Дальше нужно установить:

1. Python последней версии 64 битную

2. Perl последней версии 64 битную

3. Ruby последней версии 64 битную

configure -static -debug-and-release -platform win32-g++ -qt-zlib -qt-pcre -qt-libpng -qt-libjpeg -qt-freetype -opengl desktop -no-angle -sql-sqlite -openssl -I «c:\Qt\Tools\OpenSSL\Win\_x64\include» -L «c:\Qt\Tools\OpenSSL\Win\_x64» -opensource -confirm-license -make libs -make tools -nomake examples -nomake tests -prefix c:\Qt\5.14.2\mingw73\_64\_static

mingw32-make -k -j4

mingw32-make -k install

И чтоб отлучить об зависимостей MinGW в проект добавляем QMAKE\_LFLAGS\_RELEASE += -static -static-libgcc

Статическая сборка Qt 5.12 из исходников под Windows

Posted by [Pavelk](http://pavelk.ru/author/admin),15 августа 2019[Qt](http://pavelk.ru/category/%d0%bf%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%bc%d0%b8%d1%80%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5/qt), [Новости](http://pavelk.ru/category/news)[16 комментариев](http://pavelk.ru/staticheskaya-sborka-qt-5-12-iz-isxodnikov-pod-windows#comments)1Views :

3985

[Print](http://pavelk.ru/staticheskaya-sborka-qt-5-12-iz-isxodnikov-pod-windows)

Потребовалось мне тут, что бы Qt приложение было в одном exe.  Для этого придётся перекомпилировать Qt.

Качаем: [Git](https://github.com/git-for-windows/git/releases/latest)

Качаем: [Perl](http://strawberryperl.com/), версию 32х битную.

Качаем: [Python 2](https://www.python.org/downloads/) (У меня 2.7).

Качаем: [MinGW](https://download.qt.io/online/qtsdkrepository/windows_x86/desktop/tools_mingw/qt.tools.win32_mingw730/7.3.0-0i686-7.3.0-release-posix-dwarf-rt_v5-rev0.7z), y меня 730 (для распаковки нужен [7Zip](https://sourceforge.net/projects/sevenzip/files/latest/download), запускать «C:\Programm Files (x86)\7-Zip\7zFM.exe»).

Открываем консольку (Win+R, вводим cmd), прописываем необходимые системные переменные (пути замените на свои):

set PATH=C:\MinGW\mingw730\_32\bin\;%PATH%

set PATH=C:\Strawberry\perl\bin\;%PATH%

Переходим в папку, в которой будем развлекаться, качаем исходники Qt:

cd C:/QtStatic

git clone https://github.com/qt/qt5

cd qt5

perl init-repository

Конфигурируем:

configure -opensource -confirm-license -platform win32-g++ -static -release -opengl desktop -qt-zlib -qt-libpng -qt-libjpeg -nomake examples

В конфиге не включён SSL, кому нужно:  
Качаем и ставим [Win32OpenSSLLite](http://slproweb.com/products/Win32OpenSSL.html)  
-openssl -I «c:\Qt\3dparty\openssl-1.0.2m\include» -L «c:\Qt\3dparty\openssl-1.0.2m»

Если вдруг нужно переконфигурировать и пересобрать, то нужно сначала очистить результаты предыдущих потуг:

git submodule foreach --recursive "git clean -dfx"

git clean -dfx

Собираем (Часа 4) и ставим:

mingw32-make -j4

mingw32-make install

После установки настраиваем QtCreator для использования новой версии фреймворка:

В верхней менюшке Инструменты -> Параметры -> Сборка и запуск

Компиляторы -> Добавить  
MinGW -> C++  
Название: Qt 5.12 Static MinGW  
Путь: C:\MinGW\mingw730\_32\bin\c++.exe

Отладчики -> Добавить  
Название: Qt 5.12 Static DBG  
Путь: C:\MinGW\mingw730\_32\bin\gdb.exe

Профили Qt -> Добавить  
Название: Qt 5.12 Static  
Путь: C:\QtStatic\qt5\qtbase\bin\qmake.exe

Комплекты -> Добавить  
Название: Qt 5.12 Static  
Тип устройства: Desktop  
Компилятор C++: Qt 5.12 Static MinGW  
Отладчик: Qt 5.12 Static DBG  
Профиль Qt: Qt 5.12 Static

 Нажимаем «Применить».

