

A Guide for Cross-compiling QT 5.13.2 For Raspberry Pi 3

WRITTEN BY RONEL JORDAN TCHOULAYEU TIENTCHEU

ABOUT ME

RONEL JORDAN TCHOULAYEU TIENTCHEU

- QT/EMBEDDED SOFTWARE DEVELOPER
- MORE THAN 10 YEARS OF 7 YEARS OF EXPERIENCE WITH THE QT FRAMEWORK
- 3 YEARS OF EXPERIENCE WITH EMBEDDED LINUX AND MICROCONTROLLERS
- ANALYST – PROGRAMMER (8 Programming Languages)
- HARDWARE DESIGN EXPERIENCE

Contact: jordanprog@yahoo.fr
+237 655 27 51 09

[Linked In](#) : RONEL TCHOULAYEU
Github : Jordanprog86

CONTEXT

- Have you ever dreamed to port your Gui Application to embedded Linux devices?
- Are you a c++ or Qt Developer usually working on windows and you want to deploy your software to Raspberry pi?
- Do you often ask yourself how to create GUI applications for embedded Linux and especially raspberry pi?

Then this article is for you. We will demonstrate how to port your Qt application on windows to Linux.

Purpose

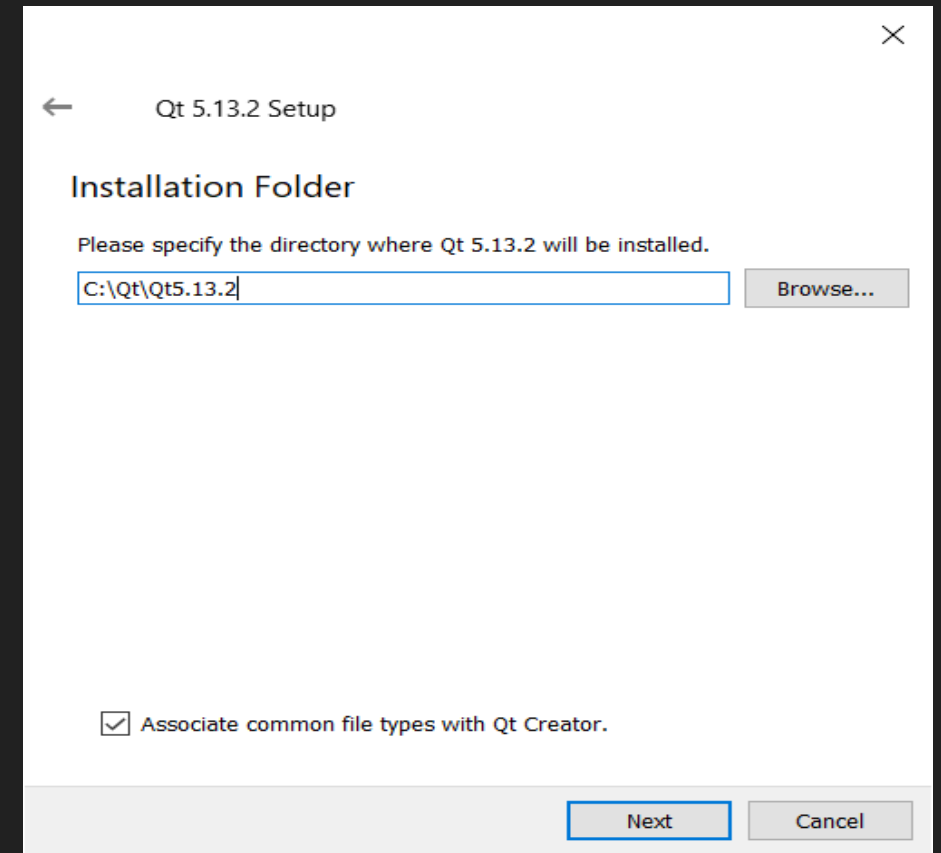
- This guide will help you building the Qt5 framework for Raspberry Pi 3 on Windows
- It assumes that you have already installed the Raspbian image in the Raspberry Pi
- You will be able to create applications on windows, and deploy them directly to the Raspberry with Qt.
- You should also download and install git, as it is required for SSH configuration in Qt

Note: After installing Raspbian, make sure to enable SSH and configure Hostname and password. The default username is pi.

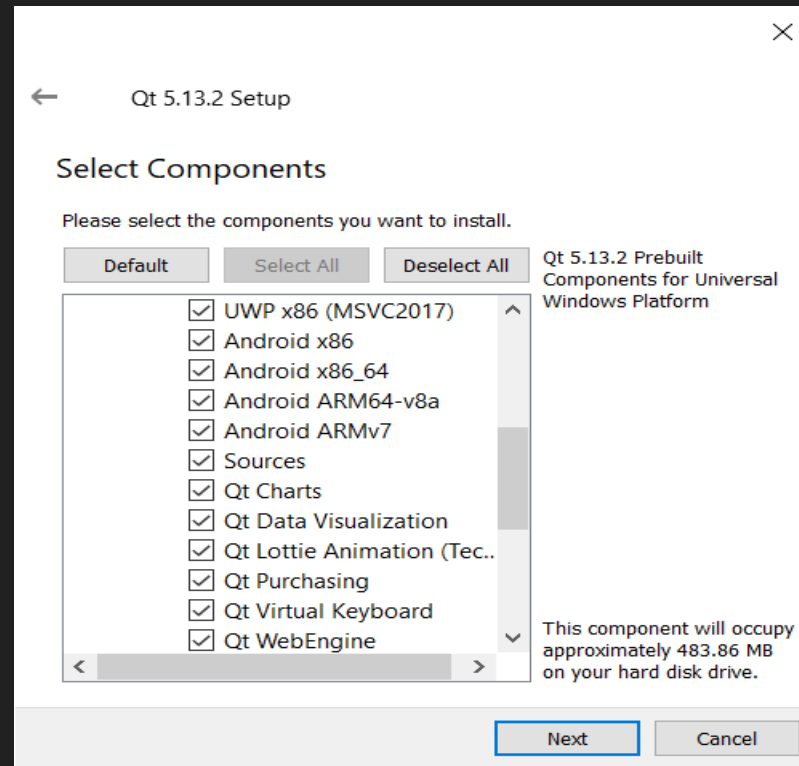
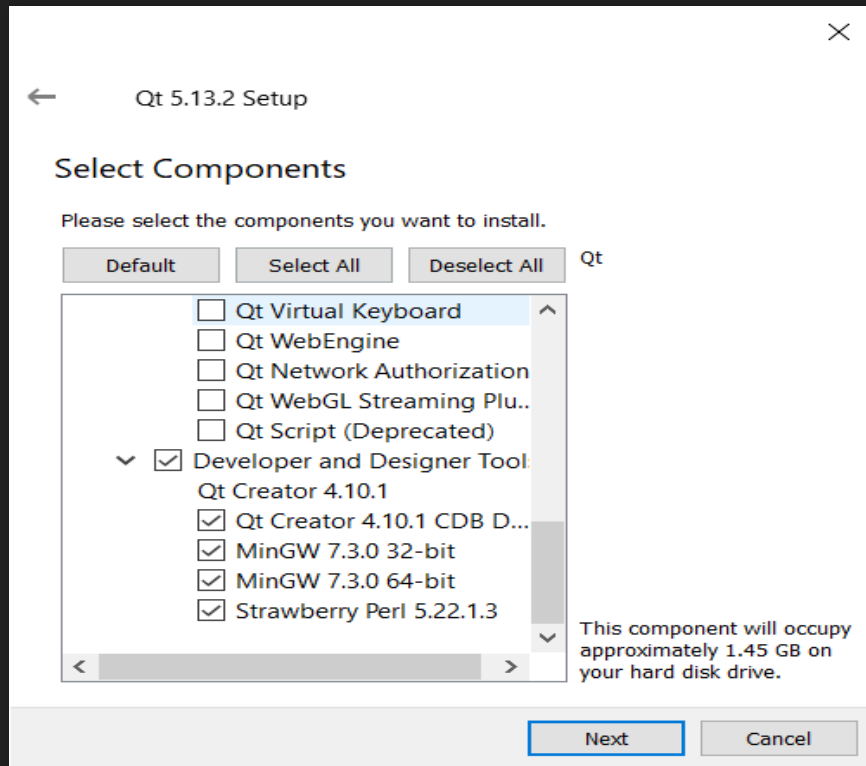
1- Downloading and Installing Qt 5.13.2

- Download Qt 5.13.2 for windows to this address <https://download.qt.io/archive/qt/5.13/5.13.2/>
- Select [qt-opensource-windows-x86-5.13.2.exe](#)
- After the Download, Launch the installer

