P2022 : Sunshare
Courjaud Melvin

Mettre en place la compilation croisée

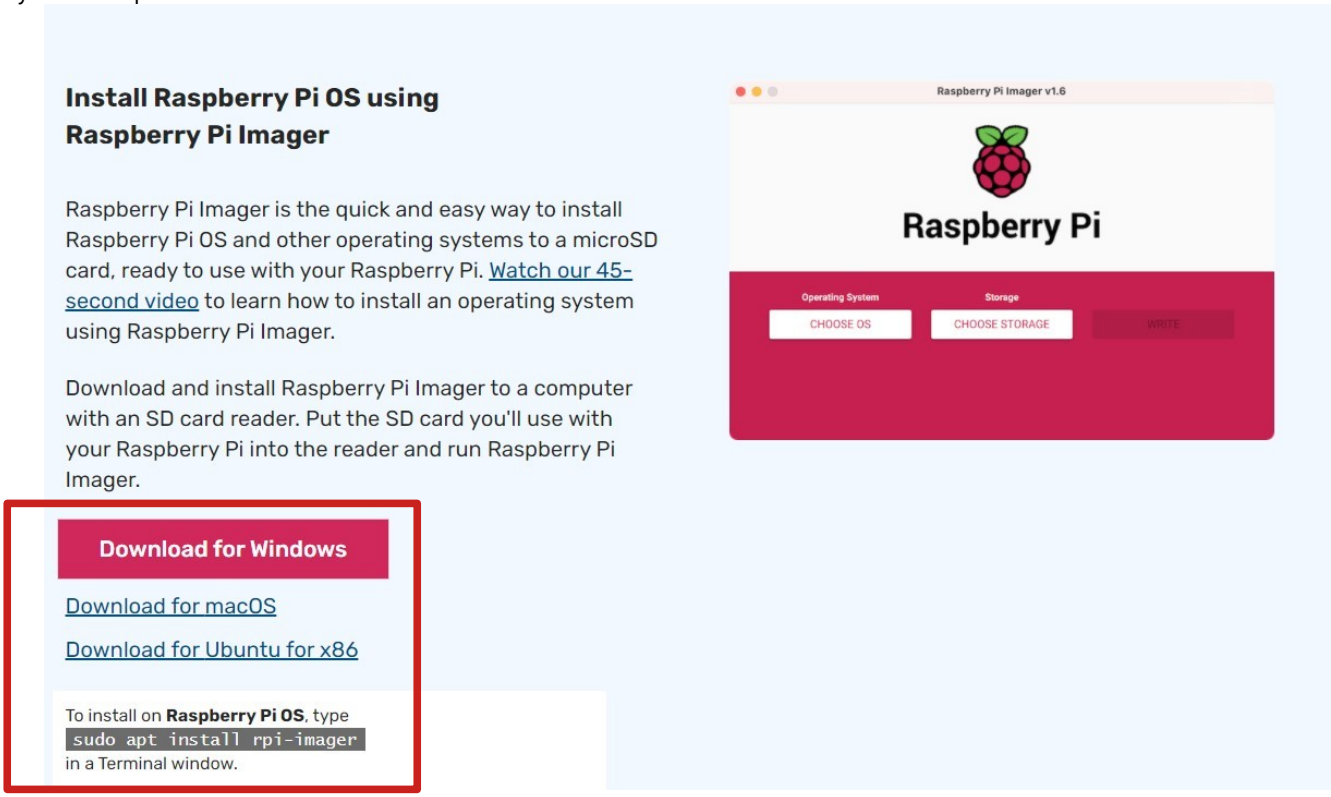
Table des matières

1 - Installation du RPI

1.1 - Préparer la carte SD

Il est recommandé d'utiliser une carte µSD d'au moins 16 Go de stockage.

Se rendre sur le site https://www.raspberrypi.com/software/ pour télécharger l'installateur, vous devez faire le choix en fonction de votre système d'exploitation.

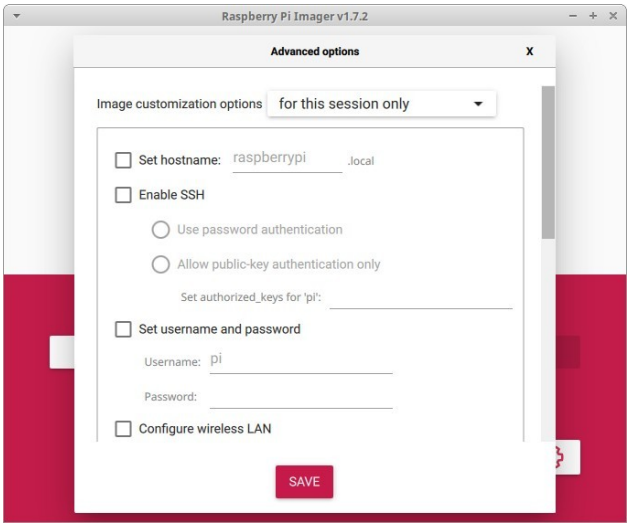


Installer le logiciel qui permet d'installer l'OS sur le Rpi en l'ouvrant à partir de l'explorateur de fichier, une fenêtre s'ouvre, appuyez sur « install ».

Désormais, vous pouvez choisir le système d'exploitation que vous voulez installer, pour les débutants, il est recommandé d'utiliser Raspberry Pi OS, le support sur lequel vous voulez l'installer (**/!\ si vous avez plusieurs appareils connectés en USB comme des clef USB et que vous vous trompez, les données seront perdues!**), puis de cliquer sur « écrire ».

1.1.1 - Nouvelle manière de configurer le RPI

Lorsque vous choisissez un OS, un écrou apparaît sous « écrire ». Vous pouvez alors configurer SSH en passant les parties 1.2 et 1.3 de ce manuel.