Можно попробовать сделать простенький проект. В параметрах сборки и запуска поставьте «Release» т.к. мы только эту версию собрали. Так же в «\*.pro» файл необходимо добавить указание на статическую сборку и линковку libGCC.

**\*.pro**

QMAKE\_LFLAGS += -static -static-libgcc

# QtCreator: Qt кросс-компиляция из linux 64 в linux 32, win32, win64 и Mac OS X; upx, usb, dmg, etc

* [Программирование](https://habr.com/ru/hub/programming/),
* [Qt](https://habr.com/ru/hub/qt_software/)

Библиотека [Qt](http://qt-project.org/) позволяет делать действительно кроссплатформенные приложения. Единожды написанный код можно откомпилировать под многие операционные системы. Но проблема именно в слове «компилировать», т.к. подразумевается, что необходимо перезагрузиться под целевую систему, иметь в ней настроенную среду разработки, установленный и настроенный зоопарк библиотек. Спасает кросс-компиляция — компиляция, производящая исполняемый код для платформы, отличной от той, на которой исполняется.

##### **Кросс-компиляция для Windows 64**

Обычно одной из наиболее востребованных проблем является сборка Windows-версии своего приложения, изначально разрабатывающегося под Linux. Пример решения этой проблемы можно увидеть [тут](http://silmor.de/qtstuff.cross.php) или [на русском](http://qtlinux.narod.ru/make_for_win.htm). Необходимо создать mkspecs-конфигурацию, положить файлы Qt в соответствующие директории и всё. Компилировать Qt в таком случае не обязательно, можно скачать бинарники с официального сайта.  
У такого подхода есть несколько минусов: 1) QtCreator об установленной таким образом библиотеке ничего не знает; 2) Официальной сборки Qt для Windows x64 не существует. И если с первой проблемой ещё как-то можно бороться, то против второй поможет только компиляция…  
  
Перед кросс-компиляцией не забудьте поставить непосредственно сам кросс-компилятор (ищется в пакетом менеджере по названию «mingw»). И скачать исходники qt-everywhere с официального сайта. В директории *mkspecs* распакованного архива копируем папку *win32-g++* в *win64-x-g++* и корректируем содержимое файла *qmake.conf*. У меня получилось следующее:

Скрытый текст

*#*

*# qmake configuration for win64-x-g++*

*#*

*# Written for MinGW*

*#*

*# Cross compile example for i686-w64-mingw32-g++:*

*# configure -xplatform win32-g++ -device-option CROSS\_COMPILE=i686-w64-mingw32-*

*#*

MAKEFILE\_GENERATOR = MINGW

load(device\_config)

equals(QMAKE\_HOST.os, Windows): EXE\_SUFFIX = .exe

TEMPLATE = app

CONFIG += qt warn\_on release link\_prl copy\_dir\_files debug\_and\_release debug\_and\_release\_target precompile\_header