Downloading and Installing Qt 5.13.2

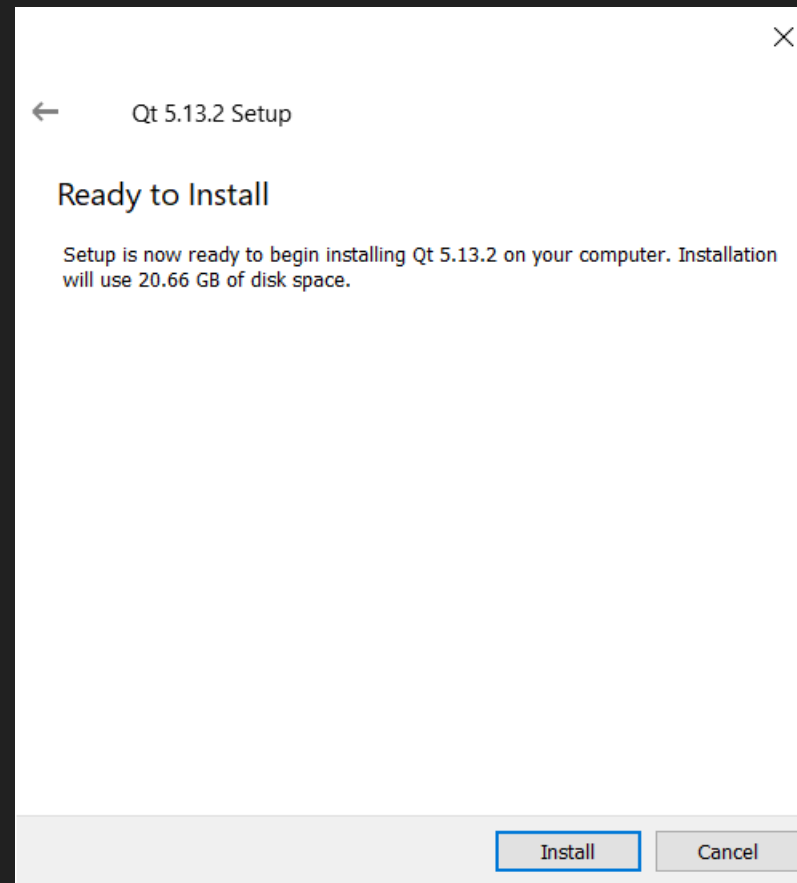
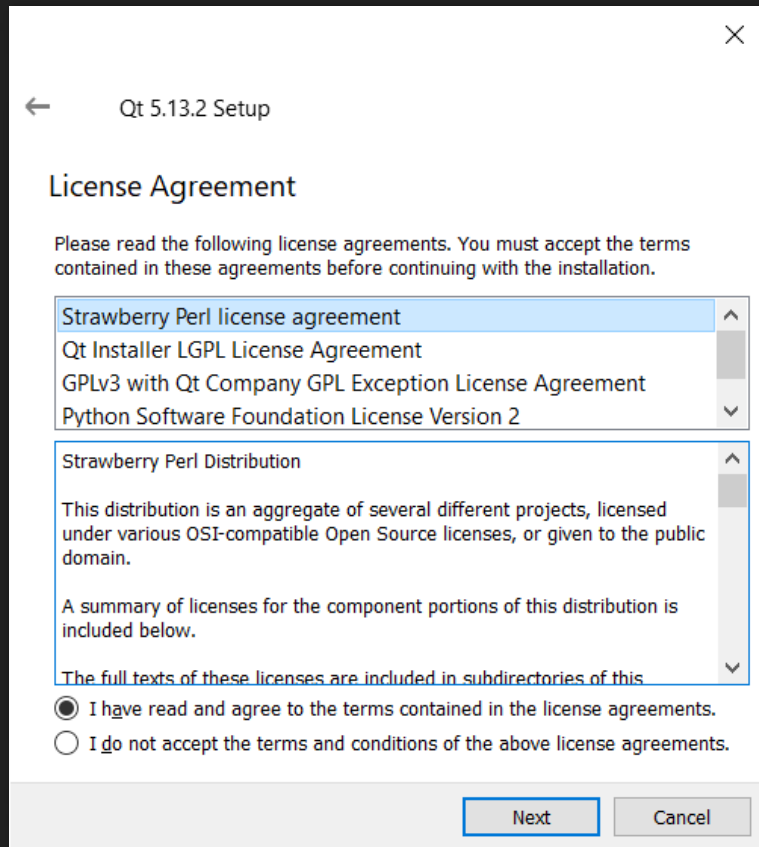


Downloading and Installing Qt 5.13.2



- Make sure you have selected all the Developer Tools as well as the sources.
- To maximize the modules we will have, It may also be good to select all modules except UWP platform

Downloading and Installing Qt 5.13.2



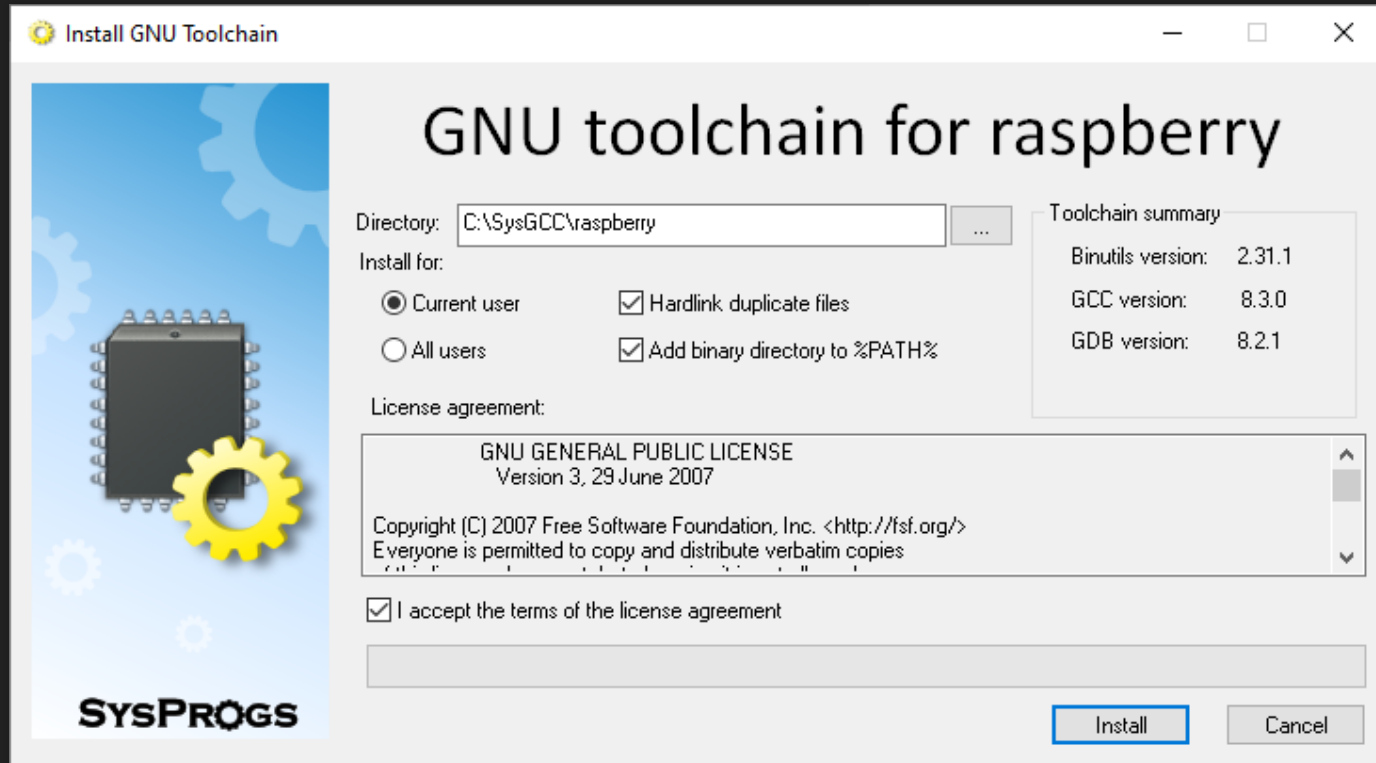
- Accept License Agreement and start to Install
- When it has finished to install, go to `C:\Qt\Qt5.13.2\vcr edist\` and install vc credits.

2- Downloading of Toolchains

We also need toolchains to compile Qt for our target environment, which is Linux in our case.