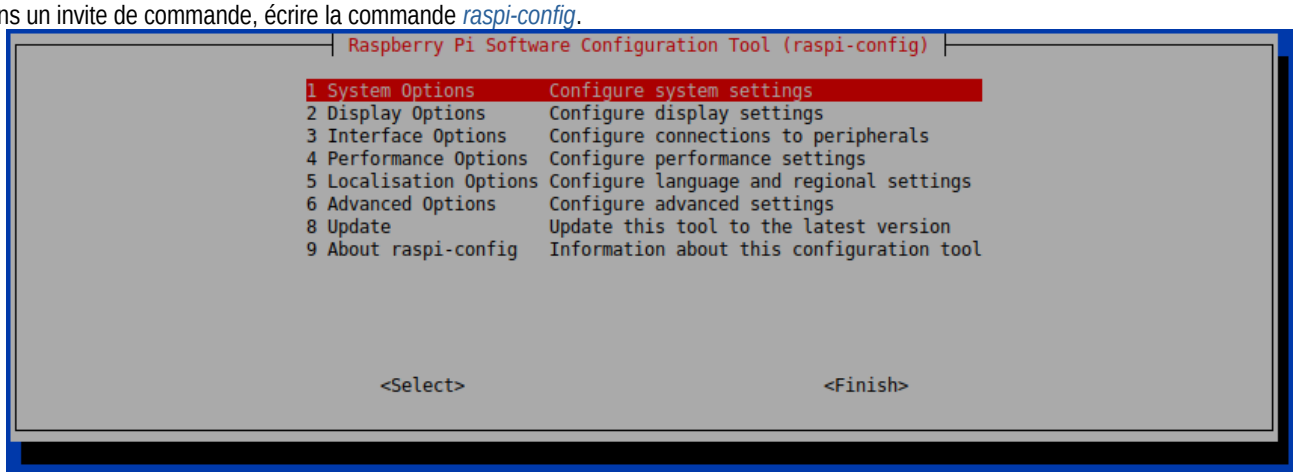


1.2 - Premier démarrage

Insérez la carte µSD dans le Rpi, connectez un clavier, une souris et un écran, puis alimentez le.

Après l'avoir démarré, une interface graphique s'affiche et vous demande votre localisation, un mot de passe et de faire des mises à jour (facultatives).

Dans un invite de commande, écrire la commande *raspi-config*.



Activer le login en console : System options > boot / autologin > console

Activer SSH : interface options > ssh

Quand vous terminé, sélectionnez finish, en bas à droite.

Pour mettre à jour le Rpi en ligne de commande : *apt-get update && apt-get upgrade*

1.3 - Connexion en SSH

pour modifier l'IP du Rpi en statique modifier le fichier : *sudo nano /etc/dhcpcd.conf*

modifier les lignes (les valeurs sont des exemples)

- static ip_addresse=10.0.122.165/27
- static routers=10.0.122.161
- static domain_name_servers=8.8.8.8

Pour se connecter en SSH, il faudra écrire la commande *ssh pi@[IP]*, ici, l'IP sera 10.0.122.165.

1.4 - Installer les bibliothèques

Installer Qt : sudo apt-get install qtbase5-dev-tools qtbase5-dev qtcreator

Installer le driver mysql (base de données) : sudo apt-get install libqt5sql5-mysql

Installer la bibliothèque pigpio

Installer les paquets python nécessaires : sudo apt install python-setuptools python3-setuptools

Compiler la bibliothèque :

```
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip master.zip
cd pigpio-master
make
sudo make install
```

2 - Mettre en place la compilation croisée

Version de Qt utilisée : 5.12.5

PC Linux 64 bit (hôte) => RPI OS 32 bit (cible)

Tutoriel original : https://digitalboxweb.wordpress.com/2020/03/20/qt5-creator-pour-raspberry-pi/

2.1 - Sur le RPI

```
sudo nano /etc/apt/sources.list
```

=> retirer le commentaire « # » à la ligne #deb-src http://raspbian.raspberrypi.org/raspbian buster main contrib non-free rpi

Installer les bibliothèques de développement pour compiler Qt :

```
sudo apt-get update
sudo apt-get build-dep qt4-x11
sudo apt-get build-dep libqt5gui5
sudo apt-get install libudev-dev libinput-dev libts-dev libxcb-xinerama0-dev libxcb-xinerama0
```

Appliquer la mise à jour :

```
sudo rpi-update
reboot
```

Créer un répertoire qui contiendra la version de Qt :

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

2.2 - Sur le PC

2.2.1 - Récupérer les fichier du RPI

Créer un dossier qui contiendra la toolchain officielle pour Rpi :

```
mkdir ~/raspi
cd ~/raspi
git clone https://github.com/raspberrypi/tools
```

Définir un répertoire « sysroot » :

```
mkdir sysroot sysroot/usr sysroot/opt
```

Récupérer les fichiers nécessaires du Rpi (peut prendre plusieurs minutes) :

```
rsync -avz pi@10.0.122.165:/lib sysroot
rsync -avz pi@10.0.122.165:/usr/include sysroot/usr
rsync -avz pi@10.0.122.165:/usr/lib sysroot/usr
```

rsync -avz pi@10.0.122.165:/opt/vc sysroot/opt

Utiliser un script pour ajuster les liens symboliques :

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

2.2.2 - Compilation des sources

Télécharger et extraire les sources Qt 5.15.2 :

wget http://download.qt.io/official_releases/qt/5.12/5.12.5/single/qt-everywhere-src-5.12.5.tar.xz
tar xvf qt-everywhere-src-5.12.5.tar.xz
cd qt-everywhere-src-5.12.5

Lancer la commande permettant d'indiquer la construction de Qt pour la plateforme Raspberry /!\ bien vérifier les **chemins** :

