

Cross-Compiling Qt 5.15.0 for Raspberry Pi 4

This is a guide for Cross-compiling Qt 5.15 for a Raspberry Pi 4 running Ubuntu 20.04 Server from a host machine running Ubuntu 20.04.
Tested on with below prerequisite.

Hardware:

Host: 12th Gen Intel® Core™ i5-1235U × 12

Target: Raspberry Pi 4 Model B

Software:

Host: Ubuntu 20.04 LTS

Target: OS Ubuntu 20.04 Server

Cross Compiler: gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabi.tar.xz

It is assumed that you have a SD card with ubuntu 20.04 Server installed in your Raspberry Pi, otherwise download it and follow the installation guide.

we have downloaded the image and prepared the SD card on Ubuntu machine.

The following list summarizes the main steps to cross-compile Qt 5.12 for Raspberry Pi, we will be describing each of them in this guide.

1. Flash the image onto on SD card
2. Update the system with the following commands-[RPI4]
3. Install development libraries-[RPI4]
4. Prepare target folder -[RPI4]
5. Prepare Ubuntu Host Machine -[Host]
6. Create working folder download QT Resources and set a toolchain-[Host]
7. Create and configure a sysroot [Host]
8. Configure Qt for cross compilation -[Host]
9. Setup Qt Creator for Raspberry Pi cross compilation

1. Flash the image onto on SD card

To flash ubuntu 20.04 server os on SD card we used Raspberry Pi imager. On Raspberry Pi imager select required options operating system, storage device, settings to configure os.

<https://www.raspberrypi.com/software>

2. Update the system with the following commands-[RPI4]

```
$ sudo apt-get update && sudo apt-get upgrade
$ reboot
```

#configure wpa_supplicant:(optional if you configured on RPi imager)

Create or edit the /etc/wpa_supplicant/wpa_supplicant.conf file to configure your WiFi network. Replace your_network_name and your_network_password with your actual SSID (network name) and password.

```
$ sudo apt install wpasupplicant
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

```
network={
ssid="your_network_name"
psk="your_network_password"
}
```

```
$ sudo nano /etc/netplan/50-cloud-init.yaml
```

```
network:
  version: 2
  wifis:
    wlan0:
      access-points:
        "your_network_name":
          password: "your_network_password"
      dhcp4: true
      optional: true
```

```
$ sudo netplan apply
```

if no output then reboot the system

```
$ journalctl -xe | grep -iE "wpa_supplicant|network|wifi"
```

```
$ sudo systemctl start systemd-networkd
```

```
$ sudo systemctl status systemd-networkd
```

```
$ ping google.com
```

#Enable UART Communication

The UART should already be enabled by default on the Raspberry Pi 4.

To check the status of the UART hardware itself, you can use the

ls /dev command.

UART devices are typically named /dev/ttyS0 (UART0) and /dev/ttyAMA0 (UART1).

```
$ ls /dev/ttyS* /dev/ttyAMA*
```

If you see these devices listed, it means UART hardware is recognized by the system.

3. Install development libraries-[RPI4]

We need to install some development libraries, so the first thing to do is to allow the system to install source packages

```
$ sudo apt-get install -y qt5-qmake
```

```
$ sudo apt-get install -y libqt5webengine-data
```

```
$ sudo apt-get install libboost1.67-tools-dev libudev-dev libinput-dev libts-dev  
libmtdev-dev libjpeg-dev libfontconfig1-dev
```

```
$ sudo apt-get install libssl-dev libdbus-1-dev libglib2.0-dev libxkbcommon-dev  
libegl1-mesa-dev libgbm-dev libgles2-mesa-dev mesa-common-dev
```

```
$ sudo apt-get install libasound2-dev libpulse-dev gstreamer1.0-omx  
libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-alsa
```

```
$ sudo apt-get install libvpx-dev libsrtp2-dev libsnappy-dev libnss3-dev
```

```
$ sudo apt-get install "^libxcb.*"
```

```
$ sudo apt-get install flex bison libxslt-dev ruby gperf libbz2-dev libcups2-dev  
libatkmm-1.6-dev libxi6 libxcomposite1
```

```
$ sudo apt-get install libfreetype6-dev libicu-dev libsqlite3-dev libxslt1-dev  
libavcodec-dev libavformat-dev libswscale-dev
```

```
# Optional package for multimedia
```

```
$ sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev  
libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-  
plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly  
gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa  
gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
```

```
$ sudo apt-get install freetds-dev libsqlite0-dev libpq-dev libiodbc2-dev firebird-  
dev libjpeg9-dev libgst-dev libxext-dev libxcb1 libxcb1-dev libx11-xcb1
```

```
$ sudo apt-get install libx11-xcb-dev libxcb-keysyms1 libxcb-keysyms1-dev  
libxcb-image0 libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4  
libxcb-icccm4-dev
```