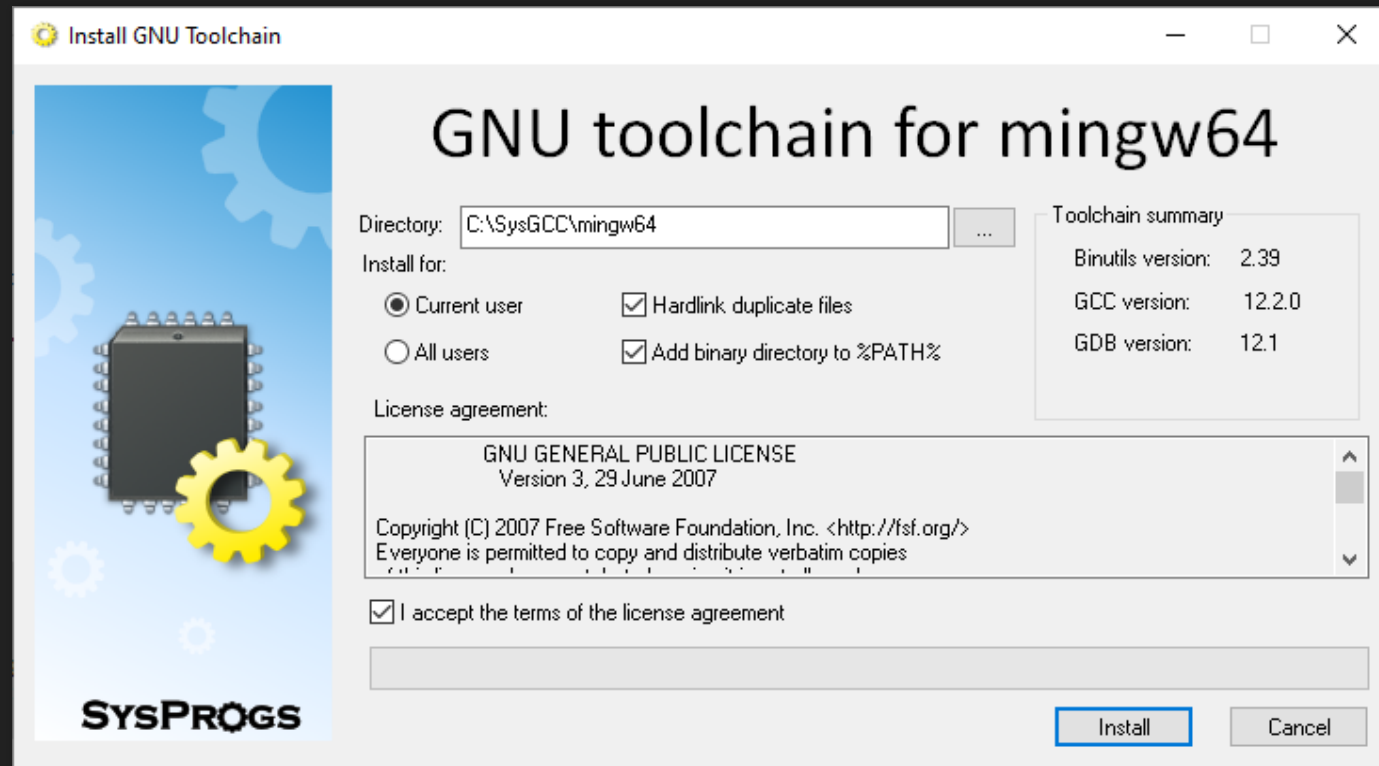
- Raspberry Toolchain : <https://gnutoolchains.com/raspberry/> GCC version 8.3.0
- Mingw32 Toolchain : <https://gnutoolchains.com/mingw32/> GCC version 4.8.1
- Mingw64 Toolchain : <https://gnutoolchains.com/mingw64/> GCC version 12.2.0

3-Installing Toolchains



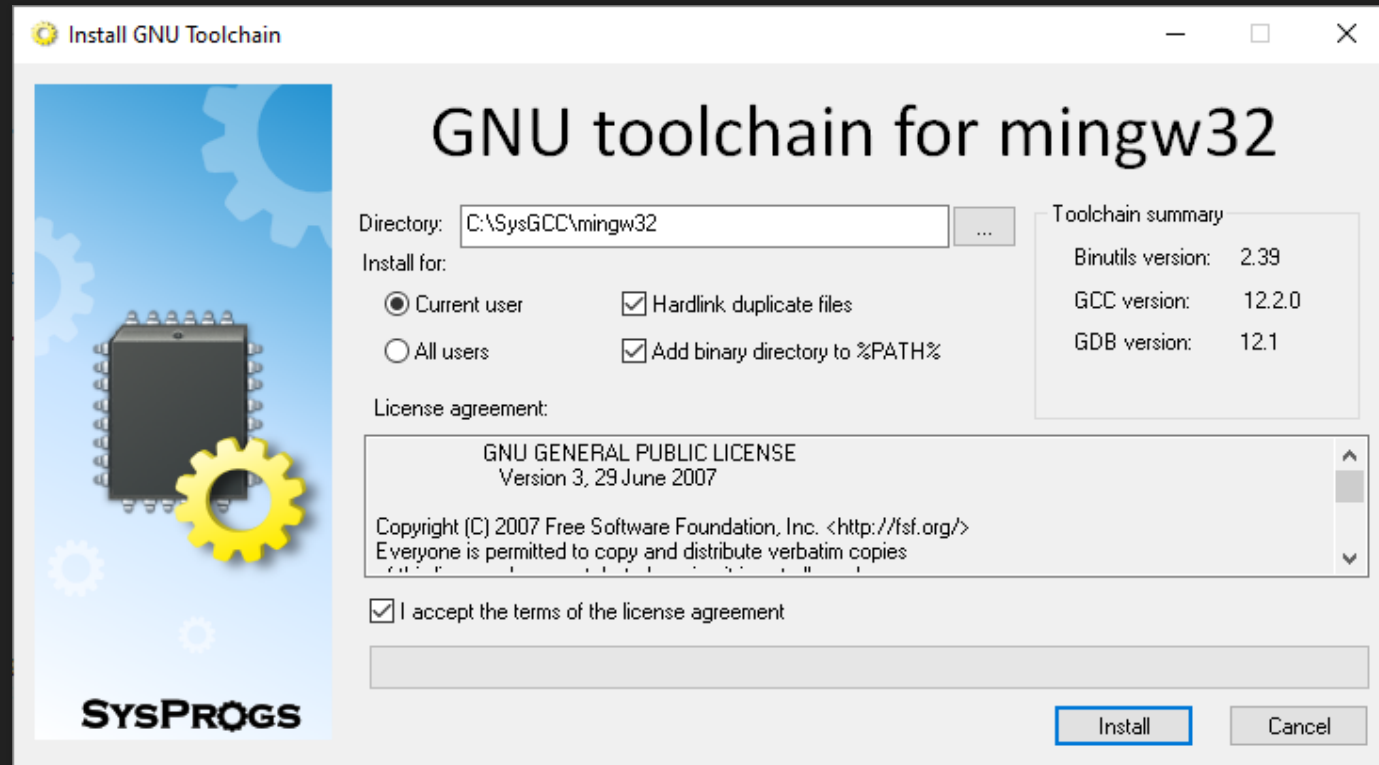
- Just accept the license agreement and install
- After the installation make sure it is added to path

Installing toolchains



- o Do the same for mingw64

Installing Toolchains



The benefits of this version of Mingw is that **msys** is included

It is a command line tool similar to linux command line tools.

Therefore, you will be able to easily run the configure script from it.

4-Installing Python

- Download and install both python 2.7.5 or later to this address <https://python.org/downloads/>
- After the installation, make sure it is added to PATH
- Python is required to build web engine

5-Synchronizing Sysroot

- In this step, we need to ensure that our local toolchain includes all the libraries and headers that the raspberry has. If not, some platform plugins will not be available like EGLFS.
- Download and install SmarTTY <https://sysprogs.com/SmarTTY/download>
- We will synchronize these remote directories:

/lib

/usr/include

/usr/lib

/usr/local/include/

/usr/local/lib

With local directory `C:\sysgcc\Raspberry\arm-linux-gnueabihf\sysroot`

Synchronizing Sysroot

- Connect your Pc and raspberry pi through a wifi connexion
- Launch SmartTTY and create a connection to the Raspberry with its Ip address. This time, the Hostname is the pi **ip Address**
- Select the smart terminal because it is easier to use
- Download each directory to the local sysroot path
- Then end the connection and go to the next step.

Synchronizing Sysroot

SmarTTY - New SSH Connection

Setup new SSH connection

Host Name: 192.168.1.75

User Name: pi

Connection alias:

Authentication method

- Password:
- ☒ Setup public key authentication and don't ask for password again
- Public key in Windows key store (associated with your user account): Auto RSA DSA
- OpenSSH key
 - ☐ Override default key location:
 - ☐ Passphrase:

☐ Use HTTP CONNECT proxy:

☐ Enable zlib compression (recommended for slow connections)

Transfer folders using: On-the-fly TAR File-by-file SCP (slow)

☐ Enable keep-alive packets every: seconds

☒ Save this connection to connections list

Connect Cancel

Virtual Terminals

This version of SmarTTY introduces a new Smart Terminal mode. When editing command lines inside Smart Terminal tabs, SmarTTY will let you use mouse and Windows key shortcuts, show command suggestion popups and display the list of files in the current directory in a separate docked pane.