QT += core gui

DEFINES += UNICODE

*#QT\_LARGEFILE\_SUPPORT*

QMAKE\_COMPILER\_DEFINES += \_\_GNUC\_\_ WIN WIN32 WIN64

QMAKE\_EXT\_OBJ = .o

QMAKE\_EXT\_RES = \_res.o

QMAKE\_CC = $${CROSS\_COMPILE}gcc

QMAKE\_LEX = flex

QMAKE\_LEXFLAGS =

QMAKE\_YACC = byacc

QMAKE\_YACCFLAGS = -d

QMAKE\_CFLAGS = -pipe -m64

QMAKE\_CFLAGS\_DEPS = -M

QMAKE\_CFLAGS\_WARN\_ON = -Wall -Wextra

QMAKE\_CFLAGS\_WARN\_OFF = -w

QMAKE\_CFLAGS\_RELEASE = -O2

QMAKE\_CFLAGS\_DEBUG = -g

QMAKE\_CFLAGS\_YACC = -Wno-unused -Wno-parentheses

QMAKE\_CXX = $${CROSS\_COMPILE}g++

QMAKE\_CXXFLAGS = $$QMAKE\_CFLAGS

QMAKE\_CXXFLAGS\_DEPS = $$QMAKE\_CFLAGS\_DEPS

QMAKE\_CXXFLAGS\_WARN\_ON = $$QMAKE\_CFLAGS\_WARN\_ON

QMAKE\_CXXFLAGS\_WARN\_OFF = $$QMAKE\_CFLAGS\_WARN\_OFF

QMAKE\_CXXFLAGS\_RELEASE = $$QMAKE\_CFLAGS\_RELEASE

QMAKE\_CXXFLAGS\_DEBUG = $$QMAKE\_CFLAGS\_DEBUG

QMAKE\_CXXFLAGS\_YACC = $$QMAKE\_CFLAGS\_YACC

QMAKE\_CXXFLAGS\_THREAD = $$QMAKE\_CFLAGS\_THREAD

QMAKE\_CXXFLAGS\_RTTI\_ON = -frtti

QMAKE\_CXXFLAGS\_RTTI\_OFF = -fno-rtti

QMAKE\_CXXFLAGS\_EXCEPTIONS\_ON = -fexceptions -mthreads

QMAKE\_CXXFLAGS\_EXCEPTIONS\_OFF = -fno-exceptions

QMAKE\_INCDIR = /usr/x86\_64-w64-mingw32/include

QMAKE\_INCDIR\_QT = $$[QT\_INSTALL\_HEADERS]

QMAKE\_LIBDIR\_QT = $$[QT\_INSTALL\_LIBS]

QMAKE\_RUN\_CC = $(CC) -c $(CFLAGS) $(INCPATH) -o $obj $src

QMAKE\_RUN\_CC\_IMP = $(CC) -c $(CFLAGS) $(INCPATH) -o $@ $<

QMAKE\_RUN\_CXX = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $obj $src

QMAKE\_RUN\_CXX\_IMP = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

QMAKE\_LINK = $${CROSS\_COMPILE}g++

QMAKE\_LINK\_C = $${CROSS\_COMPILE}gcc

*#QMAKE\_LFLAGS = -m64*

QMAKE\_LFLAGS = -m64 -mthreads -static-libgcc -static -Wl,-enable-stdcall-fixup -Wl,-enable-auto-import -Wl,-enable-runtime-pseudo-reloc -mwindows

QMAKE\_LFLAGS\_EXCEPTIONS\_ON = -mthreads

QMAKE\_LFLAGS\_EXCEPTIONS\_OFF =

QMAKE\_LFLAGS\_RELEASE = -Wl,-s

QMAKE\_LFLAGS\_DEBUG =

QMAKE\_LFLAGS\_CONSOLE = -Wl,-subsystem,console

QMAKE\_LFLAGS\_WINDOWS = -Wl,-subsystem,windows

QMAKE\_LFLAGS\_DLL = -shared

QMAKE\_LINK\_OBJECT\_MAX = 10

QMAKE\_LINK\_OBJECT\_SCRIPT= object\_script

QMAKE\_PREFIX\_STATICLIB = lib

QMAKE\_EXTENSION\_STATICLIB = a

QMAKE\_LIBS =

QMAKE\_LIBS\_CORE = -lole32 -luuid -lws2\_32 -ladvapi32 -lshell32 -luser32 -lkernel32

QMAKE\_LIBS\_GUI = -lgdi32 -lcomdlg32 -loleaut32 -limm32 -lwinmm -lwinspool -lws2\_32 -lole32 -luuid -luser32 -ladvapi32

QMAKE\_LIBS\_NETWORK = -lws2\_32

QMAKE\_LIBS\_OPENGL = -lglu32 -lopengl32 -lgdi32 -luser32

QMAKE\_LIBS\_COMPAT = -ladvapi32 -lshell32 -lcomdlg32 -luser32 -lgdi32 -lws2\_32

QMAKE\_LIBS\_QT\_ENTRY = -lmingw32 -lqtmain

!isEmpty(QMAKE\_SH) {

MINGW\_IN\_SHELL = 1

QMAKE\_DIR\_SEP = /

QMAKE\_QMAKE ~= s,\\\\,/,

QMAKE\_COPY = cp

QMAKE\_COPY\_DIR = cp -r

QMAKE\_MOVE = mv

QMAKE\_DEL\_FILE = rm

QMAKE\_MKDIR = mkdir -p

QMAKE\_DEL\_DIR = rmdir

QMAKE\_CHK\_DIR\_EXISTS = test -d

} else {

QMAKE\_COPY = copy /y

QMAKE\_COPY\_DIR = xcopy /s /q /y /i

QMAKE\_MOVE = move

QMAKE\_DEL\_FILE = del

QMAKE\_MKDIR = mkdir

QMAKE\_DEL\_DIR = rmdir

QMAKE\_CHK\_DIR\_EXISTS = if not exist

}

QMAKE\_MOC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}moc$${EXE\_SUFFIX}