./configure -release -opengl es2 -device linux-rasp-pi-g++ -device-option CROSS_COMPILE=/local/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf- -sysroot /local/raspi/sysroot -opensource -confirm-license -skip qtwayland -skip qtlocation -skip qtscript -make libs -prefix /usr/local/qt5pi -extprefix /local/raspi/qt5pi -hostprefix /local/raspi/qt5 -no-use-gold-linker -v -no-gbm

Si la commande fonctionne, le résultat est le suivant :

```
Building on: linux-g++ (x86_64, CPU features: mmx sse sse2)
Building for: devices/linux-rasp-pi-g++ (arm, CPU features: <none>)
Target compiler: gcc 4.8.3
Configuration: cross_compile compile_examples enable_new_dtags largefile precompile_header shared rpath release c++11
concurrent dbus reduce_exports stl
Build options:
  Mode ..................................... release
  Optimize release build for size ........ no
  Building shared libraries .............. yes
  Using C standard ....................... C11
  Using C++ standard ..................... C++11
  Using ccache ........................... no
  Using gold linker ...................... no
  Using new DTAGS ........................ yes
  Using precompiled headers .............. yes
  Using LTCG ............................. no
  Target compiler supports:
    NEON ................................. no
  Build parts ............................ libs
Qt modules and options:
  Qt Concurrent .......................... yes
  Qt D-Bus ............................... yes
  Qt D-Bus directly linked to libdbus .... yes
  Qt Gui ................................. yes
  Qt Network ............................. yes
  Qt Sql ................................. yes
  Qt Testlib ............................. yes
  Qt Widgets ............................. yes
  Qt Xml ................................. yes
Support enabled for:
  Using pkg-config ....................... yes
  udev ................................... yes
  Using system zlib ...................... yes
Qt Core:
  DoubleConversion ....................... yes
    Using system DoubleConversion ........ yes
  GLib ................................... yes
  iconv .................................. yes
  ICU .................................... no
  Tracing backend ........................ <none>
  Logging backends:
    journald ............................. no
    syslog ............................... no
    slog2 ................................ no
  Using system PCRE2 ..................... yes
Qt Network:
  getifaddrs() ........................... yes
  IPv6 ifname ............................ yes
  libproxy ............................... no
  Linux AF_NETLINK ....................... yes
  OpenSSL ................................ yes
    Qt directly linked to OpenSSL ........ no
  OpenSSL 1.1 ............................ yes
  DTLS ................................... yes
  SCTP ................................... no
  Use system proxies ..................... yes

Qt Sql:
```

```
  SQL item models ....................... yes
Qt Widgets:
  GTK+ .................................. no
  Styles ................................ Fusion Windows
Qt PrintSupport:
  CUPS .................................. yes
Qt Sql Drivers:
  DB2 (IBM) ............................. no
  InterBase ............................. no
  MySql ................................. no
  OCI (Oracle) .......................... no
  ODBC .................................. yes
  PostgreSQL ............................ yes
  SQLite2 ............................... no
  SQLite ................................ yes
    Using system provided SQLite ......... no
  TDS (Sybase) .......................... yes
Qt Testlib:
  Tester for item models ................. yes
Qt SerialBus:
  Socket CAN ............................ yes
  Socket CAN FD ......................... yes
Further Image Formats:
  JasPer ................................ no
  MNG ................................... no
  TIFF .................................. yes
    Using system libtiff ................. yes
  WEBP .................................. yes
    Using system libwebp ................. no
```

Il faut ensuite compiler Qt (très long, prévoir 2-3h) :

make -j4
sudo make install

Copier le dossier « qt5pi » généré dans le dossier raspi :

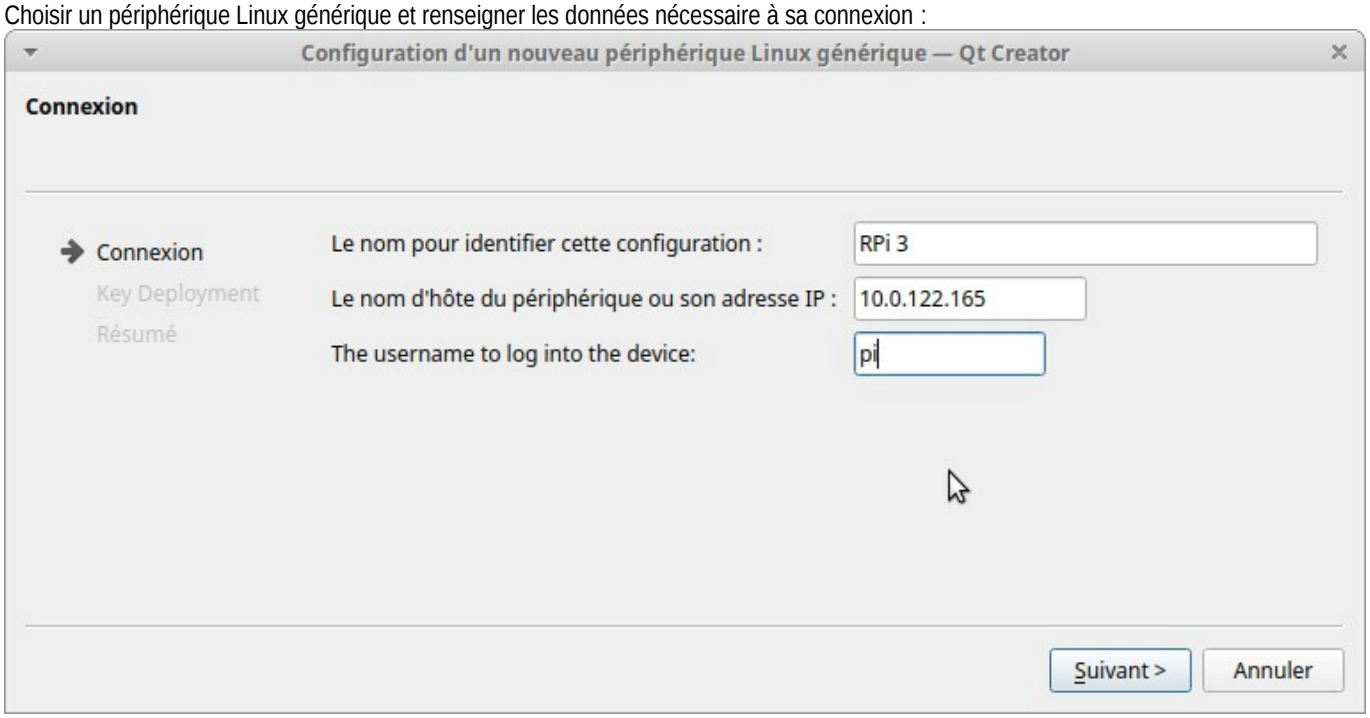
cd ~/raspi
rsync -avz qt5pi pi@ip_locale:/usr/local