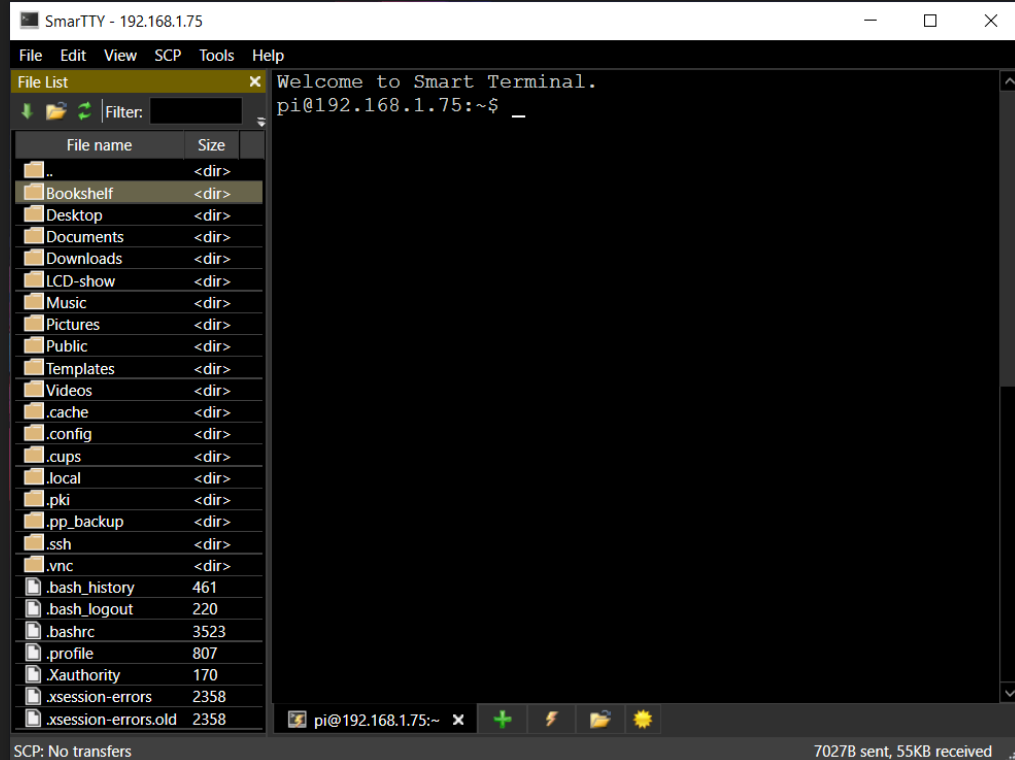
Please select the type of a terminal to create for the new session:

⚡ Start with a Smart Terminal

💻 Start with a regular Terminal

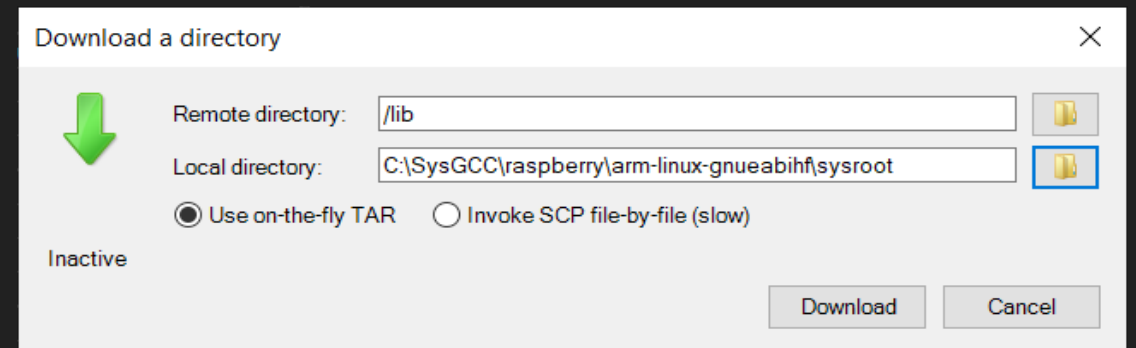
☐ Remember the choice for all further sessions

Synchronizing Sysroot



Select **SCP** then **Download a directory**

Download each required directory



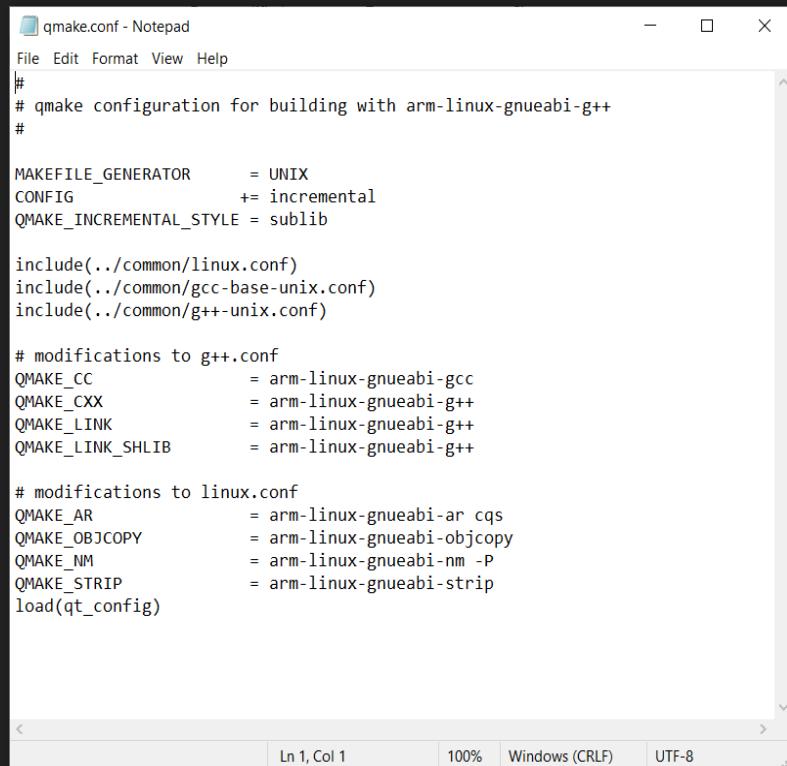
Synchronizing Sysroot

- We want to enable EGLFS support for Raspberry pi. By default, its libraries should be located in the opt/lib folder of the building host.
- Go to `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\arm-linux-gnueabihf` and copy `libEGL.so.1` and `libGLv2.so.2` to `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\opt\vc\lib` and `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot`
- Rename those files to `libEGL.so` and `libGLv2.so`
- In the `opt\vc\lib` folder copy and rename `libEGL_static.a` to `libEGL.a` and `libGLv2_static.a` to `libGLv2.a`

6-Modify qmake.conf files

- Create a new folder called qtraspberry
- Go to `path_of_installed_qt\Qt5.13.2\5.13.2` and copy Src folder to qtraspberry
- Go to `qtraspberry\Src` and open the file `\qtbase\mkspecs\linux-arm-gnueabi-g++\qmake.conf` in a textEditor
- Replace `arm-linux-gnueabi-` with `arm-linux-gnueabihf-` in the whole file
- Go to `\qtbase\mkspecs\win32-g++\qmake.conf` and add `-U__STRICT_ANSI__` to CXXFLAGS
- Results should be like this:

Modify qmake.conf files



```
qmake.conf - Notepad
File Edit Format View Help
#
# qmake configuration for building with arm-linux-gnueabi-g++
#

MAKEFILE_GENERATOR      = UNIX
CONFIG                  += incremental
QMAKE_INCREMENTAL_STYLE = sublib

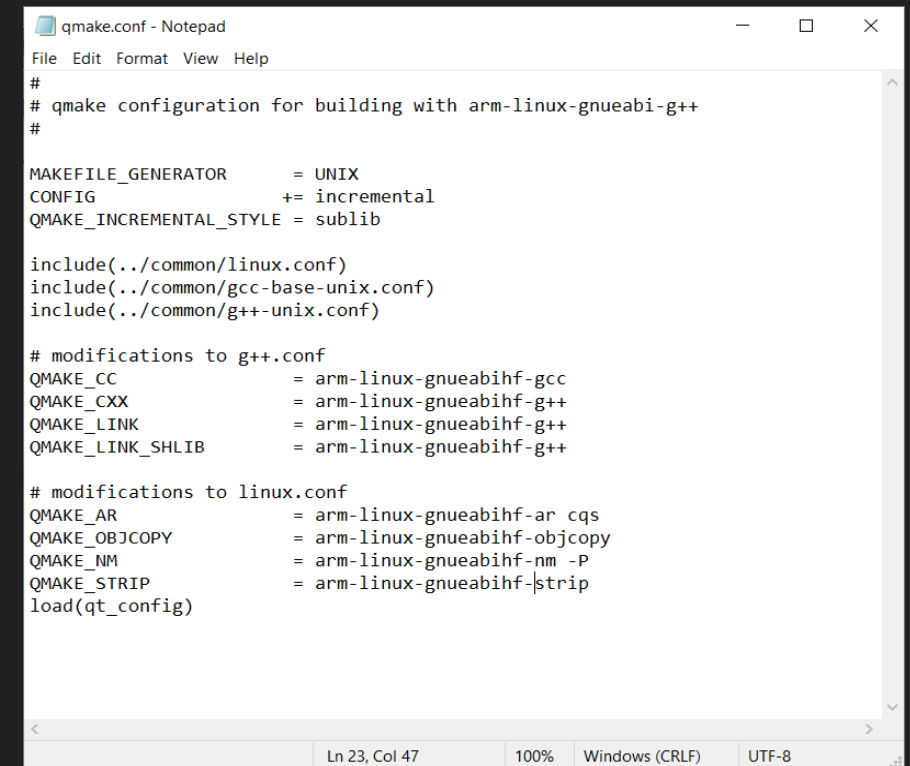
include(../common/linux.conf)
include(../common/gcc-base-unix.conf)
include(../common/g++-unix.conf)

# modifications to g++.conf
QMAKE_CC                = arm-linux-gnueabi-gcc
QMAKE_CXX               = arm-linux-gnueabi-g++
QMAKE_LINK              = arm-linux-gnueabi-g++
QMAKE_LINK_SHLIB        = arm-linux-gnueabi-g++

# modifications to linux.conf
QMAKE_AR                = arm-linux-gnueabi-ar cqs
QMAKE_OBJCOPY           = arm-linux-gnueabi-objcopy
QMAKE_NM               = arm-linux-gnueabi-nm -P
QMAKE_STRIP             = arm-linux-gnueabi-strip
load(qt_config)
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

Modifications of
qmake.conf in
linux-arm-gnueabi-
g++ folder



