Raspberry Pi with Qt

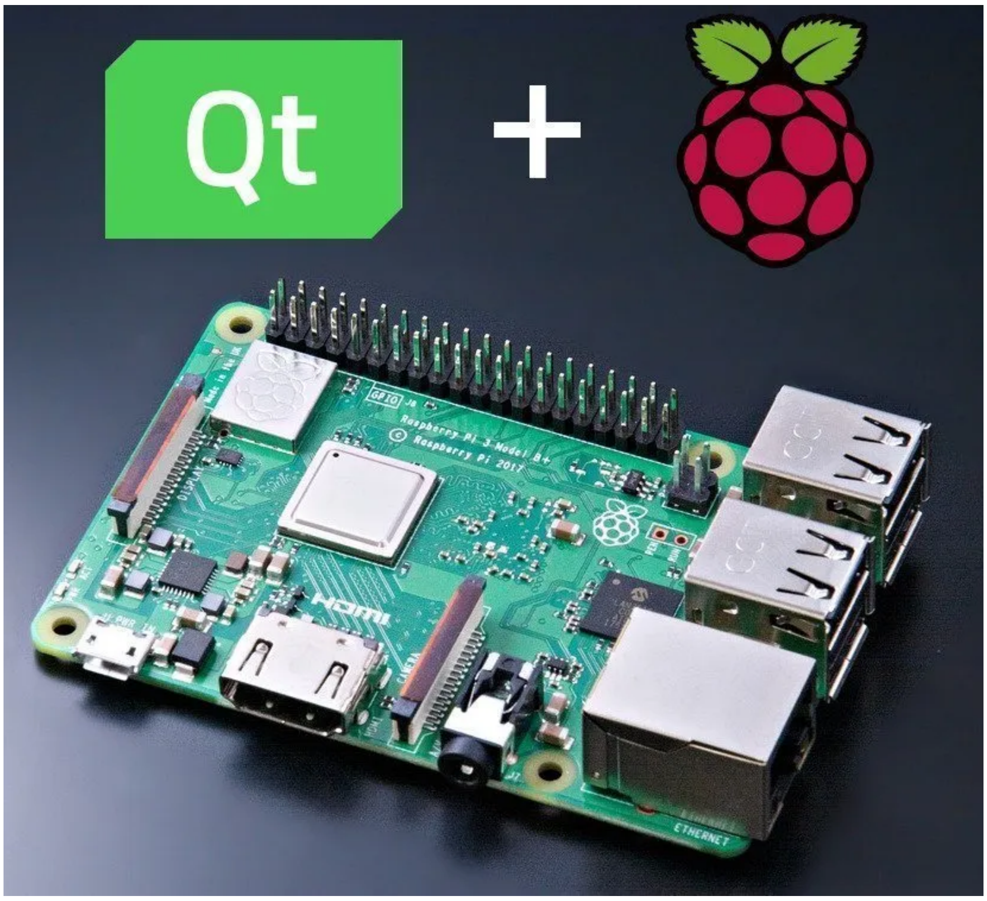


Table of Contents

Raspberry-PI with Qt 5.14.2

Introduction

Creating applications with Qt for Raspberry Pi is a rewarding and fun experience that allows you to unleash your creativity and innovation. Qt is a cross-platform framework that enables you to develop graphical user interfaces (GUIs) and applications that run on various hardware and operating systems. Raspberry Pi is a low-cost, credit-card-sized computer that can run Linux and other operating systems, and can be used for a variety of projects, such as robotics, gaming, smart home, and education. In this documentation, you will learn how to set up your development environment, how to create a cross compiler for Raspberry Pi, how to design and build GUIs with Qt Creator, and how to deploy and run your applications on Raspberry Pi. By the end of this documentation, you will have the skills and knowledge to create your own amazing applications with Qt for Raspberry Pi.

What is Cross-Compilation

Cross compilation in Qt for Raspberry Pi is a process of building Qt applications on a host machine (such as a PC) and then transferring them to a target device (such as a Raspberry Pi) for execution. This way, you can take advantage of the faster CPU and RAM of the host machine, and avoid the long compilation time and limited resources of the Raspberry Pi. Cross compilation also allows you to use Qt Creator, a powerful IDE for Qt development, on your host machine, and debug your applications remotely on the Raspberry Pi. To cross-compile Qt for Raspberry Pi, you need to have a cross-compiler toolchain, a sysroot directory that contains the target device's file system, and a CMake toolchain file that specifies the configuration options for Qt.