2.3 - Configuration de Qt Creator

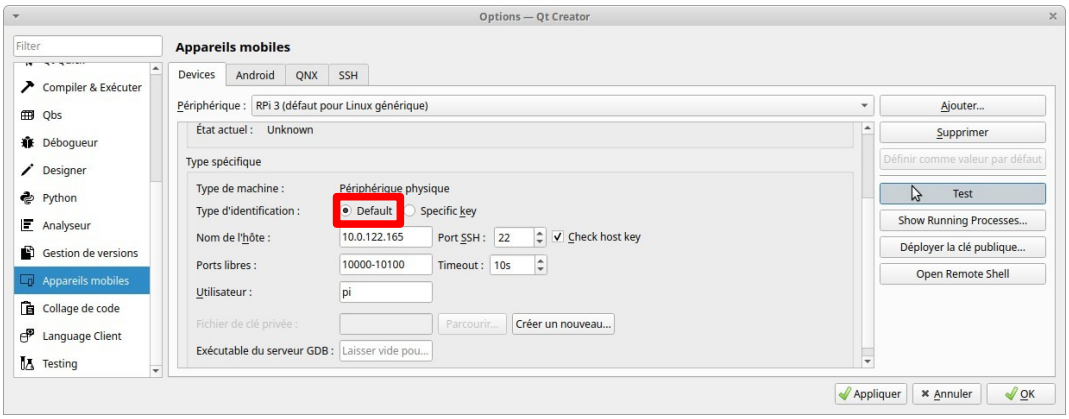
Installer Qt Creator sur l'ordinateur à l'aide de l'installeur, il faudra créer un compte, choisir les composants suivants :

Dans Outils>Options, connecter le Rpi au PC comme ceci :

Choisir un périphérique Linux générique et renseigner les données nécessaire à sa connexion :



Testez ensuite la connexion :



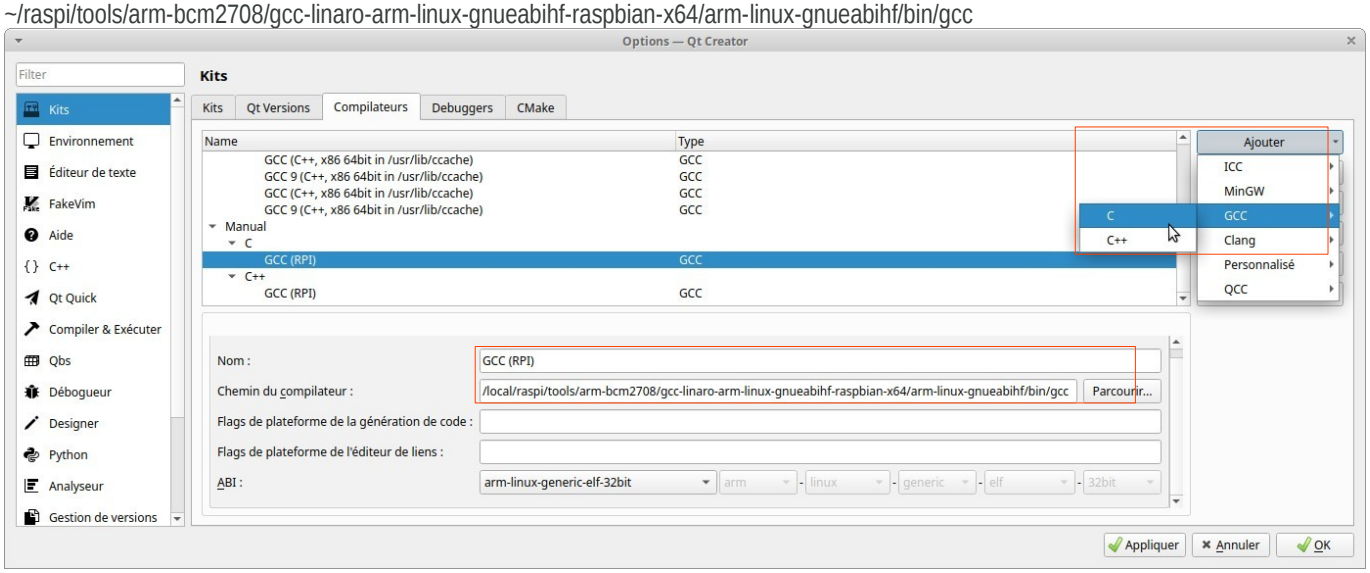
Il faut maintenant renseigner le mdp du Rpi, si le test réussi, il affiche le texte suivant :

```
Connexion à l'hôte...
Verification de la version du noyau...
Linux 5.10.90-v7+ armv7l

Vérification si les ports spécifiés sont disponibles...
Tous les ports spécifiés sont disponibles.

Checking whether an SFTP connection can be set up...
SFTP service available.

Checking whether rsync works...
rsync is functional.

Le test du périphérique s'est terminé avec succès.
```