```
$ sudo apt-get install libxcb-sync1 libxcb-sync-dev libxcb-render-util0 libxcb-  
render-util0-dev libxcb-xfixes0-dev libxrender-dev libxcb-shape0-dev libxcb-  
randr0-dev
```

```
$ sudo apt-get install libxcb-glx0-dev libxi-dev libdrm-dev libssl-dev libxcb-  
xinerama0 libxcb-xinerama0-dev
```

```
$ sudo apt-get install libatspi2.0-0 libssl-dev libxcursor-dev libxcomposite-dev  
libxdamage-dev libfontconfig1-dev
```

```
$ sudo apt-get install libxss-dev libxtst-dev libpci-dev libcap-dev libsrtplib2-dev  
libxrandr-dev libnss3-dev libdirectfb-dev libaudio-dev
```

4. Prepare target folder -[RPI4]

This step just involves creating a folder in our Raspberry Pi for the pi user. This folder (/usr/local/qt5pi) will be used to deploy Qt from our computer to the Raspberry Pi.

```
$ sudo mkdir /usr/local/qt5pi  
$ sudo chown pi:pi /usr/local/qt5pi
```

5. Prepare Ubuntu Host Machine -[Host]

First of all, bring Ubuntu up to date and install some required libraries:
ping your target device

```
$ ping 192.168.16.25 (target board ip)  
$ sudo apt-get update && sudo apt-get upgrade  
$ sudo bash  
$ apt-get install gcc git bison python gperf pkg-config  
$ apt install make  
$ apt install libclang-dev  
$ apt install build-essential
```

6. Create working folder download QT Resources and set a toolchain-[Host]

```
$ mkdir /opt/qt5pi  
$ chown username:username/opt/qt5pi  
$ cd /opt/qt5pi/  
  
$ wget https://download.qt.io/official\_releases/qt/5.15/5.15.2/single/qt-everywhere-src-5.15.2.tar.xz  
$ tar xf qt-everywhere-src-5.15.2.tar.xz  
  
$ wget https://releases.linaro.org/components/toolchain/binaries/latest-7/arm-linux-gnueabi/hf/ gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabi/hf.tar.xz  
$ tar xf gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabi/hf.tar.xz
```

```
$ export PATH=$PATH:/opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin
$ nano ~/.bashrc
```

7.Create and configure a sysroot [Host]

```
$ mkdir sysroot sysroot/usr sysroot/opt
$ rsync -avz pi@192.168.1.2:/lib sysroot
$ rsync -avz pi@192.168.1.2:/usr/include sysroot/usr
$ rsync -avz pi@192.168.1.2:/usr/lib sysroot/usr
$ rsync -avz pi@192.168.1.2:/usr/lib/arm-linux-gnueabihf sysroot/opt

$ ls sysroot/usr/lib/arm-linux-gnueabihf/libEGL.so.1.1.0

$ mv sysroot/usr/lib/arm-linux-gnueabihf/libEGL.so.1.1.0 sysroot/usr/lib/arm-
linux-gnueabihf/libEGL.so.1.1.0_backup

$ ln -s sysroot/usr/lib/arm-linux-gnueabihf/libEGL.so sysroot/usr/lib/arm-linux-
gnueabihf/libEGL.so.1.1.0

$ mv sysroot/usr/lib/arm-linux-gnueabihf/libGLSv2.so.2.1.0
sysroot/usr/lib/arm-linux-gnueabihf/libGLSv2.so.2.1.0_backup
$ ln -s sysroot/usr/lib/arm-linux-gnueabihf/libGLSv2.so sysroot/usr/lib/arm-
linux-gnueabihf/libGLSv2.so.2.1.0

$ ln -s sysroot/usr/lib/arm-linux-gnueabihf/libEGL.so sysroot/usr/lib/arm-linux-
gnueabihf/libEGL.so.1
$ ln -s sysroot/usr/lib/arm-linux-gnueabihf/libGLSv2.so sysroot/usr/lib/arm-
linux-gnueabihf/libGLSv2.so.2
```

Next, we need to adjust our symbolic links in sysroot to be relative since this folder structure is in both our computer and Raspberry Pi.