What are the prerequisite before follow this documentation

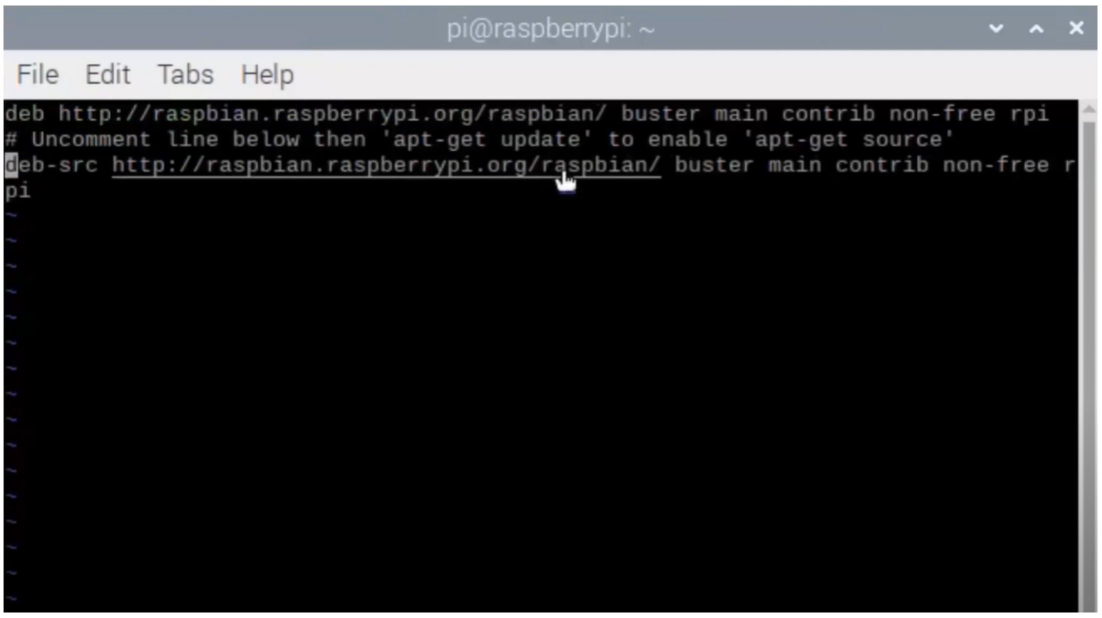
- Raspberry-Pi (This documentation is tested for Raspberry-Pi 2 model B. You should need to install raspbian OS to raspberry Pi. Next you need to install RealVNC server to connect Raspberry Pi to your host computer. Next you need to turn on ssh connection in Raspberry Pi.
- Host Machine (This documentation is tested for Ubuntu version is 22.04 – Ubuntu-22.04-desktop-amd64. After install Ubuntu you need to install RealVNC client to your machine. Turn on ssh connection in your host machine)
- Ethernet cable (for connect Raspberry Pi and Host machine)
- Power cable (for give power supple to Raspberry Pi)

Note :- After you connect your Raspberry Pi to Host machine via Ethernet you have to change your Raspberry Pi Ip Address. Host machine IP and Raspberry Pi IP should be same network

Follow this video URL :- Watch

Steps for configure Raspberry Pi

sudo vi /etc/apt/sources.list



```
deb http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free rpi
# Uncomment line below then 'apt-get update' to enable 'apt-get source'
deb-src http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free r
pi
```

Figure 1: Uncomment Last Line

sudo apt update

sudo apt full-upgrade

sudo reboot

sudo rpi-update

sudo reboot

Install Following Things

sudo apt-get build-dep qt5-qmake

sudo apt-get build-dep libqt5webengine-data

sudo apt-get install libboost1.58-all-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 libsrtp0-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

sudo apt-get install libgstreamer0.10-dev gstreamer-tools libraspberrypi-dev libx11-dev libglib2.0-dev

```
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 libatspi-dev libssl-dev libxcursor-dev libxcomposite-dev libxdamage-dev libfontconfig1-dev

sudo apt-get install libxss-dev libxtst-dev libpci-dev libcap-dev libsrtp0-dev libxrandr-dev libnss3-dev libdirectfb-dev libaudio-dev
```

Make Following Directory

```
sudo mkdir /usr/local/qt5pi

sudo chown <username>:<username> /usr/local/qt5pi
```

<u>Steps for configure host machine</u>

First update and upgrade your host machine

```
sudo apt-get update

sudo apt-get upgrade
```

Test your Raspberry Pi connection

```
ping <ip address of raspberry>
```

Go to root

```
sudo bash
```

Install following things

```
apt-get install gcc git bison python gperf pkg-config
apt install make
apt install libclang-dev
apt install build-essential
```

Note :- build-essential install gcc and g++ latest compilers. But this tutorial work with only gcc – 9 and g++ -9. for that we have to install gcc previous compilers.