QMAKE\_UIC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}uic$${EXE\_SUFFIX}

QMAKE\_IDC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}idc$${EXE\_SUFFIX}

QMAKE\_IDL = midl

QMAKE\_LIB = $${CROSS\_COMPILE}ar -ru

QMAKE\_RC = $${CROSS\_COMPILE}windres

QMAKE\_ZIP = zip -r -9

QMAKE\_STRIP = $${CROSS\_COMPILE}strip

QMAKE\_STRIPFLAGS\_LIB += --strip-unneeded

load(qt\_config)

По сути в файле спецификации были заменены только пути.  
  
Я выполнял *configure* со следующими параметрами:  
./configure -xplatform win64-x-g++ CROSS\_COMPILE=x86\_64-w64-mingw32- -prefix /usr/local/qt4win64 -no-webkit -no-phonon -no-phonon-backend -no-script -no-scripttools -no-multimedia -no-qt3support -fast -nomake demos -nomake examples -nomake tools -device-option -little-endian -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked -no-fontconfig -no-3dnow -no-ssse3 -continue  
Здесь собираю минимальную версию Qt без webkit, phonon, multimedia и т.п. Полный список опций можно посмотреть по команде *./configure --help*  
  
Соответственно, для такой сборки должен быть установлен пакет *g++-mingw-w64-x86-64*, содержащий в себе *x86\_64-w64-mingw32-***g++** (в убунту пакет надо ставить отдельно).  
  
Далее *make && sudo make install*. На первом этапе компиляции используется родной системный компилятор, он собирает необходимые утилиты для linux, которые будут использоваться для сборки уже windows-бинарников.  
После установки у меня в */usr/local/qt4win64/bin* лежат PE32+ DLL и несколько ELF 64-bit LSB executable, в том числе: *qmake, uic, moc, rcc*. Вот они то и пригодятся для QtCreator!  
  
После установки **не** удаляйте распакованную директорию — она используется.

##### **Кросс-компиляция для Windows 32**

Аналогична компиляции для Win64. За исключением того, что есть официальная сборка, и саму библиотеку компилировать не нужно! Достаточно собрать *qmake, uic, moc, rcc*.  
  
configure:

Скрытый текст

./configure -xplatform win32-x-g++ -prefix /usr/local/qt4win32 CROSS\_COMPILE=i686-w64-mingw32- -no-webkit -no-phonon -no-phonon-backend -no-script -no-scripttools -no-multimedia -no-qt3support -fast -nomake demos -nomake examples -nomake tools -device-option -little-endian -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked -no-fontconfig -no-3dnow -no-ssse3 -continue

qmake.conf

Скрытый текст

*#*

*# qmake configuration for win32-x-g++*

*#*

*# Written for MinGW*

*#*

*# Cross compile example for i686-w64-mingw32-g++:*

*# configure -xplatform win32-g++ -device-option CROSS\_COMPILE=i686-w64-mingw32-*

*#*

MAKEFILE\_GENERATOR = MINGW

load(device\_config)

equals(QMAKE\_HOST.os, Windows): EXE\_SUFFIX = .exe

TEMPLATE = app

CONFIG += qt warn\_on release link\_prl copy\_dir\_files debug\_and\_release debug\_and\_release\_target precompile\_header