Il faut maintenant configurer le compilateur de Qt Creator pour qu'il compile pour le raspberry pi et non le PC :

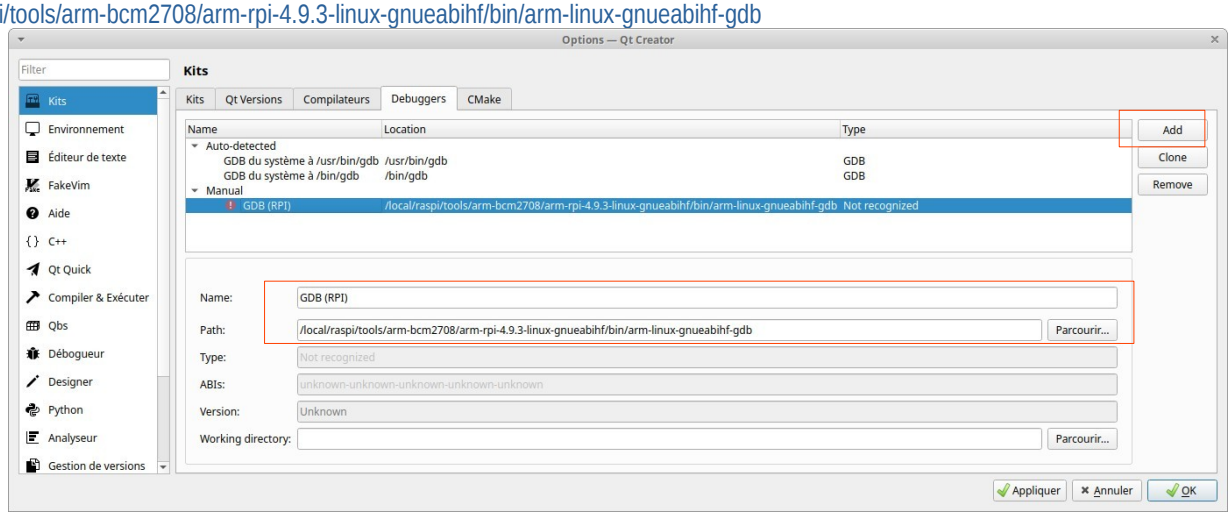
Pour les compilateurs (mettre C et C++), chemin à copier coller :

~/raspi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/arm-linux-gnueabihf/bin/gcc

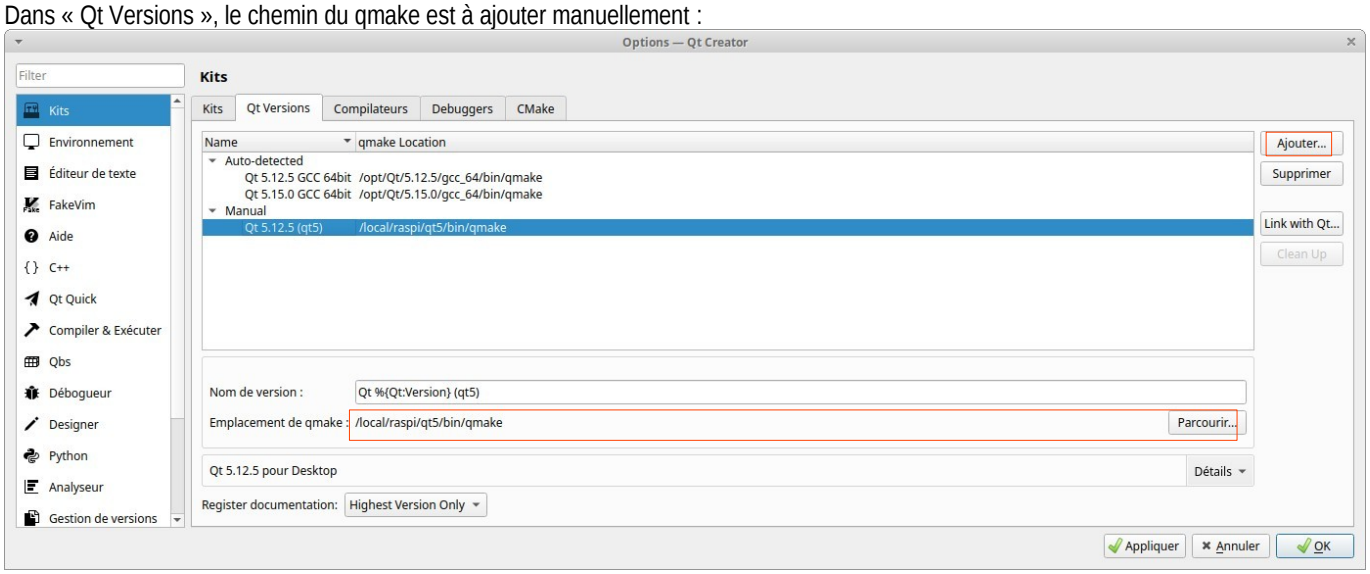


Pour le debugger :