Install GCC with Ubuntu via Toolchain PPA

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/ppa -y
sudo apt update
sudo apt install gcc-9 g++-9 gcc-10 g++-10 gcc-11 g++-11 g++-12 gcc-12 g++-13 gcc-13
```

Configure the priority of each version

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-12 40 -- slave /usr/bin/g++ g++ /usr/bin/g++-13 --slave /usr/bin/gcov gcov /usr/bin/gcov-13

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-13 90 -- slave /usr/bin/g++ g++ /usr/bin/g++-12 --slave /usr/bin/gcov gcov /usr/bin/gcov-12

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-11 80 -- slave /usr/bin/g++ g++ /usr/bin/g++-11 --slave /usr/bin/gcov gcov /usr/bin/gcov-11

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-10 60 -- slave /usr/bin/g++ g++ /usr/bin/g++-10 --slave /usr/bin/gcov gcov /usr/bin/gcov-10

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 100 -- slave /usr/bin/g++ g++ /usr/bin/g++-9 --slave /usr/bin/gcov gcov /usr/bin/gcov-9

Create new Directory and give permission to that

mkdir /opt/qt5pi
chown <username>:<username> /opt/qt5pi
cd /opt/qt5pi/

Get Linaro toolchain

Wget https://releases.linaro.org/components/toolchain/binaries/latest-7/ arm-linux-gnueabihf/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux- gnueabihf.tar.xz

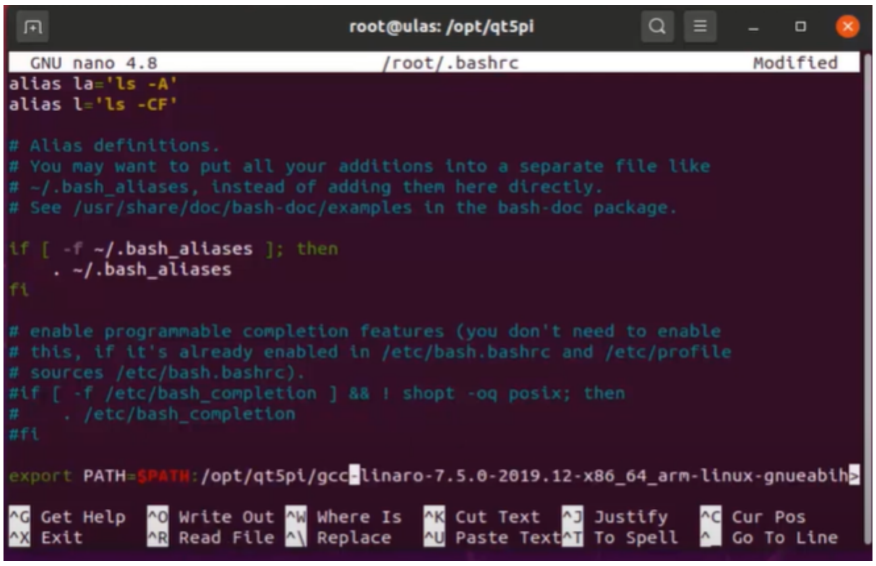
tar xf gcc-linaro-7.5.0-2019.12-x86_64_arm-linux- gnueabihf.tar.xz

export PATH=$PATH:/opt/qt5pi/gcc-linaro-7.5.0-2019.12- x86_64_arm- linux-gnueabihf/bin

Add above path to .bashrc

nano ~/.bashrc

Figure 2: Add path to .bashrc



```
root@ulas: /opt/qt5pi

  GNU nano 4.8                    /root/.bashrc                    Modified
alias la='ls -A'
alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

export PATH=$PATH:/opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabih>

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line
```

Download Qt

```
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
```

Create sysroot directory

```
mkdir sysroot
chown <username>:<username> sysroot
```

Open terminal in current directory and run follow step one by one