QT += core gui

DEFINES += UNICODE

*#QT\_LARGEFILE\_SUPPORT*

QMAKE\_COMPILER\_DEFINES += \_\_GNUC\_\_ WIN WIN32

QMAKE\_EXT\_OBJ = .o

QMAKE\_EXT\_RES = \_res.o

QMAKE\_CC = $${CROSS\_COMPILE}gcc

QMAKE\_LEX = flex

QMAKE\_LEXFLAGS =

QMAKE\_YACC = byacc

QMAKE\_YACCFLAGS = -d

QMAKE\_CFLAGS = -pipe -m32

QMAKE\_CFLAGS\_DEPS = -M

QMAKE\_CFLAGS\_WARN\_ON = -Wall -Wextra

QMAKE\_CFLAGS\_WARN\_OFF = -w

QMAKE\_CFLAGS\_RELEASE = -O2

QMAKE\_CFLAGS\_DEBUG = -g

QMAKE\_CFLAGS\_YACC = -Wno-unused -Wno-parentheses

QMAKE\_CXX = $${CROSS\_COMPILE}g++

QMAKE\_CXXFLAGS = $$QMAKE\_CFLAGS

QMAKE\_CXXFLAGS\_DEPS = $$QMAKE\_CFLAGS\_DEPS

QMAKE\_CXXFLAGS\_WARN\_ON = $$QMAKE\_CFLAGS\_WARN\_ON

QMAKE\_CXXFLAGS\_WARN\_OFF = $$QMAKE\_CFLAGS\_WARN\_OFF

QMAKE\_CXXFLAGS\_RELEASE = $$QMAKE\_CFLAGS\_RELEASE

QMAKE\_CXXFLAGS\_DEBUG = $$QMAKE\_CFLAGS\_DEBUG

QMAKE\_CXXFLAGS\_YACC = $$QMAKE\_CFLAGS\_YACC

QMAKE\_CXXFLAGS\_THREAD = $$QMAKE\_CFLAGS\_THREAD

QMAKE\_CXXFLAGS\_RTTI\_ON = -frtti

QMAKE\_CXXFLAGS\_RTTI\_OFF = -fno-rtti

QMAKE\_CXXFLAGS\_EXCEPTIONS\_ON = -fexceptions -mthreads

QMAKE\_CXXFLAGS\_EXCEPTIONS\_OFF = -fno-exceptions

QMAKE\_INCDIR = /usr/i686-w64-mingw32/include

QMAKE\_INCDIR\_QT = $$[QT\_INSTALL\_HEADERS]

QMAKE\_LIBDIR\_QT = $$[QT\_INSTALL\_LIBS]

QMAKE\_RUN\_CC = $(CC) -c $(CFLAGS) $(INCPATH) -o $obj $src

QMAKE\_RUN\_CC\_IMP = $(CC) -c $(CFLAGS) $(INCPATH) -o $@ $<

QMAKE\_RUN\_CXX = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $obj $src

QMAKE\_RUN\_CXX\_IMP = $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

QMAKE\_LINK = $${CROSS\_COMPILE}g++

QMAKE\_LINK\_C = $${CROSS\_COMPILE}gcc

QMAKE\_LFLAGS = -m32 -mthreads -static-libgcc -static -Wl,-enable-stdcall-fixup -Wl,-enable-auto-import -Wl,-enable-runtime-pseudo-reloc -mwindows

QMAKE\_LFLAGS\_EXCEPTIONS\_ON = -mthreads

QMAKE\_LFLAGS\_EXCEPTIONS\_OFF =

QMAKE\_LFLAGS\_RELEASE = -Wl,-s

QMAKE\_LFLAGS\_DEBUG =

QMAKE\_LFLAGS\_CONSOLE = -Wl,-subsystem,console

QMAKE\_LFLAGS\_WINDOWS = -Wl,-subsystem,windows

QMAKE\_LFLAGS\_DLL = -shared

QMAKE\_LINK\_OBJECT\_MAX = 10

QMAKE\_LINK\_OBJECT\_SCRIPT= object\_script

QMAKE\_PREFIX\_STATICLIB = lib

QMAKE\_EXTENSION\_STATICLIB = a

QMAKE\_LIBS =

QMAKE\_LIBS\_CORE = -lole32 -luuid -lws2\_32 -ladvapi32 -lshell32 -luser32 -lkernel32

QMAKE\_LIBS\_GUI = -lgdi32 -lcomdlg32 -loleaut32 -limm32 -lwinmm -lwinspool -lws2\_32 -lole32 -luuid -luser32 -ladvapi32