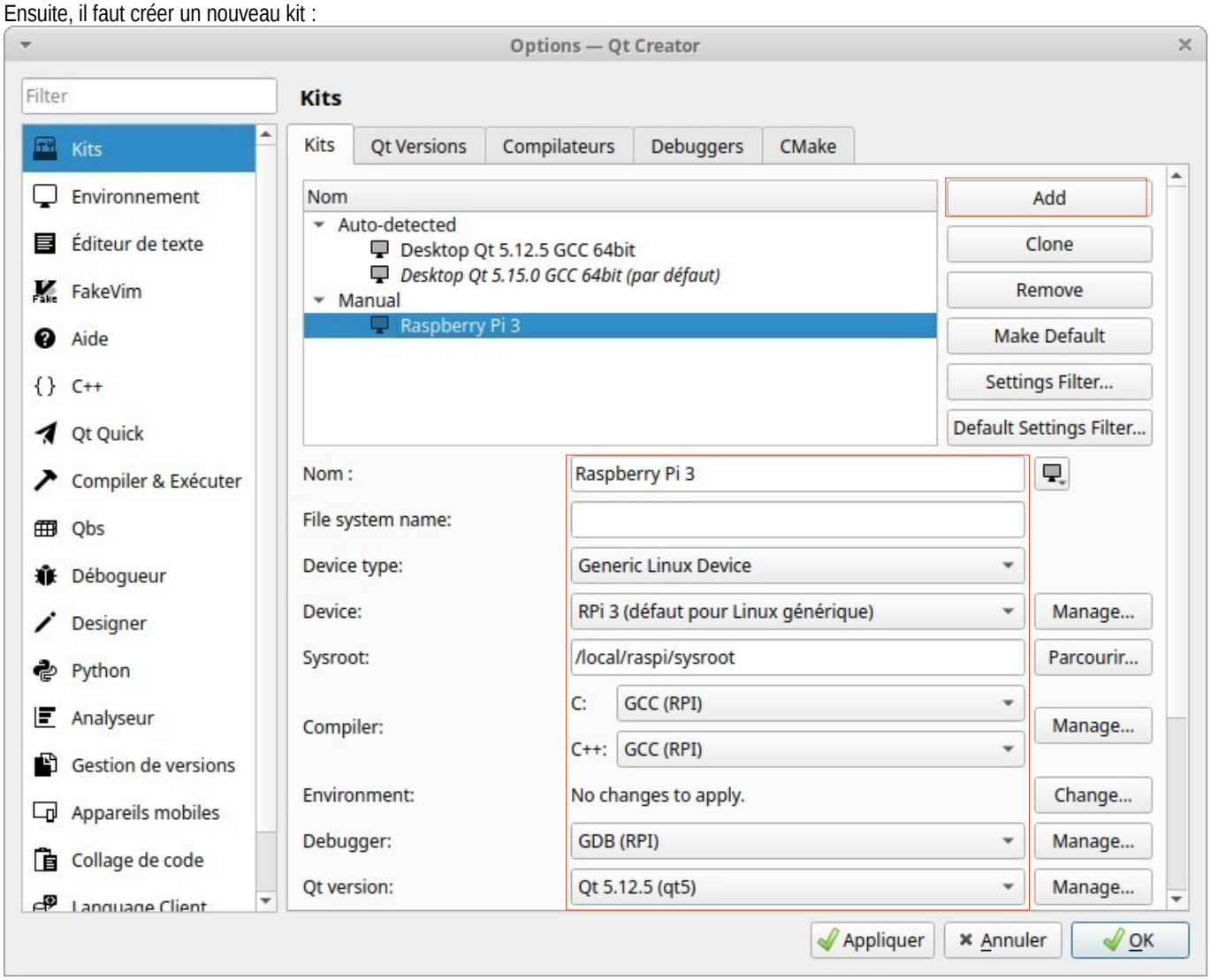
~/raspi/tools/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf/bin/arm-linux-gnueabihf-gdb



Dans « Qt Versions », le chemin du qmake est à ajouter manuellement :

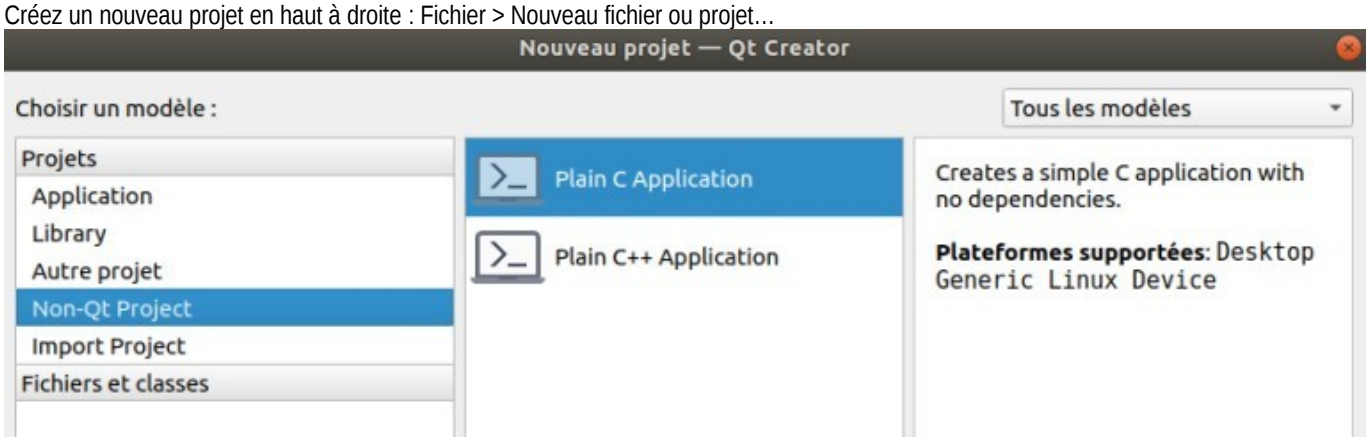


Ensuite, il faut créer un nouveau kit :