```
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/usr/include sysroot/usr
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/lib sysroot
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/usr/lib sysroot/usr
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/opt/vc 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/opt/vc/lib/libEGL.so sysroot/usr/lib/arm-linux-gnueabihf/libEGL.so.1.1.0
mv sysroot/usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.1.0 sysroot/usr/lib/arm-linux-gnueabihf/
libGLESv2.so.2.1.0_backup
ln -s sysroot/opt/vc/lib/libGLESv2.so sysroot/usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.1.0
mv sysroot/usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.1.0 sysroot/usr/lib/arm-linux-gnueabihf/
libGLESv2.so.2.1.0_backup
ls sysroot/usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.1.0
ln -s sysroot/opt/vc/lib/libGLESv2.so sysroot/usr/lib/arm-linux-gnueabihf/libGLESv2.so.2.1.0
ln -s sysroot/opt/vc/lib/libEGL.so sysroot/opt/vc/lib/libEGL.so.1
ln -s sysroot/opt/vc/lib/libGLESv2.so sysroot/opt/vc/lib/libGLESv2.so.2
```

Get sysroot-relativelinks.py

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

Check again sync all file with raspberry pi

```
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/lib sysroot
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/usr/include sysroot/usr
rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/usr/lib sysroot/usr

rsync -avz --rsync-path="sudo rsync" <username>@<rasp-IP>:/opt/vc sysroot/opt/

./sysroot-relativelinks.py sysroot
```

Create qt5build directory

```
mkdir qt5build

cd qt5build/
```

Compile All things

```
../qt-everywhere-src-5.15.2/configure -opengl es2 -device linux-rasp-pi2-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
qtdatavis3d -nomake examples -make libs -pkg-config -no-use-gold-linker -v
```

Note :- Check again above command paths according to your host machine. Stay one or two hour for complete this task

Make it