```
qmake.conf - Notepad
File Edit Format View Help
#
# qmake configuration for building with arm-linux-gnueabi-g++
#

MAKEFILE_GENERATOR      = UNIX
CONFIG                  += incremental
QMAKE_INCREMENTAL_STYLE = sublib

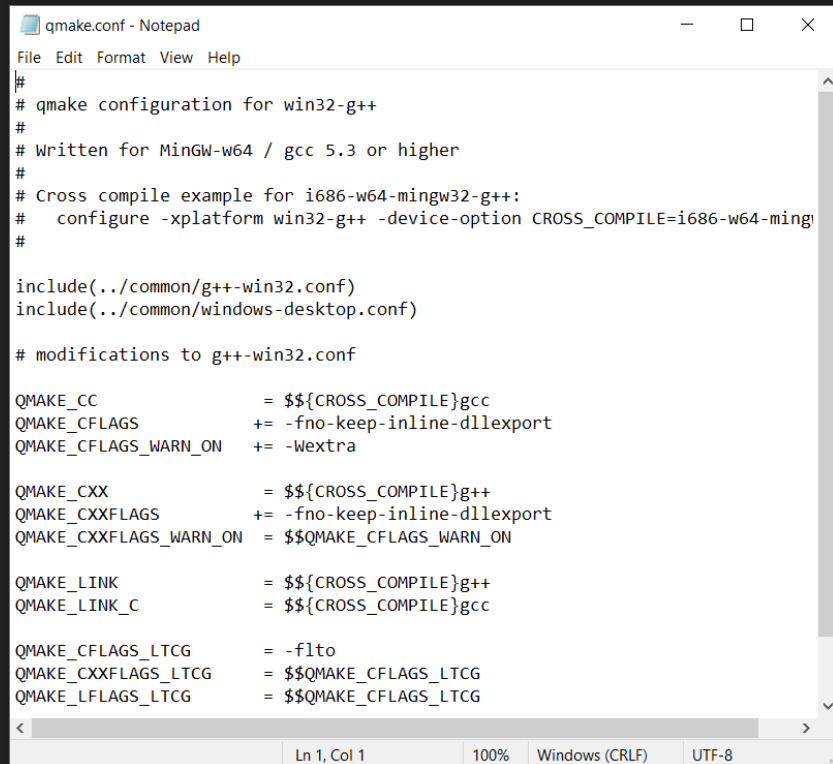
include(../common/linux.conf)
include(../common/gcc-base-unix.conf)
include(../common/g++-unix.conf)

# modifications to g++.conf
QMAKE_CC                = arm-linux-gnueabi-gcc
QMAKE_CXX               = arm-linux-gnueabi-g++
QMAKE_LINK              = arm-linux-gnueabi-g++
QMAKE_LINK_SHLIB        = arm-linux-gnueabi-g++

# modifications to linux.conf
QMAKE_AR                = arm-linux-gnueabi-ar cqs
QMAKE_OBJCOPY           = arm-linux-gnueabi-objcopy
QMAKE_NM               = arm-linux-gnueabi-nm -P
QMAKE_STRIP             = arm-linux-gnueabi-strip
load(qt_config)
```

Ln 23, Col 47 100% Windows (CRLF) UTF-8

Modify qmake.conf files



```
qmake.conf - Notepad
File Edit Format View Help
#
# qmake configuration for win32-g++
#
# Written for MinGW-w64 / gcc 5.3 or higher
#
# Cross compile example for i686-w64-mingw32-g++:
#   configure -xplatform win32-g++ -device-option CROSS_COMPILE=i686-w64-ming
#

include(../common/g++-win32.conf)
include(../common/windows-desktop.conf)

# modifications to g++-win32.conf

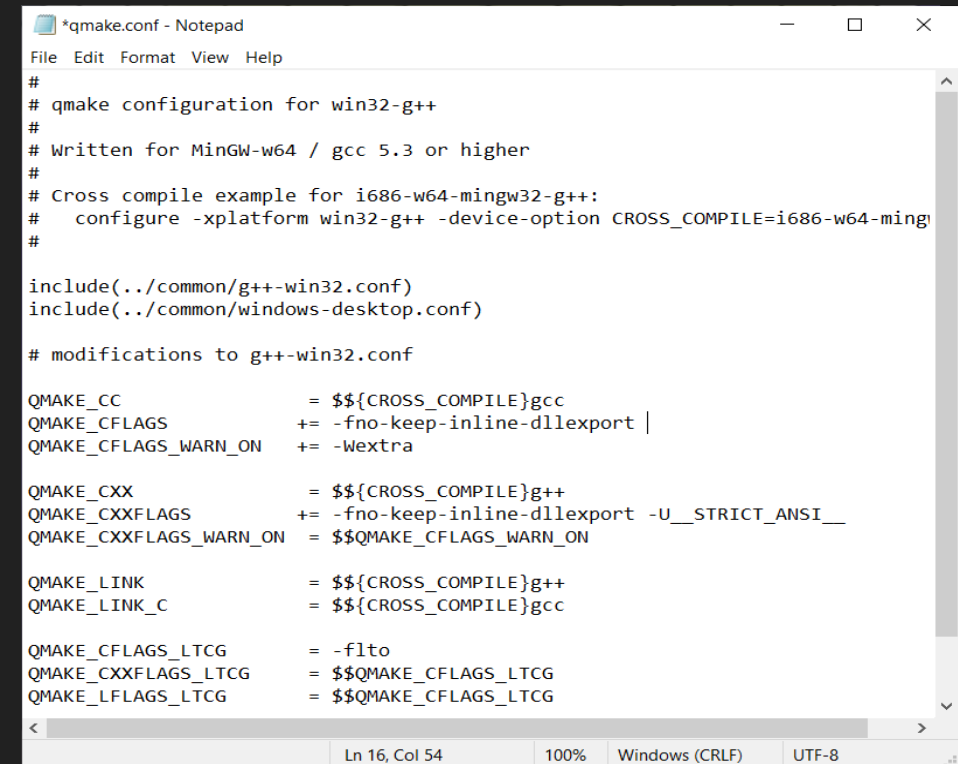
QMAKE_CC                = ${CROSS_COMPILE}gcc
QMAKE_CFLAGS            += -fno-keep-inline-dllexport
QMAKE_CFLAGS_WARN_ON    += -Wextra

QMAKE_CXX               = ${CROSS_COMPILE}g++
QMAKE_CXXFLAGS           += -fno-keep-inline-dllexport
QMAKE_CXXFLAGS_WARN_ON  = ${QMAKE_CFLAGS_WARN_ON}

QMAKE_LINK              = ${CROSS_COMPILE}g++
QMAKE_LINK_C            = ${CROSS_COMPILE}gcc

QMAKE_CFLAGS_LTCG        = -flto
QMAKE_CXXFLAGS_LTCG     = ${QMAKE_CFLAGS_LTCG}
QMAKE_LFLAGS_LTCG       = ${QMAKE_CFLAGS_LTCG}
```

Modifications of
qmake.conf in
win32-g++ folder



```
*qmake.conf - Notepad
File Edit Format View Help
#
# qmake configuration for win32-g++
#
# Written for MinGW-w64 / gcc 5.3 or higher
#
# Cross compile example for i686-w64-mingw32-g++:
#   configure -xplatform win32-g++ -device-option CROSS_COMPILE=i686-w64-ming
#

include(../common/g++-win32.conf)
include(../common/windows-desktop.conf)

# modifications to g++-win32.conf

QMAKE_CC                = ${CROSS_COMPILE}gcc
QMAKE_CFLAGS            += -fno-keep-inline-dllexport |
QMAKE_CFLAGS_WARN_ON    += -Wextra

QMAKE_CXX               = ${CROSS_COMPILE}g++
QMAKE_CXXFLAGS           += -fno-keep-inline-dllexport -U__STRICT_ANSI__
QMAKE_CXXFLAGS_WARN_ON  = ${QMAKE_CFLAGS_WARN_ON}