QMAKE\_LIBS\_NETWORK = -lws2\_32

QMAKE\_LIBS\_OPENGL = -lglu32 -lopengl32 -lgdi32 -luser32

QMAKE\_LIBS\_COMPAT = -ladvapi32 -lshell32 -lcomdlg32 -luser32 -lgdi32 -lws2\_32

QMAKE\_LIBS\_QT\_ENTRY = -lmingw32 -lqtmain

!isEmpty(QMAKE\_SH) {

MINGW\_IN\_SHELL = 1

QMAKE\_DIR\_SEP = /

QMAKE\_QMAKE ~= s,\\\\,/,

QMAKE\_COPY = cp

QMAKE\_COPY\_DIR = cp -r

QMAKE\_MOVE = mv

QMAKE\_DEL\_FILE = rm

QMAKE\_MKDIR = mkdir -p

QMAKE\_DEL\_DIR = rmdir

QMAKE\_CHK\_DIR\_EXISTS = test -d

} else {

QMAKE\_COPY = copy /y

QMAKE\_COPY\_DIR = xcopy /s /q /y /i

QMAKE\_MOVE = move

QMAKE\_DEL\_FILE = del

QMAKE\_MKDIR = mkdir

QMAKE\_DEL\_DIR = rmdir

QMAKE\_CHK\_DIR\_EXISTS = if not exist

}

QMAKE\_MOC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}moc$${EXE\_SUFFIX}

QMAKE\_UIC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}uic$${EXE\_SUFFIX}

QMAKE\_IDC = $$[QT\_INSTALL\_BINS]$${DIR\_SEPARATOR}idc$${EXE\_SUFFIX}

QMAKE\_IDL = midl

QMAKE\_LIB = $${CROSS\_COMPILE}ar -ru

QMAKE\_RC = $${CROSS\_COMPILE}windres

QMAKE\_ZIP = zip -r -9

QMAKE\_STRIP = $${CROSS\_COMPILE}strip

QMAKE\_STRIPFLAGS\_LIB += --strip-unneeded

load(qt\_config)

##### **Кросс-компиляция для Mac OS X**

Кросс-компиляция для мака тоже очень похожа, за исключением того, что надо будет собрать и компилятор. Я собирал по [этой инструкции](https://github.com/Tatsh/xchain). Это отняло полный день времени и кучу нервов. В процессе будет нужна рабочая Mac OS X (как минимум на виртуальной машине) с установленным XCode, чтобы взять оттуда необходимые файлы. При компилировании своих Qt-приложений запущенная Mac OS X не нужна.  
  
configure:

Скрытый текст

./configure -xplatform macx-x-g++ -prefix /usr/x86\_64-apple-darwin11/usr/Qt CROSS\_COMPILE=/usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11- -no-webkit -no-phonon -no-phonon-backend -no-script -no-scripttools -no-multimedia -no-qt3support -fast -nomake demos -nomake examples -nomake tools -device-option -little-endian -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked -no-fontconfig -no-3dnow -no-ssse3 -continue

qmake.conf:

Скрытый текст

*#macx-x-g++ (different from g++.conf)*

*#*

*# qmake configuration for macx-g++*

*#*

*# Mac OS X + command-line compiler*

*#*

MAKEFILE\_GENERATOR = UNIX

TARGET\_PLATFORM = macx

TEMPLATE = app

CONFIG += qt warn\_on release app\_bundle incremental global\_init\_link\_order lib\_version\_first plugin\_no\_soname link\_prl

QT += core gui

QMAKE\_INCREMENTAL\_STYLE = sublib

*#include(../common/mac.conf)*

*#*

*# qmake configuration for common Mac OS X*

*#*

*#QMAKE\_RESOURCE = /Developer/Tools/Rez*

QMAKE\_EXTENSION\_SHLIB = dylib

QMAKE\_INCDIR = /usr/x86\_64-apple-darwin11/usr/include

QMAKE\_LIBDIR =

QMAKE\_INCDIR\_QT = /usr/x86\_64-apple-darwin11/usr/Qt/include

QMAKE\_LIBDIR\_QT = /usr/x86\_64-apple-darwin11/usr/Qt/lib

QMAKE\_INCDIR\_OPENGL = /System/Library/Frameworks/OpenGL.framework/Headers \

/System/Library/Frameworks/AGL.framework/Headers/

QMAKE\_FRAMEWORKDIR\_QT= /usr/x86\_64-apple-darwin11/usr/QtFrameworks

QMAKE\_FIX\_RPATH = install\_name\_tool -id

QMAKE\_LFLAGS\_RPATH =

QMAKE\_LIBS\_DYNLOAD =

QMAKE\_LIBS\_OPENGL = -framework OpenGL -framework AGL

QMAKE\_LIBS\_OPENGL\_QT = $$QMAKE\_LIBS\_OPENGL

QMAKE\_LIBS\_THREAD =

QMAKE\_MOC = $$[QT\_INSTALL\_BINS]/moc

QMAKE\_UIC = $$[QT\_INSTALL\_BINS]/uic

QMAKE\_AR = /usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11-ar cq

QMAKE\_RANLIB = /usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11-ranlib -s