```
make -j8

make install

cd /opt/qt5pi/

rsync -avz --rsync-path="sudo rsync" sysroot/usr/local/qt5pi <username>@<rasp-IP>:/usr/local
```

Note:- Now cross-compiler is ready. After that you have to install Qt creator

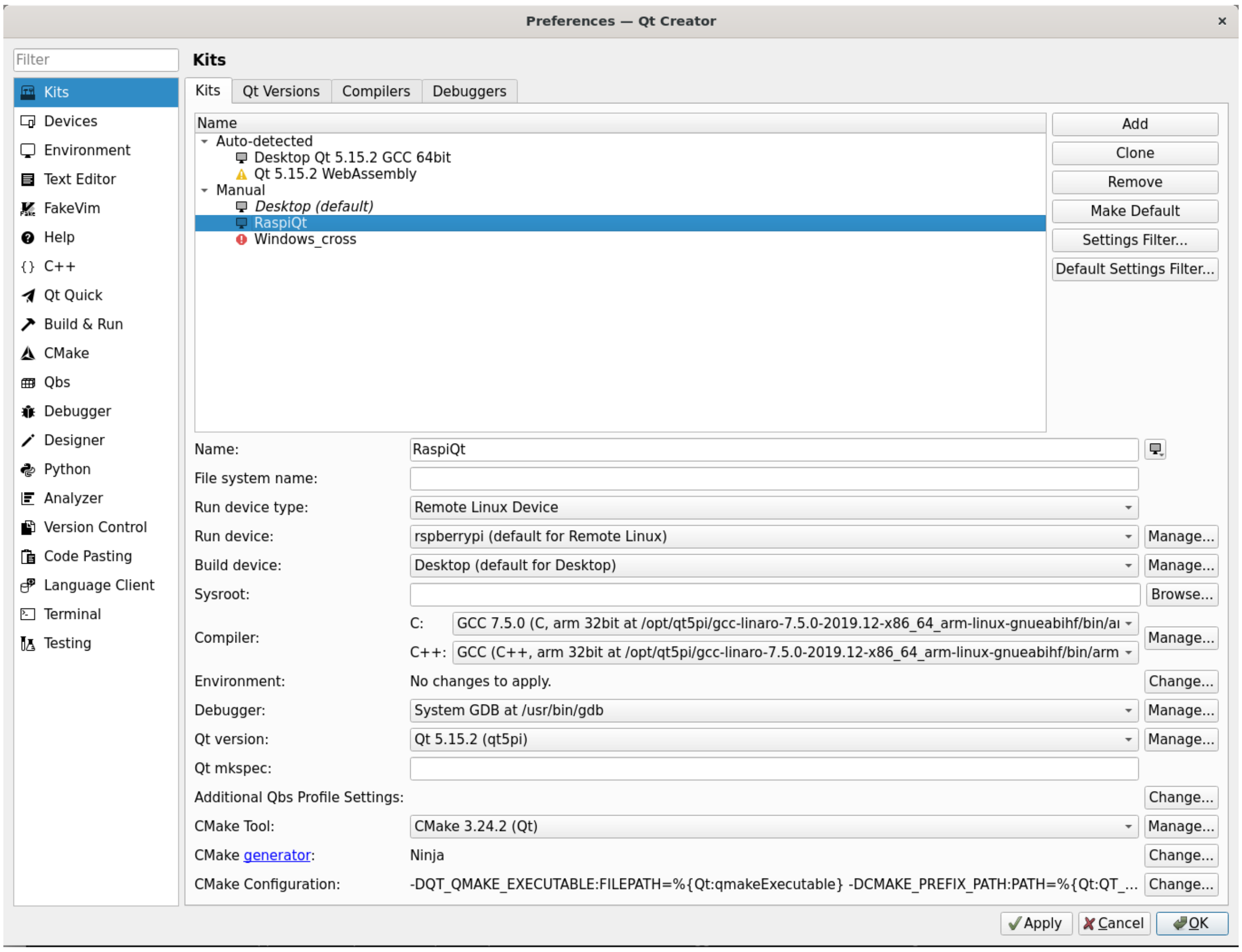
Install Qt Creator

Download Qt creator online installer and install it. Use qt 5.xx.xx version. Because in this we use qt5.

Configure Qt Creator

Create New Kit

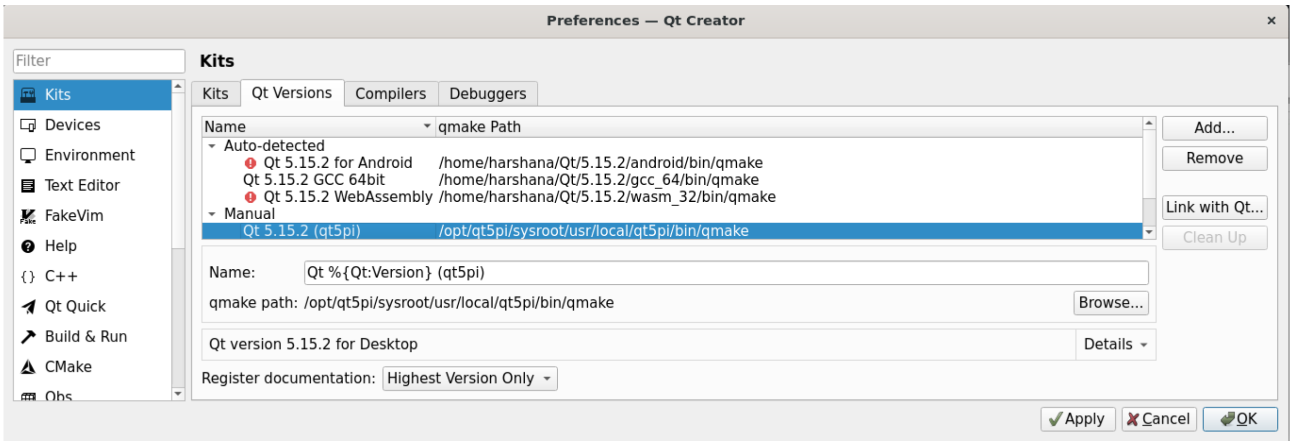
Note:- See follow screenshots and use that paths for your configurations



Name
- Auto-detected
  - Desktop Qt 5.15.2 GCC 64bit
  - Qt 5.15.2 WebAssembly
- Manual
  - *Desktop (default)*
  - RaspiQt
  - Windows_cross

Name:	RaspiQt
File system name:	
Run device type:	Remote Linux Device
Run device:	rspberrypi (default for Remote Linux)
Build device:	Desktop (default for Desktop)
Sysroot:	
Compiler:	C: GCC 7.5.0 (C, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/a