QMAKE_LINK              = ${CROSS_COMPILE}g++
QMAKE_LINK_C            = ${CROSS_COMPILE}gcc

QMAKE_CFLAGS_LTCG        = -flto
QMAKE_CXXFLAGS_LTCG     = ${QMAKE_CFLAGS_LTCG}
QMAKE_LFLAGS_LTCG       = ${QMAKE_CFLAGS_LTCG}
```

7-Building Qt5.13.2 for Raspberry Pi 3

- Navigate to `qtraspberry` folder and create a new folder called `build`
- Enter in the build folder. Then launch `msys.bat`. It is located in the folder `c:\SysGCC\MinGW32\msys\1.0\msys.bat`
- Navigate to build folder with `msys` : `cd /c/work/qtraspberry/build`
- Copy and run the build script :

```
../Src/configure -platform win32-g++ -device-option CROSS_COMPILE=arm-linux-gnueabihf- -  
release -opengl es2 -device linux-rasp-pi3-g++ -sysroot C:/SysGCC/Raspberry/arm-linux-  
gnueabihf/sysroot -opensource -confirm-license -I C:/SysGCC/raspberry/arm-linux-  
gnueabihf/sysroot/opt/vc/include -L C:/SysGCC/raspberry/arm-linux-gnueabihf/sysroot/opt/vc/lib
```
- In the build script, mentioning both device and xplatform is not supported for this version, hence we removed xplatform option

Building Qt5.13.2 for Raspberry Pi 3

```
Carl@DESKTOP-A9N93DC /c/work/qtraspberry/test
$ ../Src/configure -platform win32-g++ -device-option CROSS_COMPILE=arm-linux-gnueabi- -release -opengl es2 -device
linux-rasp-pi3-g++ -sysroot C:/SysGCC/Raspberry/arm-linux-gnueabi/sysroot -opensource -confirm-license
+ cd qtbase
+ /c/work/qtraspberry/Src/qtbase/configure -top-level -platform win32-g++ -device-option CROSS_COMPILE=arm-linux-gnueabi-
hf- -release -opengl es2 -device linux-rasp-pi3-g++ -sysroot C:/SysGCC/Raspberry/arm-linux-gnueabi/sysroot -opensource
-confirm-license
Preparing build tree...
Creating qmake...
.Done.
Info: creating stash file C:\work\qtraspberry\test\.qmake.stash

This is the Qt Open Source Edition.
```

```
MINGW32:/c/work/qtraspberry/test

This is the Qt Open Source Edition.

You have already accepted the terms of the Open Source license.

Running configuration tests...
Checking for machine tuple... yes
Checking for valid makespec... yes
Checking for target architecture... arm
Checking for host architecture... i386
Checking for alloca() in alloca.h... yes
Checking for C++14 support... yes
Checking for C++17 support... yes
Checking for C99 support... yes
Checking for C11 support... yes
```


Building Qt5.13.2 for Raspberry Pi 3

- When the configuration is successful you should see

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 ..... no
  EGLFS GBM ..... no
  EGLFS VSP2 ..... no
  EGLFS Mali ..... no
  EGLFS Raspberry Pi ..... yes
  EGLFS X11 ..... no
LinuxFB ..... yes
VNC ..... yes
Mir client ..... no
```

Note: No wayland-egl support detected. Cross-toolkit compatibility disabled.

WARNING: QDoc will not be compiled, probably because libclang could not be located. This means that you cannot build the Qt documentation.

Either ensure that llvm-config is in your PATH environment variable, or set LLVM_INSTALL_DIR to the location of your llvm installation.

On Linux systems, you may be able to install libclang by installing the libclang-dev or libclang-devel package, depending on your distribution.

On macOS, you can use Homebrew's llvm package.

On Windows, you must set LLVM_INSTALL_DIR to the installation path.

WARNING: Python version 2 (2.7.5 or later) is required to build QtWebEngine.

WARNING: host pkg-config not found

Qt is now configured for building. Just run 'gmake.exe'.

Once everything is built, you must run 'gmake.exe install'.

Qt will be installed into 'C:\SysGCC\Raspberry\arm-linux-gnueabihf\sysroot\usr\local\Qt-5.13.2'.

Prior to reconfiguration, make sure you remove any leftovers from the previous build.

Building Qt5.13.2 for Raspberry Pi 3

- You can now run `gmake && gmake install` or `make && make install`
- Otherwise, you can firstly run `gmake` or `make`, and after the build, `gmake install` or `make install`
- You might encounter errors while building `qtscrip`. If so, restart `gmake.exe -i`
- The `i` option is for ignoring errors
- After the build you will see as follows. You can finally Run `gmake.exe install`

```
gmake[2]: Leaving directory 'c:/work/qtraspberry/build/qtdoc/examples'
cd doc/src/cmake/ && ( test -e Makefile || c:/work/qtraspberry/build/qtbase/bin/qmake.exe -o Makefile C:/work/qtraspberry/Src/qtdoc/doc/src/cmake/cmake.pro ) && c:/Strawberry/c/bin/gmake -f Makefile
gmake[2]: Entering directory 'c:/work/qtraspberry/build/qtdoc/doc/src/cmake'
gmake[2]: Nothing to be done for 'first'.
gmake[2]: Leaving directory 'c:/work/qtraspberry/build/qtdoc/doc/src/cmake'
cd doc/ && ( test -e Makefile || c:/work/qtraspberry/build/qtbase/bin/qmake.exe -o Makefile C:/work/qtraspberry/Src/qtdoc/doc/doc.pro ) && c:/Strawberry/c/bin/gmake -f Makefile
gmake[2]: Entering directory 'c:/work/qtraspberry/build/qtdoc/doc'
gmake[2]: Nothing to be done for 'first'.
gmake[2]: Leaving directory 'c:/work/qtraspberry/build/qtdoc/doc'
gmake[1]: Leaving directory 'c:/work/qtraspberry/build/qtdoc'

Carl@DESKTOP-A9N93DC /c/work/qtraspberry/build
$
```

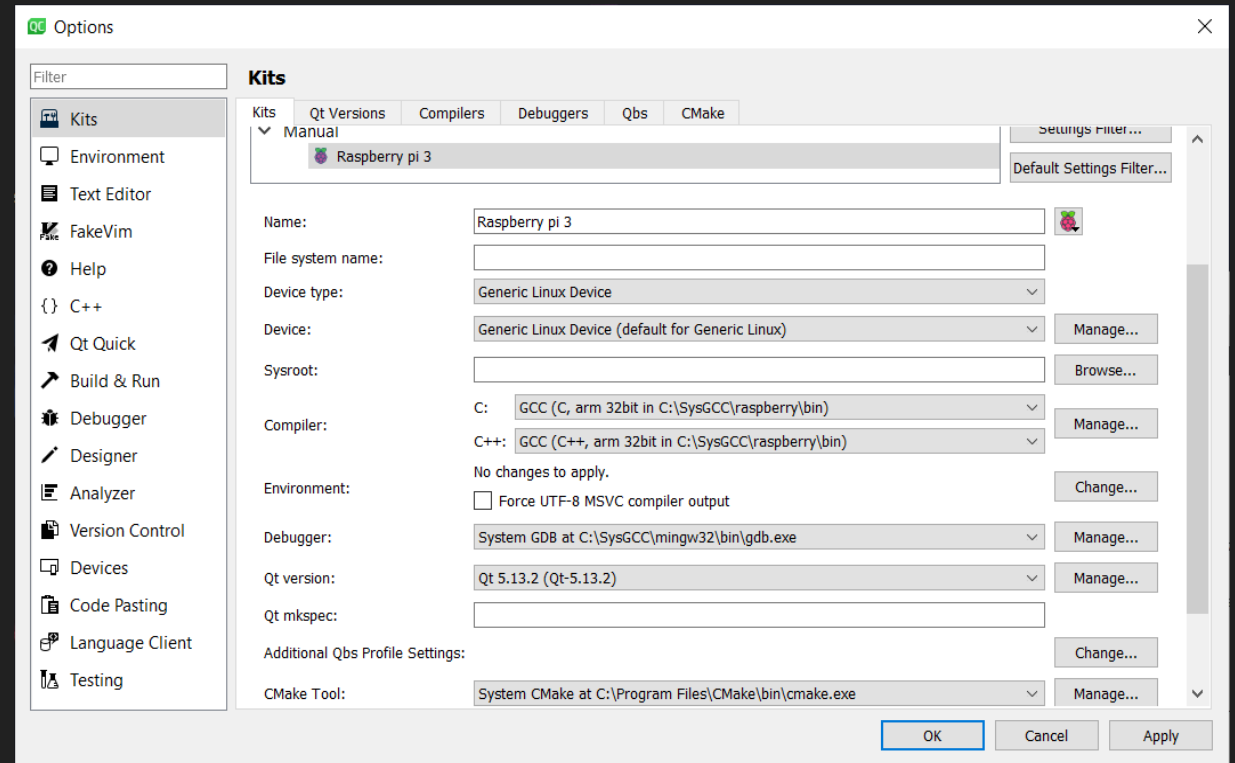
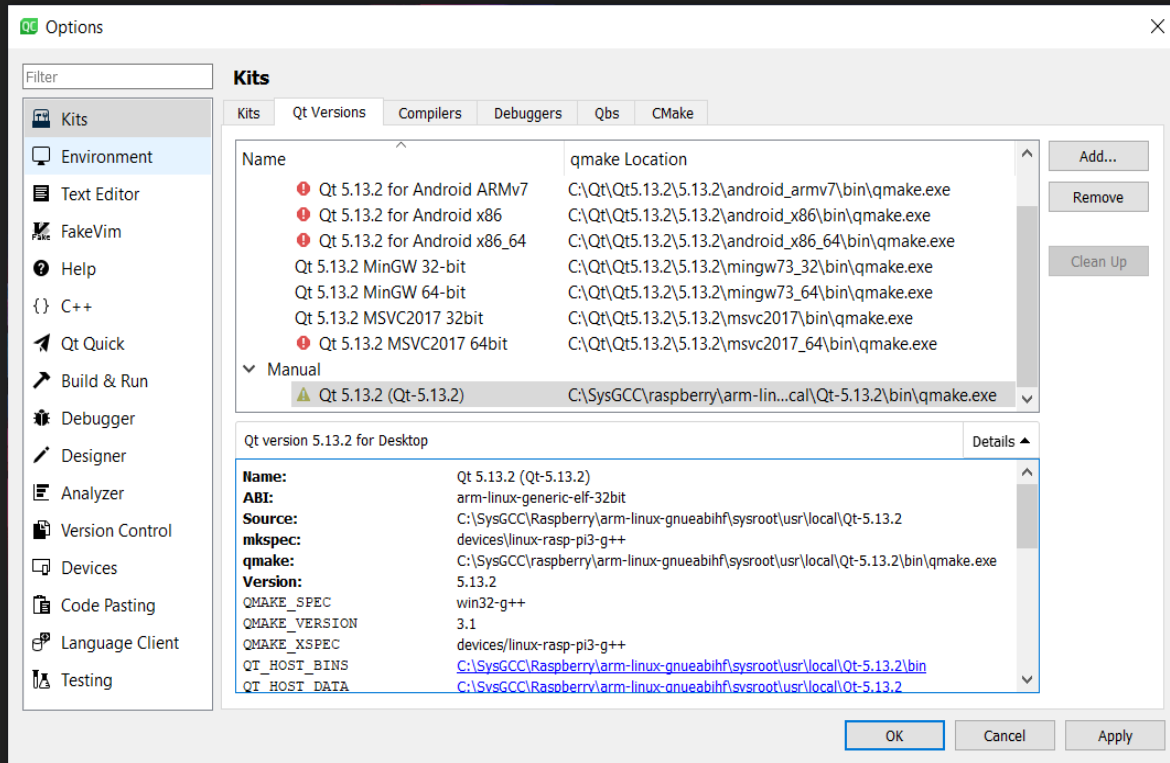
8- COPY GENERATED BUILD TO RASPBERRY

