index containing files.

```
# wget -q0-  
https://raspbian.raspberrypi.com/raspbian/pool/main/r/raspbian-archive-keyring/  
-----  
-----  
...  
raspbian-archive-keyring_20120528.4_all.deb  
raspbian-archive-keyring_20120528.4.dsc  
...
```

The current package version is 20120528.4. Download the .deb. If exists remove the raspbian-keyring directory.

```
rm -rf /tmp/raspbian-keyring  
cd /tmp  
# wget  
https://raspbian.raspberrypi.com/raspbian/pool/main/r/raspbian-archive-keyring/  
raspbian-archive-keyring_20120528.4_all.deb
```

Extract it — don't install it. Just as we did with the Raspberry Pi keyring:

```
# dpkg-deb -x raspbian-archive-keyring_20120528.4_all.deb /tmp/raspbian-  
keyring
```

Now find the key, named similar like: 'raspbian-archive-keyring.gpg':

```
# find /tmp/raspbian-keyring -type f  
-----  
...  
/tmp/raspbian-keyring/usr/share/keyrings/raspbian-archive-keyring.gpg  
...
```

Inspect the key:

```
# gpg --show-keys --with-fingerprint /tmp/raspbian-  
keyring/usr/share/keyrings/raspbian-archive-keyring.gpg
```

You should see the Raspbian archive key:

```
A0DA 38D0 D76E 8B5D 6388 7281 9165 938D 90FD DD2E
```

You can install/copy it into your custom location:

```
sudo cp /tmp/raspbian-keyring/usr/share/keyrings/raspbian-archive-  
keyring.gpg \  
/etc/apt/keyrings/raspbian-archive-for-x-build.gpg
```

Check keys

Verify that on the host system both keyrings now exist:

```
# ls -l /etc/apt/keyrings
-----
/etc/apt/keyrings/raspberrypi-archive-for-x-build.gpg
/etc/apt/keyrings/raspbian-archive-for-x-build.gpg
```

Create Raspberry Pi OS source list

On the host we now create for APT a new source list, that included the raspberry repositories and include the right “signed-by”. Create `rpios-bookwork-x-cross-armhf.list`:

```
cat > /etc/apt/sources.list.d/rpios-bookwork-x-cross-armhf.list <<'EOF'
deb [arch=armhf signed-by=/etc/apt/keyrings/raspbian-archive-for-x-
build.gpg] http://raspbian.raspberrypi.com/raspbian/ bookworm main contrib
non-free rpi
deb [arch=armhf signed-by=/etc/apt/keyrings/raspberrypi-archive-for-x-
build.gpg] http://archive.raspberrypi.com/debian/ bookworm main
EOF
```

Compare these entries with the read-only repository inventory from the target. Use the production release's actual suites and components if they differ. Never substitute `stable` for `bookworm`: `stable` changes when Debian publishes a new release.

Choose target development packages

Every external library used by the application normally needs its `target -dev` package. Examples include:

Application dependency	Target development package
C and POSIX	<code>libc6-dev</code>
C++ standard library	<code>libstdc++-12-dev</code>
OpenSSL	<code>libssl-dev</code>
libcurl with OpenSSL	<code>libcurl4-openssl-dev</code>
ALSA	<code>libasound2-dev</code>
USB	<code>libusb-1.0-0-dev</code>
SQLite	<code>libsqlite3-dev</code>

Only request packages the project needs. A suitable example package set is:

```
TARGET_PACKAGES='libc6-dev,libstdc++-12-dev,libssl-dev,libcurl4-openssl-
dev,libasound2-dev'
```

Package names can vary between releases. Check availability without creating a sysroot by adding `--`

simulate to the command in the next section.

Resolve and download the sysroot

mmdebstrap (in unprivileged mode) uses unshare -user to create a user namespace where your user is remapped to “fake root” (UID 0) inside the namespace. It relies on mapping entries in /etc/subuid and /etc/subgid. If we just want to use it as normal user (oscar) add the --mode=fakeroot flag).

```
$ mmdebstrap \
  --simulate \
  --mode=fakeroot \
  --variant=extract \
  --architectures=armhf \
  --include="$TARGET_PACKAGES" \
  bookworm \
  ./sysroot
```

Review the selected package names, architectures and versions. They should be Bookworm armhf packages or architecture-independent all packages. Create now the sysroot:

```
$ mmdebstrap \
  --mode=fakeroot \
  --variant=extract \
  --architectures=armhf \
  --include="$TARGET_PACKAGES" \
  bookworm \
  ./sysroot
```

mmdebstrap downloads and extracts the requested packages and their hard dependencies. It does not modify the production Pi or install these packages into the host root filesystem.

Inspect the important paths:

```
$ find "$SYSROOT/usr/include" -maxdepth 2 -type d | head
$ find "$SYSROOT/usr/lib" -maxdepth 2 -type d | head
$ find "$SYSROOT/lib" -maxdepth 2 -type d | head
```

Confirm that target libraries are ARM objects:

```
$ file "$SYSROOT/lib/arm-linux-gnueabi/libc.so.6"
readelf -h "$SYSROOT/lib/arm-linux-gnueabi/libc.so.6" | \
  grep -E 'Class:|Machine:'
```

Expected characteristics include:

```
ELF 32-bit
Machine: ARM
```

Compile a direct GCC test

Create `hello.c`:

```
cat > hello.c <<'EOF'
#include <stdio.h>

int main(void)
{
    puts("Hello from ARM");
    return 0;
}
EOF
```

Compile against the sysroot:

```
arm-linux-gnueabi-hf-gcc \
    --sysroot="$SYSROOT" \
    -Wall \
    -Wextra \
    -O2 \
    hello.c \
    -o hello-armhf
```

Inspect the result:

```
file hello-armhf
readelf -h hello-armhf | grep -E 'Class:|Machine:'
readelf -d hello-armhf | grep NEEDED
readelf --version-info hello-armhf
```

The result should be a 32-bit ARM hard-float executable. Copy it to a test Pi and run it there before using the environment for production releases.

CPU compatibility warning

The `armhf` architecture identifies the 32-bit hard-float ABI, but it does not fully identify the CPU instruction baseline.

A standard Debian `arm-linux-gnueabi-hf` cross-toolchain normally assumes an ARMv7 baseline. This is appropriate for many Raspberry Pi 2 and newer systems. The original Raspberry Pi and Pi Zero/Zero W use ARMv6 and may reject an ARMv7 binary with `Illegal instruction`.

For ARMv6 targets, flags commonly include:

```
-march=armv6
-mfpu=vfp
```

```
-mfloat-abi=hard
```

Compiler flags alone are not enough: compiler runtime objects such as `libgcc` must also support ARMv6. Use a Raspberry Pi OS-compatible ARMv6 toolchain when Pi 1 or Pi Zero compatibility is required. Maintain a separate toolchain and sysroot profile for that target class.

For 64-bit Raspberry Pi OS, use a separate environment based on:

Architecture:	arm64
Compiler:	aarch64-linux-gnu-gcc
Debian meta-package:	crossbuild-essential-arm64
Separate sysroot directory:	/opt/sysroots/rpios-bookworm-arm64

Never combine armhf and arm64 files in one sysroot.

Links

- [Debian `mmdebstrap` manual](#)
- [Debian `mmdebstrap` manual](#)
- [Raspberry Pi OS documentation](#)
- [Raspberry Pi software sources](#)
- [Raspberry Pi package archive](#)

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

Permanent link:
<https://wiki.oscardegroot.nl/doku.php?id=raspberry:cross-platform-compile:setup-for-different-debian-versions&rev=1789289659>

Last update: 2026/09/13 08:54