	C++: GCC (C++, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm
Environment:	No changes to apply.
Debugger:	System GDB at /usr/bin/gdb
Qt version:	Qt 5.15.2 (qt5pi)
Qt mkspec:	
Additional Qbs Profile Settings:	
CMake Tool:	CMake 3.24.2 (Qt)
CMake generator:	Ninja
CMake Configuration:	-DQT_QMAKE_EXECUTABLE:FILEPATH=%{Qt:qmakeExecutable} -DCMAKE_PREFIX_PATH:PATH=%{Qt:QT_...

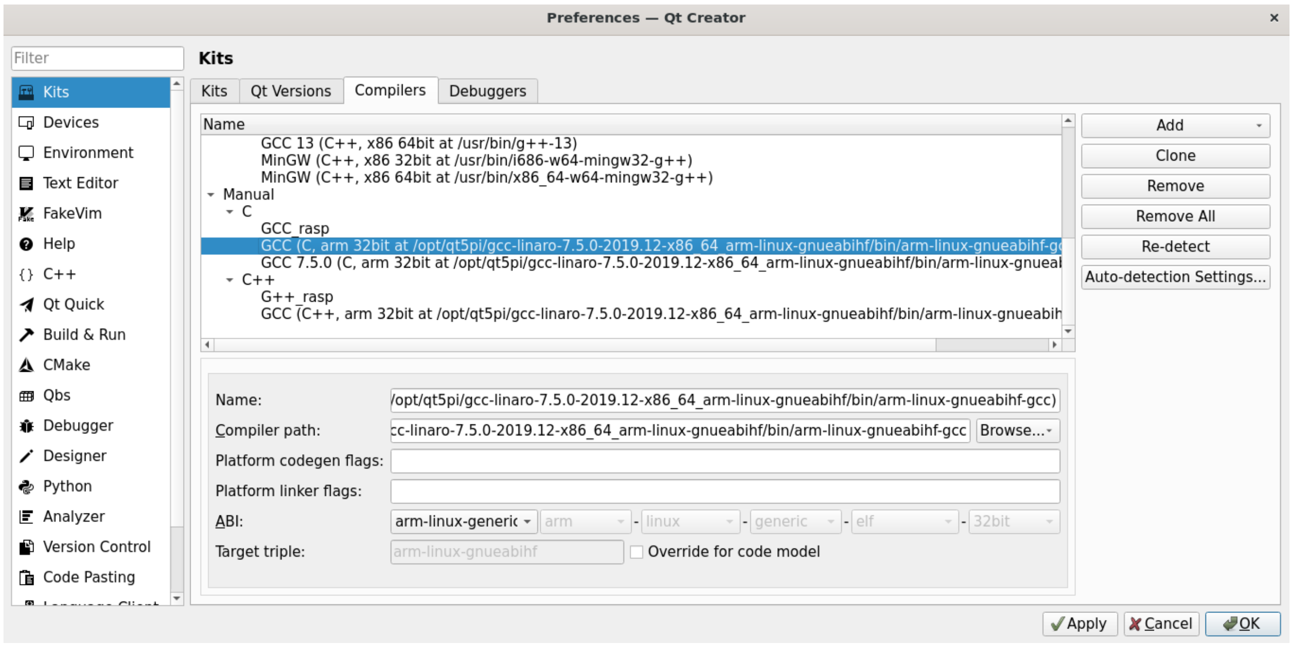
Figure 3: Kit Configuration



Name	qmake Path
Auto-detected	
Qt 5.15.2 for Android	/home/harshana/Qt/5.15.2/android/bin/qmake
Qt 5.15.2 GCC 64bit	/home/harshana/Qt/5.15.2/gcc_64/bin/qmake
Qt 5.15.2 WebAssembly	/home/harshana/Qt/5.15.2/wasm_32/bin/qmake
Manual	
Qt 5.15.2 (qt5pi)	/opt/qt5pi/sysroot/usr/local/qt5pi/bin/qmake

Name: Qt %{Qt:Version} (qt5pi)
qmake path: /opt/qt5pi/sysroot/usr/local/qt5pi/bin/qmake
Qt version 5.15.2 for Desktop
Register documentation: Highest Version Only

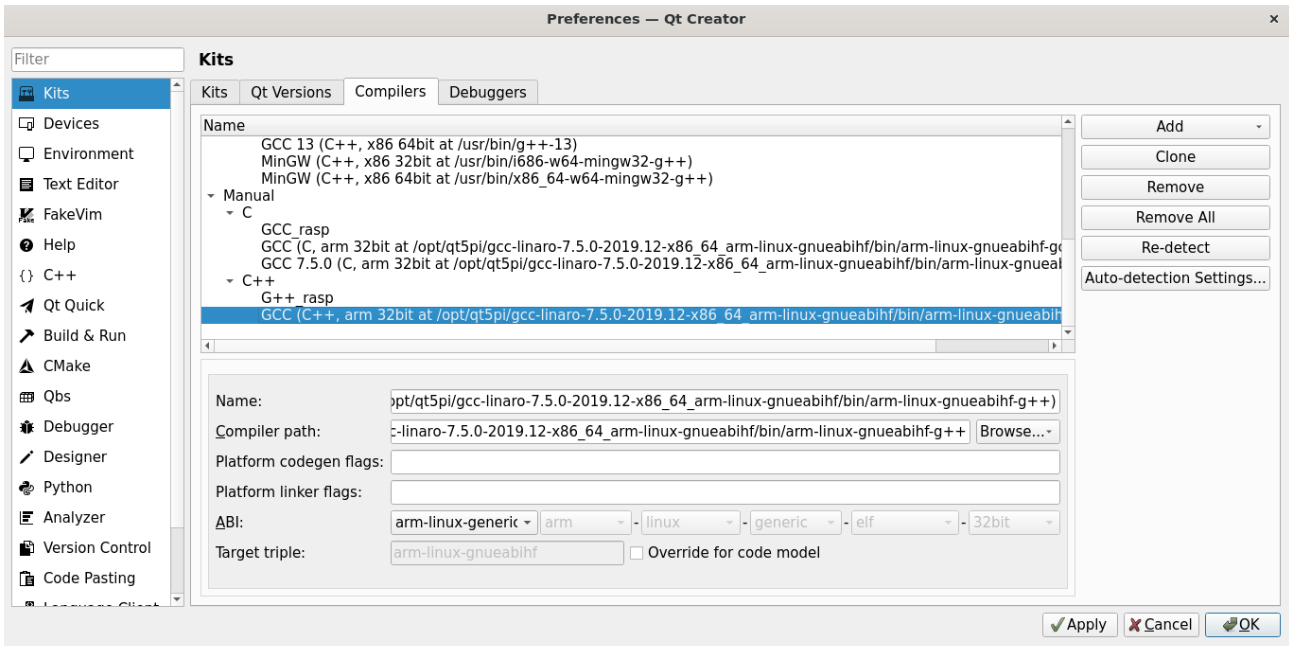
Figure 4: Qt Version Configuration