```
$ wget https://raw.githubusercontent.com/riscv/riscv-poky/master/scripts/
sysroot-relativelinks.py
$ chmod +x sysroot-relativelinks.py
$ ./sysroot-relativelinks.py sysroot
```

```
$ rsync -avz pi@192.168.1.2:/lib sysroot
$ rsync -avz pi@192.168.1.2:/usr/include sysroot/usr
$ rsync -avz pi@192.168.1.2:/usr/lib sysroot/usr
$ rsync -avz pi@192.168.1.2:/usr/lib/arm-linux-gnueabihf sysroot/opt
$ ./sysroot-relativelinks.py sysroot
```

8. Configure Qt for cross compilation -[Host]

```
$ mkdir qt5build
$ cd qt5build/

$ ../qt-everywhere-src-5.15.2/configure -opengl es2 -device linux-rasp-pi4-v3d-
g++ -device-option CROSS_COMPILE=/opt/qt5pi/gcc-linaro-7.5.0-2019.12-
x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf- -sysroot
/opt/qt5pi/sysroot -prefix /usr/local/qt5pi -opensource -confirm-license -skip
qtscript -skip qtwayland -skip qtdataavis3d -nomake examples -make libs -pkg-
config -no-use-gold-linker -v
```

After a few minutes, the script should be finished and the following conditions should be set or not set.

QPA backends:

```
DirectFB ..... no
EGLFS ..... yes [
EGLFS details:
  EGLFS OpenWFD ..... no
  EGLFS i.Mx6 ..... no
  EGLFS i.Mx6 Wayland ..... no
  EGLFS RCAR ..... no
  EGLFS EGLDevice ..... yes
  EGLFS GBM ..... yes
  EGLFS VSP2 ..... no
```

```
EGLFS Mali ..... no
EGLFS Raspberry Pi ..... no
EGLFS X11 ..... yes
LinuxFB ..... yes
VNC ..... yes
```

The exact compilation time depends on your computer performance. This process can take up to 2 or 3 hours in nowadays common computers. Compiler processes can be executed in parallel by means of the `j` flag. For four make processes, execute: `make -j 8`.

```
$ make -j8
```

```
$ make install
```

Once Qt is compiled, it can be deployed to your Raspberry Pi using the `rsync` command.

```
$ cd /opt/qt5pi/
```

```
$ rsync -avz --rsync-path="sudo rsync" sysroot/usr/local/qt5pi
pi@192.168.16.25:/usr/local
```

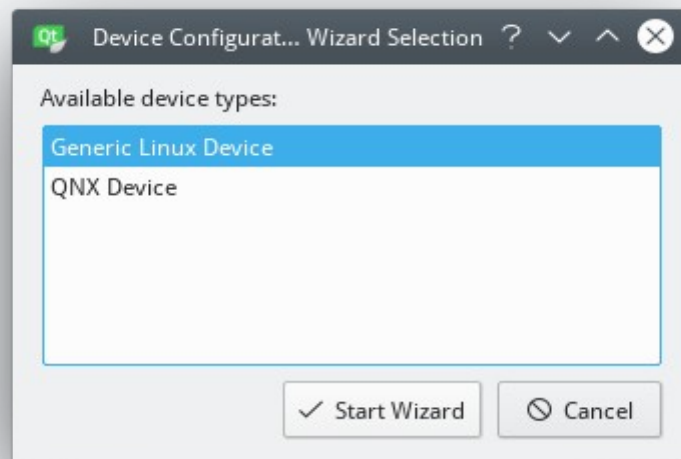
9.Setup Qt Creator for Raspberry Pi cross compilation

If you do not have installed Qt Creator, follow the instructions in the [Qt download page](#) to download and install it