- After the installation, qt build will be available in `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\usr\local\Qt-5.13.2`
- You will need to copy the whole folder to remote `/usr/local` in Raspberry pi.
- Before that you will need to download fonts in `dejavu-fonts.github.io` , and copy them to `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\usr\local\Qt-5.13.2\lib\fonts` as Qt does not deploy fonts.
- When done, copy the Qt folder to `/usr/local` in Raspberry pi. The easiest way to achieve that would be to transfer it firstly in a USB key , then launch file manager as root with `sudo pcmanfm`
- Finally, you can paste qt build in the correct directory without root permission.

9-CREATE A KIT FOR RASPBERRY IN QT

- Open Qt Creator and go to Tools->Options->kits->Qt Version.
- Click on Add and navigate to `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\usr\local\Qt-5.13.2\bin` and select `qmake.exe`
- Go to kits and select Add. Give it the name `Raspberry pi`. In Device type, select Generic Linux Device
- Concerning the compilers, select GCC (`c:\SysGcc\raspberry\bin`)
- At Qt version select the version you previously created.
- Finally apply the changes and go to devices.

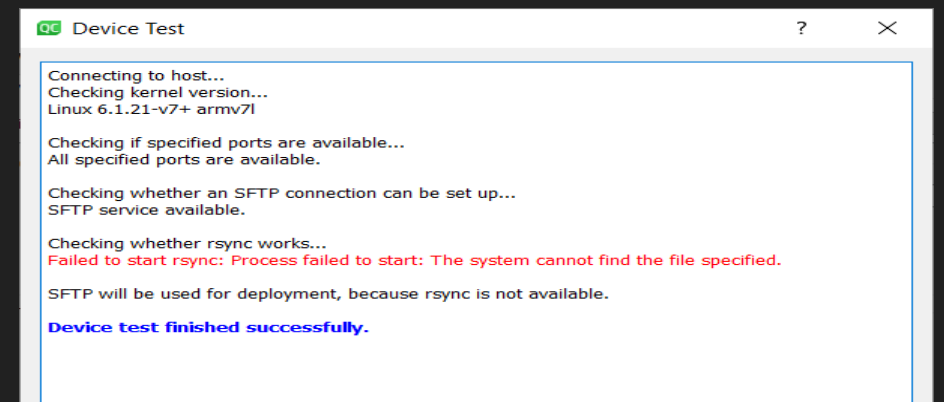
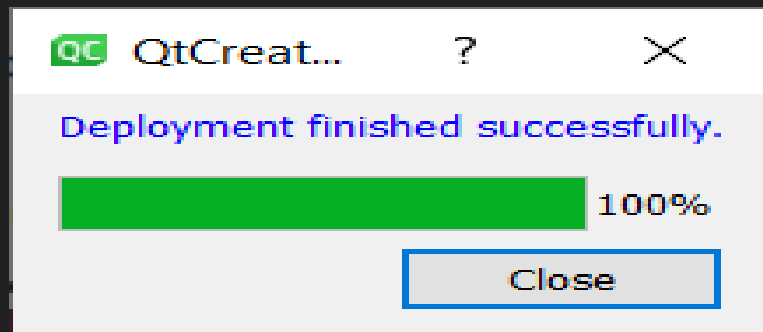
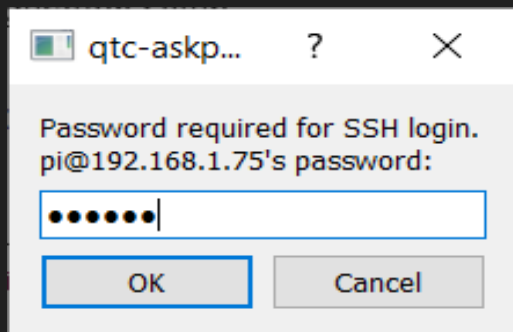
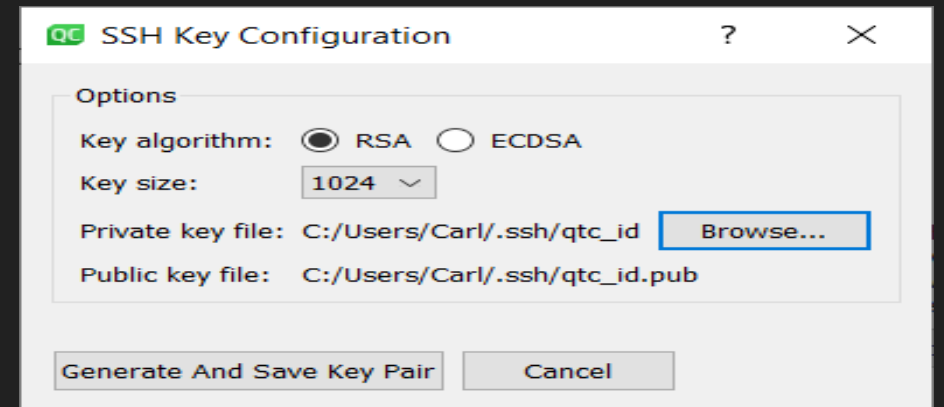
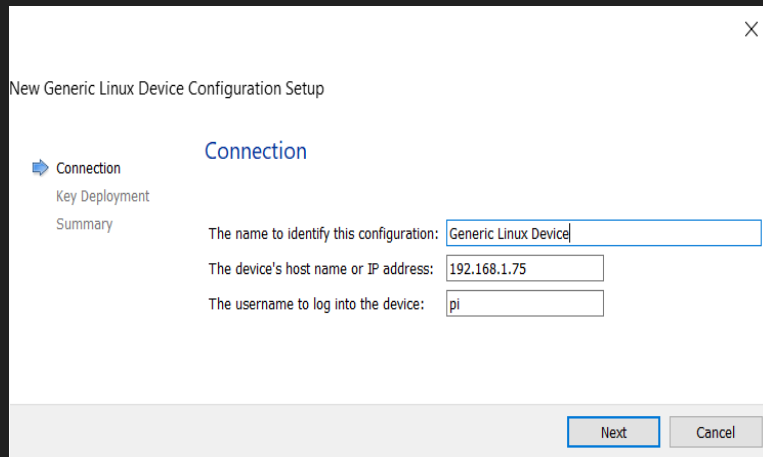
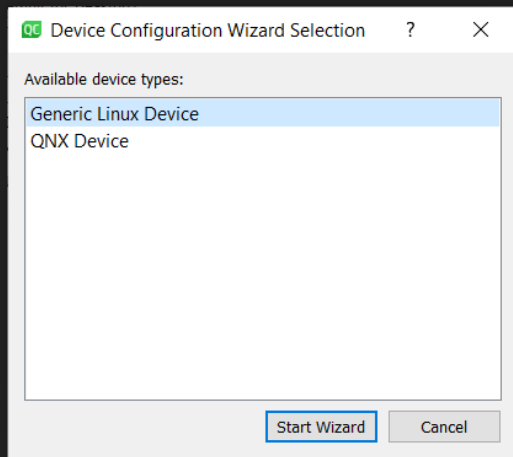
9-CREATE A KIT FOR RASPBERRY IN QT