```
Name
        GCC 13 (C++, x86 64bit at /usr/bin/g++-13)
        MinGW (C++, x86 32bit at /usr/bin/i686-w64-mingw32-g++)
        MinGW (C++, x86 64bit at /usr/bin/x86_64-w64-mingw32-g++)
  Manual
    C
        GCC_rasp
        GCC (C, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gc
        GCC 7.5.0 (C, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueal
    C++
        G++_rasp
        GCC (C++, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabih
```

Name: /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc)
Compiler path: cc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
Platform codegen flags:
Platform linker flags:
ABI: arm-linux-generic - arm - linux - generic - elf - 32bit
Target triple: arm-linux-gnueabihf
Override for code model

Figure 5: Add GCC Compiler - Linaro Toolchain

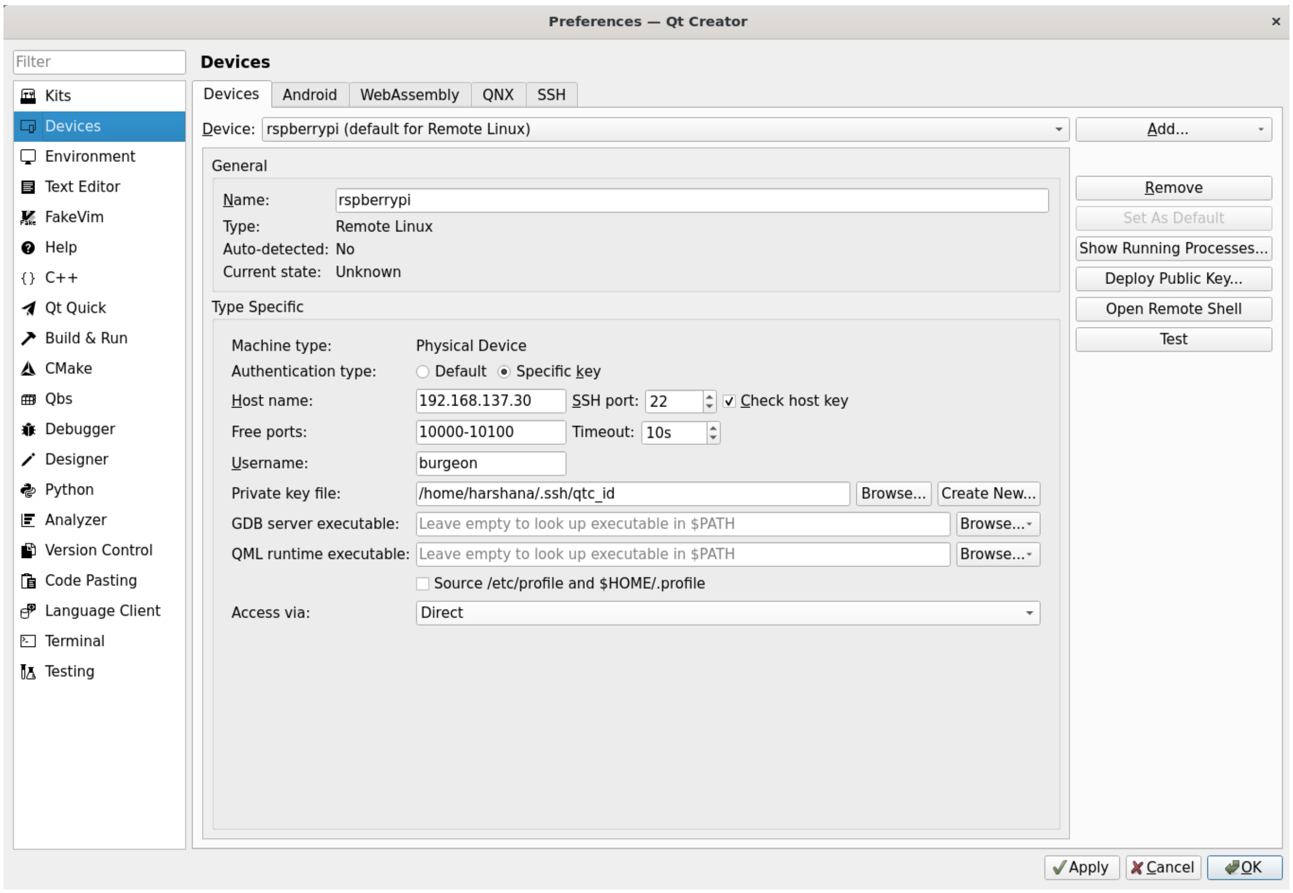