```
or $ sudo apt install qtcreator
```

then, follow these instructions to configure cross-compilation support.

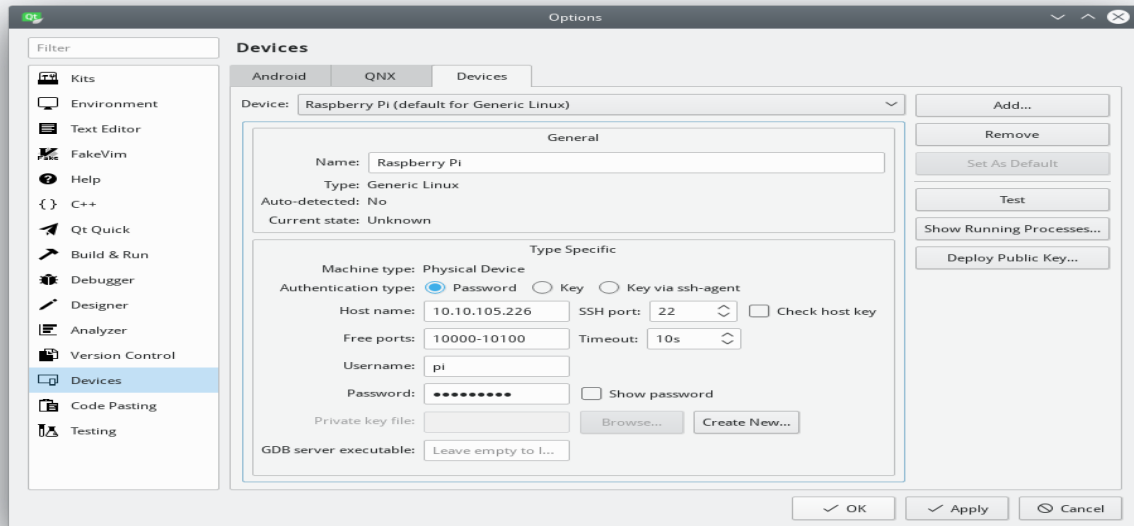
First, open Qt Creator, go to the Tools -> Options menu, select the Devices section and devices tab. Add a new Generic Linux Device.



Set a name for the configuration (Raspberry Pi), the network name or IP, the username (pi) and password (by default raspberry).



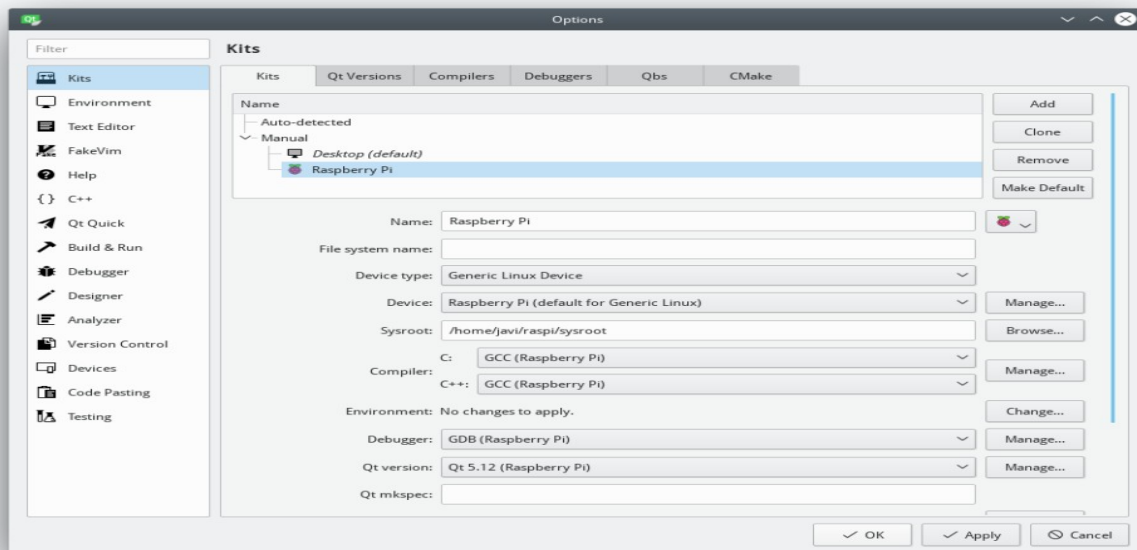
Before you finish the wizard, the connection to your Raspberry Pi will be verified and you will have then a new device configured in Qt Creator.

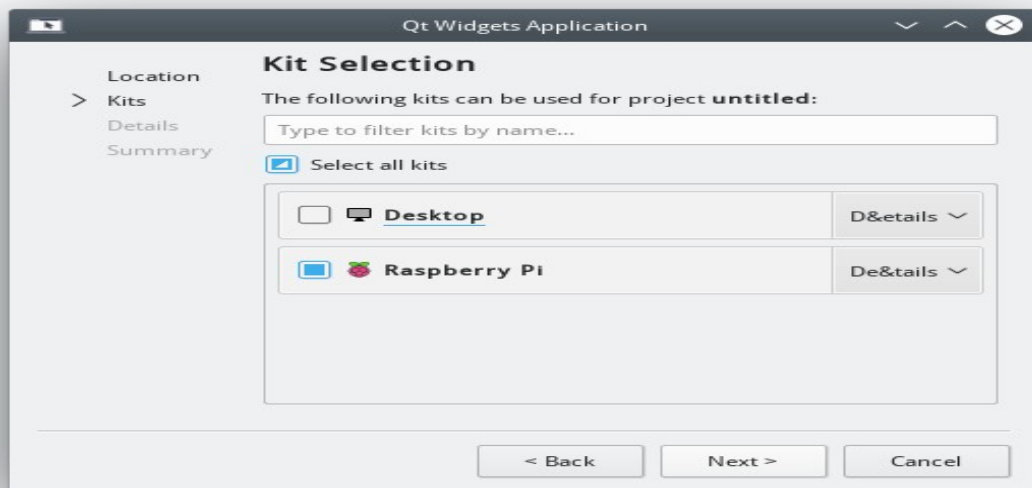


Next, go to the Kits section and Compilers tab, Add GCC C and C++ cross compilers for Raspberry Pi.

The path for the GCC C Raspberry Pi compiler is gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabi.tar.xz

Finally, go to the Kits tab, Add a new kit, set the name and icon you prefer, and pay attention to the following configuration





Congratulations! You can now design, build and deploy your Qt Raspberry Pi apps in you computer and, execute and debug them directly in your Raspberry Pi.