10-ADDING A GENERIC LINUX DEVICE






- Select *Generic Linux Device*, then click on add
- Give a name to the device, enter Your Pi's IP Address, And the username (*generally pi*)
- Now in SSH Key Configuration Generate And save the key, then deploy it to your remote raspberry pi
- After the deployment you can test your device. When the device test is successful, Go back to kits, select your raspberry kit and change the device to the one you just added.
- You can now create projects and test them

10-ADDING A GENERIC LINUX DEVICE



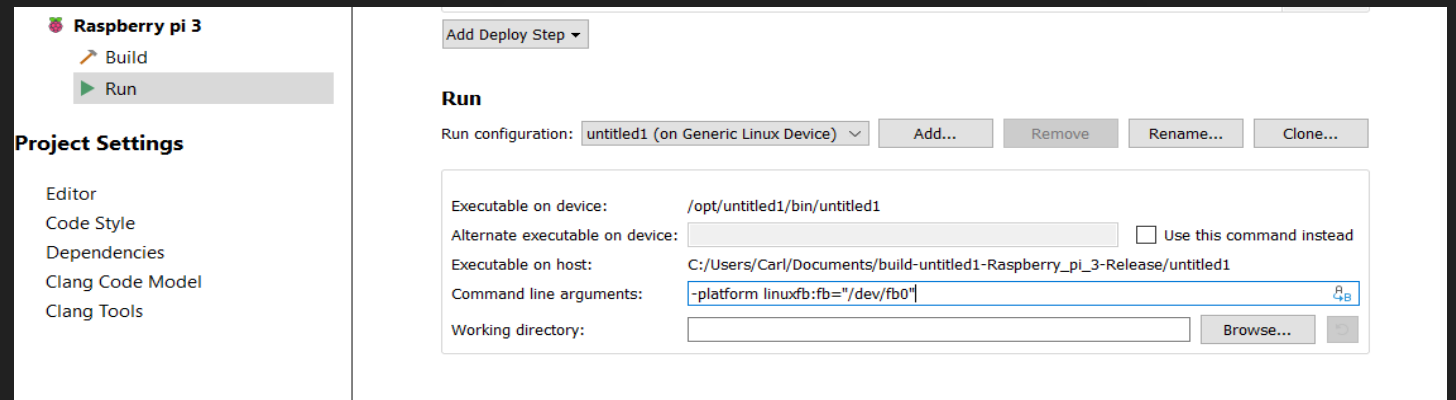
11-TESTING A FIRST PROJECT

- Create a new project
- Select the Raspberry kit especially release option
- In the project file change target.path to `/home/pi`
- Compile and Run the project file. Qt will require you to deploy some libraries
- You will see them in your local `C:\SysGCC\raspberry\arm-linux-gnueabihf\sysroot\lib\arm-linux-gnueabihf` directory. Upload them to remote `/usr/lib/arm-linux-gnueabihf`

 libicudata.so.63	25M
 libicui18n.so	2293K
 libicui18n.so.63	2293K
 libicui18n.so.63.1	2293K
 libicuuc.so.63	1489K

11-TESTING A FIRST PROJECT

- BY default Qt uses EGLFS as plugin to launch Apps.However, EGLFS has not yet been enabled therefore, you will face an Error when Running.
 - You can actively start your app with linuxfb plugin by adding this command in Qt
- Platform linuxfb:fb="/dev/fb0"



12-ENABLING EGLFS

- In your local sysroot copy the vc folder in `\opt\` and paste it in remote `/opt` directory
- Open the shell and navigate to `/opt/vc/lib`
- Create symlinks of EGLFS libraries:
`sudo ln -s libbrcmEGL.so libEGL.so.1`
`sudo ln -s libbrcmGLESv2.so libGLESv2.so.2`
`export LD_LIBRARY_PATH=/opt/vc/lib`
- Go to `/home/pi` , where your program is , and start `ldd your_program`.

12-ENABLING EGLFS

```
pi@192.168.1.75:/opt/vc/lib$ sudo rm libEGL.so.3
pi@192.168.1.75:/opt/vc/lib$ cd ../
pi@192.168.1.75:/opt/vc$ cd lib
pi@192.168.1.75:/opt/vc/lib$ sudo ln -s libbrcmEGL.so libEGL.so.1
pi@192.168.1.75:/opt/vc/lib$ ls
libbcm_host.so      libcontainers.so    libEGL.so          libGLESv2.so       libmmal_components.so  libopenmaxil.so  libvcsn.so
libbrcmEGL.so       libdebug_sym.so     libEGL.so.1        libGLESv2.so.2     libmmal_core.so        libvchiq_arm.so  pkgconfig
libbrcmGLESv2.so    libdebug_sym_static.a  libEGL_static.a    libGLESv2_static.a  libmmal.so             libvchostif.a    plugins
libbrcmOpenVG.so    libdtovl.so          libelftoolchain.so  libkhrn_client.a   libmmal_util.so        libvcilcs.a
libbrcmWFC.so        libEGL.a             libGLESv2.a         libkhrn_static.a   libmmal_vc_client.so   libvcos.so
pi@192.168.1.75:/opt/vc/lib$ export LD_LIBRARY_PATH=/opt/vc/lib
```

12-ENABLING EGLFS

```
pi@192.168.1.75:/$ cd home
pi@192.168.1.75:/home$ cd pi
pi@192.168.1.75:~$ ldd untitled
linux-vdso.so.1 (0x7ef45000)
/usr/lib/arm-linux-gnueabi/libarmmmem-${PLATFORM}.so => /usr/lib/arm-linux-gnueabi/libarmmmem-v7l.so (0x76f92000)
libQt5Quick.so.5 => /usr/local/Qt-5.13.2/lib/libQt5Quick.so.5 (0x76bdd000)
libQt5Gui.so.5 => /usr/local/Qt-5.13.2/lib/libQt5Gui.so.5 (0x766f8000)
libQt5Qml.so.5 => /usr/local/Qt-5.13.2/lib/libQt5Qml.so.5 (0x76304000)
libQt5Network.so.5 => /usr/local/Qt-5.13.2/lib/libQt5Network.so.5 (0x76170000)
libQt5Core.so.5 => /usr/local/Qt-5.13.2/lib/libQt5Core.so.5 (0x75c63000)
libGLSv2.so.2 => /opt/vc/lib/libGLSv2.so.2 (0x75c3e000)
libpthread.so.0 => /lib/arm-linux-gnueabi/libpthread.so.0 (0x75bff000)
libstdc++.so.6 => /lib/arm-linux-gnueabi/libstdc++.so.6 (0x75a77000)
libm.so.6 => /lib/arm-linux-gnueabi/libm.so.6 (0x75a08000)
libgcc_s.so.1 => /lib/arm-linux-gnueabi/libgcc_s.so.1 (0x759db000)
libc.so.6 => /lib/arm-linux-gnueabi/libc.so.6 (0x75887000)
libpng16.so.16 => /lib/arm-linux-gnueabi/libpng16.so.16 (0x75848000)
libz.so.1 => /lib/arm-linux-gnueabi/libz.so.1 (0x75820000)
librt.so.1 => /lib/arm-linux-gnueabi/librt.so.1 (0x75808000)
libdl.so.2 => /lib/arm-linux-gnueabi/libdl.so.2 (0x757f4000)
libcui18n.so.63 => /lib/arm-linux-gnueabi/libcui18n.so.63 (0x755a6000)
libcuc.so.63 => /lib/arm-linux-gnueabi/libcuc.so.63 (0x75420000)
libcudata.so.63 => /lib/arm-linux-gnueabi/libcudata.so.63 (0x73a22000)
/lib/ld-linux-armhf.so.3 (0x76fa7000)
libbrcmEGL.so => /opt/vc/lib/libbrcmEGL.so (0x739e9000)
libbcm_host.so => /opt/vc/lib/libbcm_host.so (0x739c2000)
libvchiq_arm.so => /opt/vc/lib/libvchiq_arm.so (0x739ac000)
libvcos.so => /opt/vc/lib/libvcos.so (0x73993000)
pi@192.168.1.75:~$ ./untitled
Unable to query physical screen size, defaulting to 100 dpi.
To override, set QT_QPA_EGLFS_PHYSICAL_WIDTH and QT_QPA_EGLFS_PHYSICAL_HEIGHT (in millimeters).
QFontDatabase: Cannot find font directory /usr/local/Qt-5.13.2/lib/fonts.
```

As soon as you see `libGLSv2.so.2` and `libbrcmEGL.so` in the ldd result, EGLFS is enabled



THANK YOU!