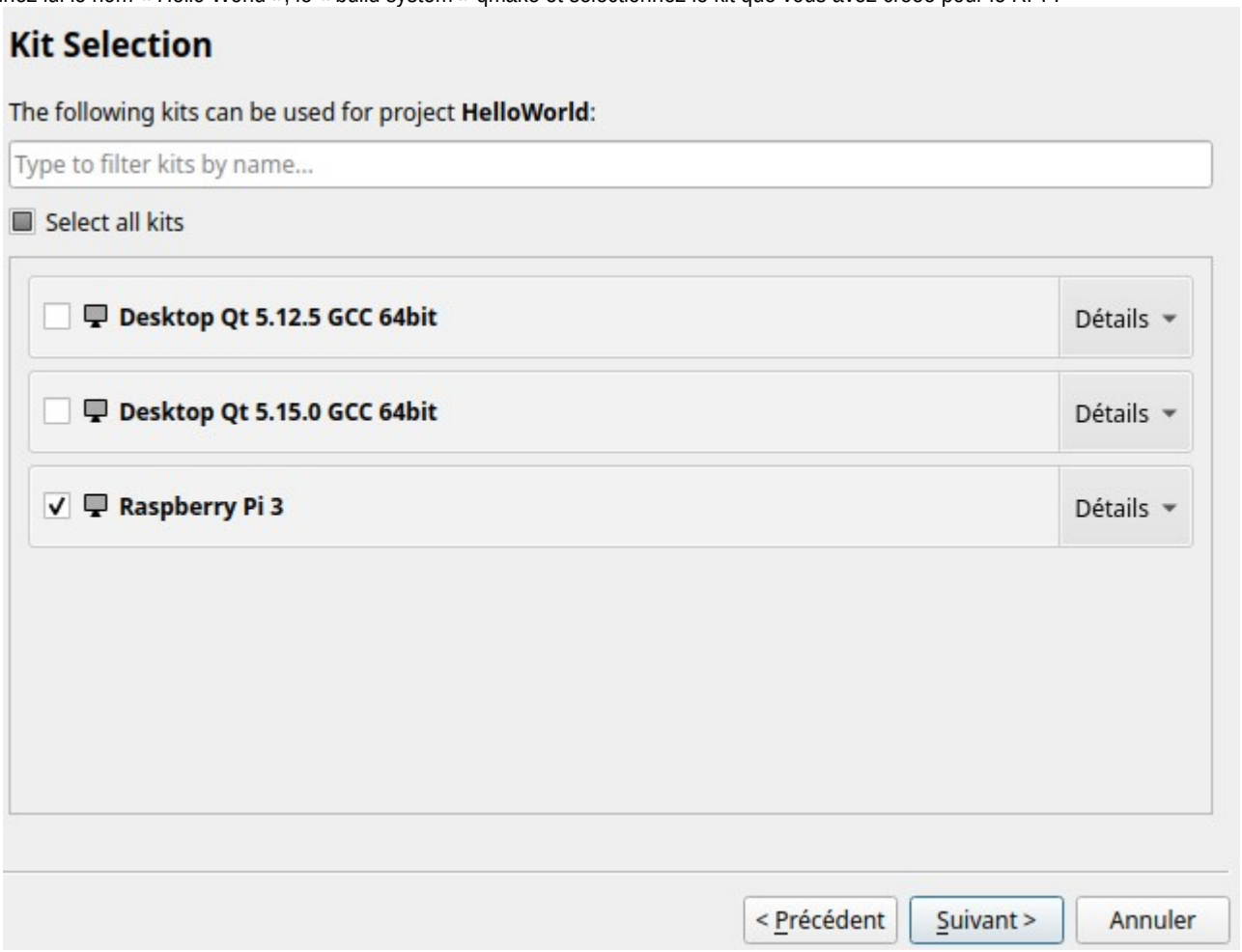


2.4 - Tester la compilation croisée

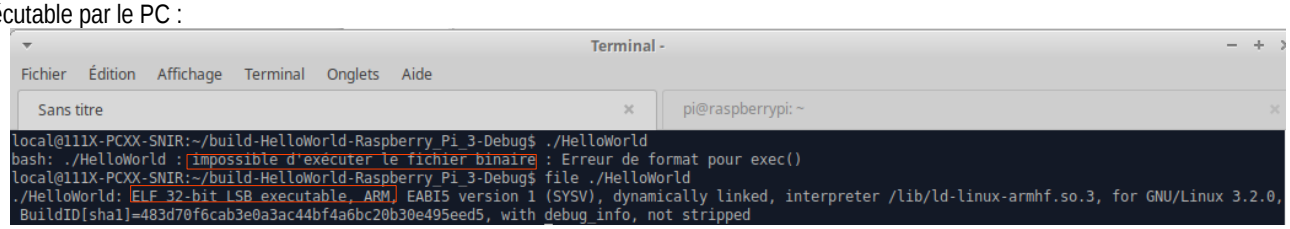
Créez un nouveau projet en haut à droite : Fichier > Nouveau fichier ou projet...



Donnez lui le nom « Hello World », le « build system » qmake et sélectionnez le kit que vous avez créée pour le RPI :



Lancez ensuite la compilation du projet (Ctrl + B) et allez ensuite dans le répertoire de compilation pour tester le fichier ; il n'est pas exécutable par le PC :



Pour tester le programme sur la Rpi, il faut envoyer le fichier sur l'appareil :