```
Name
        GCC 13 (C++, x86 64bit at /usr/bin/g++-13)
        MinGW (C++, x86 32bit at /usr/bin/i686-w64-mingw32-g++)
        MinGW (C++, x86 64bit at /usr/bin/x86_64-w64-mingw32-g++)
  Manual
    C
        GCC_rasp
        GCC (C, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gc
        GCC 7.5.0 (C, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueal
    C++
        G++_rasp
        GCC (C++, arm 32bit at /opt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabih
```

Name: pt/qt5pi/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++)
Compiler path: -linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++
Platform codegen flags:
Platform linker flags:
ABI: arm-linux-generic - arm - linux - generic - elf - 32bit
Target triple: arm-linux-gnueabihf
Override for code model

Figure 6: Add G++ Compiler - Linaro Toolchain

Add Raspberry Pi Device



Device: rspberrypi (default for Remote Linux)

General

Name: rspberrypi
Type: Remote Linux
Auto-detected: No
Current state: Unknown

Type Specific

Machine type: Physical Device
Authentication type: Default [✓ Specific key]
Host name: 192.168.137.30 SSH port: 22 [✓] Check host key
Free ports: 10000-10100 Timeout: 10s
Username: burgeon
Private key file: /home/harshana/.ssh/qtc_id
GDB server executable: Leave empty to look up executable in $PATH
QML runtime executable: Leave empty to look up executable in $PATH
[ ] Source /etc/profile and $HOME/.profile
Access via: Direct

Note :- This step is necessary one. In this step you have to create private key file when you are going to add the device. Otherwise you will have errors when you are going to compile Qt applications

Finish

Now all configurations are complete. Now you can create application for raspberry pi. When you are going to create new project choose newly created kit. Then create project.

After build your project you can copy that build folder to raspberry pi. Then you can execute that application on raspberry pi

Reference

Tutorial Video -> Video

All Commands -> All Commands

Linaro toolchain path -> Toolchain