QMAKE\_TAR = tar -cf

QMAKE\_GZIP = gzip -9f

QMAKE\_COPY = cp -f

QMAKE\_COPY\_FILE = $$QMAKE\_COPY

QMAKE\_COPY\_DIR = $$QMAKE\_COPY -R

QMAKE\_MOVE = mv -f

QMAKE\_DEL\_FILE = rm -f

QMAKE\_DEL\_DIR = rmdir

QMAKE\_CHK\_DIR\_EXISTS = test -d

QMAKE\_MKDIR = mkdir -p

QMAKE\_MACOSX\_DEPLOYMENT\_TARGET = 10.5 *# overridden to 10.5 for Cocoa on the compiler command line*

*#*

*# qmake configuration for common unix*

*#*

QMAKE\_LEX = flex

QMAKE\_LEXFLAGS +=

QMAKE\_YACC = yacc

QMAKE\_YACCFLAGS += -d

QMAKE\_YACCFLAGS\_MANGLE += -p $base -b $base

QMAKE\_YACC\_HEADER = $base.tab.h

QMAKE\_YACC\_SOURCE = $base.tab.c

QMAKE\_PREFIX\_SHLIB = lib

QMAKE\_PREFIX\_STATICLIB = lib

QMAKE\_EXTENSION\_STATICLIB = a

*#include(../common/gcc-base-macx.conf)*

*#*

*# This file is used as a basis for the following compilers:*

*#*

*# - The QNX qcc compiler*

*#*

*# The only difference between this and gcc-base.conf is that -pipe is removed*

*# as it's on by default in qcc (and has an analogous -nopipe option)*

*#*

QMAKE\_CFLAGS +=

QMAKE\_CFLAGS\_DEPS += -M

QMAKE\_CFLAGS\_WARN\_ON += -Wall -W

QMAKE\_CFLAGS\_WARN\_OFF += -w

QMAKE\_CFLAGS\_RELEASE += -O2

QMAKE\_CFLAGS\_DEBUG += -g

QMAKE\_CFLAGS\_SHLIB += -fPIC

QMAKE\_CFLAGS\_STATIC\_LIB += -fPIC

QMAKE\_CFLAGS\_YACC += -Wno-unused -Wno-parentheses

QMAKE\_CFLAGS\_HIDESYMS += -fvisibility=hidden

QMAKE\_CXXFLAGS += $$QMAKE\_CFLAGS

QMAKE\_CXXFLAGS\_DEPS += $$QMAKE\_CFLAGS\_DEPS

QMAKE\_CXXFLAGS\_WARN\_ON += $$QMAKE\_CFLAGS\_WARN\_ON

QMAKE\_CXXFLAGS\_WARN\_OFF += $$QMAKE\_CFLAGS\_WARN\_OFF

QMAKE\_CXXFLAGS\_RELEASE += $$QMAKE\_CFLAGS\_RELEASE

QMAKE\_CXXFLAGS\_DEBUG += $$QMAKE\_CFLAGS\_DEBUG

QMAKE\_CXXFLAGS\_SHLIB += $$QMAKE\_CFLAGS\_SHLIB

QMAKE\_CXXFLAGS\_STATIC\_LIB += $$QMAKE\_CFLAGS\_STATIC\_LIB

QMAKE\_CXXFLAGS\_YACC += $$QMAKE\_CFLAGS\_YACC

QMAKE\_CXXFLAGS\_HIDESYMS += $$QMAKE\_CFLAGS\_HIDESYMS -fvisibility-inlines-hidden