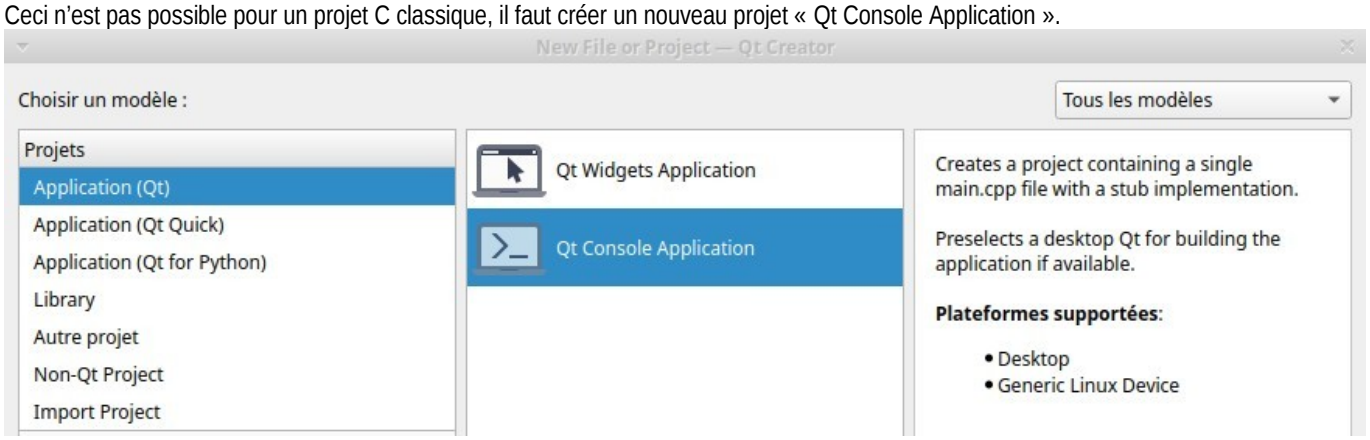
rsync ./HelloWorld pi@10.0.122.165:.

Puis le tester, le programme fonctionne :

```
pi@raspberrypi:~ $ ./HelloWorld
Hello World!
pi@raspberrypi:~ $ file ./HelloWorld
./HelloWorld: 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]=483d70f6cab3e0a3ac44bf4a6bc20b30e495eed5, with debug_info, not stripped
```

2.5 - Synchroniser la compilation

Ceci n'est pas possible pour un projet C classique, il faut créer un nouveau projet « Qt Console Application ».

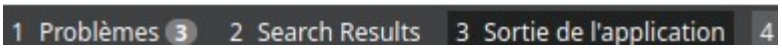


Utiliser les mêmes paramètres que pour le projet précédent, et utiliser le code suivant :

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

int main()
{
    cout << "Hello World!\n";

    return 0;
}
```

Exécutez et déployez le code (crtl + r), un mot de passe pour la connexion ssh vous sera peut-être demandé, le programme se lance sans problème :

```
10:50:17: Starting /opt/HelloWorld/bin/HelloWorld ...
Hello World!
10:50:17: Application finished with exit code 0.
```



Il faut maintenant localiser où l'exécutable est créée, il faut aller dans le .pro :

```
18 # Default rules for deployment.
19 qnx: target.path = /tmp/$${TARGET}/bin
20 else: unix:!android: target.path = /opt/$${TARGET}/bin
21 !isEmpty(target.path): INSTALLS += target
```

Ici, le chemin est /opt/{NOM_PRJET}/bin du Rpi.

On peut exécuter le programme directement depuis le Rpi et le résultat est le même :

```
pi@raspberrypi:/opt/HelloWorld/bin $ ls
HelloWorld
pi@raspberrypi:/opt/HelloWorld/bin $ ./HelloWorld
Hello World!
```

2.5.1 - Ajouter la bibliothèque QserialPort

Dans le .pro, ajouter : QT += serialport

Dans le .cpp, ajouter : #include <QSerialPort>

Dans le main :

```
QSerialPort portserie ("Qserial port !");
cout << portserie.portName().toStdString() << endl;
```

```
11:12:47: Starting /opt/Qserial/bin/Qserial ...
Qserial port !
11:12:47: Application finished with exit code 0.
```

Le programme fonctionne.

2.5.2 - Ajouter la bibliothèque pigpio

La bibliothèque doit être installée à la fois sur le PC et le RPI.

Rappel d'installation :

```
sudo apt install python-setuptools python3-setuptools
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip master.zip
cd pigpio-master
make
sudo make install
```

Dans un nouveau projet, ajouter dans le .pro :

```
INCLUDEPATH += /local/pigpio-master/
LIBS += -lpigpio -lrt -lpthread
```

Dans le .cpp : #include <pigpio.h>

Dans le main :

```
if (gpioInitialise() < 0)
  {
    cout << "pigpio initialisation failed\n";
  }
else
{
   cout << "pigpio initialisation passed\n";
}
gpioTerminate();
```

Lorsque vous compilez depuis Qt Creator, vous avez une erreur de permission :

```
11:21:43: Starting /opt/Qserial/bin/Qserial ...
pigpio initialisation failed
2022-01-25 11:21:43 initCheckPermitted:
+-----------------------------------------------------------+
|Sorry, you don't have permission to run this program.     |
|Try running as root, e.g. precede the command with sudo. |
+-----------------------------------------------------------+


11:21:43: Application finished with exit code 0.
```

Et lorsque l'on essaie d'exécuter directement depuis le RPI :

```
pi@raspberrypi:/opt/Qserial/bin $ ./Qserial
2022-01-25 11:23:38 initCheckPermitted:
+-----------------------------------------------------+
|Sorry, you don't have permission to run this program.    |
|Try running as root, e.g. precede the command with sudo. |
+-----------------------------------------------------+


pigpio initialisation failed
pi@raspberrypi:/opt/Qserial/bin $ sudo ./Qserial
pigpio initialisation passed
```

Il faudra donc exécuter le programme depuis l'invite de commande du RPI.

2.5.3 - Compiler le driver mysql

En compilation croisée, le driver n'est compatible qu'avec certaines versions de Qt, c'est pourquoi je n'ai pas réussi à la mettre en place.

Lien d'un forum le mettant en place : https://forum.qt.io/topic/116742/qmysql-driver-not-loaded-cross-compiling-for-raspberry-pi-3/26