QMAKE\_LFLAGS +=

QMAKE\_LFLAGS\_DEBUG +=

QMAKE\_LFLAGS\_APP +=

QMAKE\_LFLAGS\_RELEASE +=

*#*

*# Base qmake configuration for GCC on Mac OS X*

*#*

*# Before making changes to this file, please read the comment in*

*# gcc-base.conf, to make sure the change goes in the right place.*

*#*

*# To verify that your change has the desired effect on the final configuration*

*# you can use the manual test in tests/manual/mkspecs.*

*#*

QMAKE\_COMPILER\_DEFINES += \_\_APPLE\_\_ \_\_GNUC\_\_

QMAKE\_CFLAGS\_X86 += -arch i386

QMAKE\_CFLAGS\_X86\_64 += -arch x86\_64

QMAKE\_CFLAGS\_PPC += -arch ppc

QMAKE\_CFLAGS\_PPC\_64 += -arch ppc64

QMAKE\_CFLAGS\_DWARF2 += -gdwarf-2

QMAKE\_CXXFLAGS\_X86 += $$QMAKE\_CFLAGS\_X86

QMAKE\_CXXFLAGS\_X86\_64 += $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_CXXFLAGS\_PPC += $$QMAKE\_CFLAGS\_PPC

QMAKE\_CXXFLAGS\_PPC\_64 += $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_CXXFLAGS\_DWARF2 += $$QMAKE\_CFLAGS\_DWARF2

QMAKE\_OBJECTIVE\_CFLAGS = $$QMAKE\_CFLAGS

QMAKE\_OBJECTIVE\_CFLAGS\_WARN\_ON = $$QMAKE\_CFLAGS\_WARN\_ON

QMAKE\_OBJECTIVE\_CFLAGS\_WARN\_OFF = $$QMAKE\_CFLAGS\_WARN\_OFF

QMAKE\_OBJECTIVE\_CFLAGS\_DEBUG = $$QMAKE\_CFLAGS\_DEBUG

QMAKE\_OBJECTIVE\_CFLAGS\_RELEASE = $$QMAKE\_CFLAGS\_RELEASE

QMAKE\_OBJECTIVE\_CFLAGS\_HIDESYMS = $$QMAKE\_CXXFLAGS\_HIDESYMS

QMAKE\_OBJECTIVE\_CFLAGS\_X86 = $$QMAKE\_CFLAGS\_X86

QMAKE\_OBJECTIVE\_CFLAGS\_X86\_64 = $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_OBJECTIVE\_CFLAGS\_PPC = $$QMAKE\_CFLAGS\_PPC

QMAKE\_OBJECTIVE\_CFLAGS\_PPC\_64 = $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_LFLAGS\_X86 += $$QMAKE\_CFLAGS\_X86

QMAKE\_LFLAGS\_X86\_64 += $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_LFLAGS\_PPC += $$QMAKE\_CFLAGS\_PPC

QMAKE\_LFLAGS\_PPC\_64 += $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_LFLAGS += -headerpad\_max\_install\_names

QMAKE\_LFLAGS\_SHLIB += -single\_module -dynamiclib

QMAKE\_LFLAGS\_PLUGIN += $$QMAKE\_LFLAGS\_SHLIB

QMAKE\_LFLAGS\_INCREMENTAL += -undefined suppress -flat\_namespace

QMAKE\_LFLAGS\_SONAME += -install\_name$${LITERAL\_WHITESPACE}

QMAKE\_LFLAGS\_VERSION += -current\_version$${LITERAL\_WHITESPACE}

QMAKE\_LFLAGS\_COMPAT\_VERSION += -compatibility\_version$${LITERAL\_WHITESPACE}

*#*

*# Qmake configuration for the GNU C++ compiler*

*#*

*# Before making changes to this file, please read the comment in*

*# gcc-base.conf, to make sure the change goes in the right place.*

*#*

*# To verify that your change has the desired effect on the final configuration*

*# you can use the manual test in tests/manual/mkspecs.*

*#*

QMAKE\_CC = /usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11-g++

QMAKE\_LINK\_C = $$QMAKE\_CC -F/usr/x86\_64-apple-darwin11/usr/Frameworks

QMAKE\_LINK\_C\_SHLIB = $$QMAKE\_CC -F/usr/x86\_64-apple-darwin11/usr/Frameworks

QMAKE\_CFLAGS\_RELEASE\_WITH\_DEBUGINFO += -O2 -g

QMAKE\_CXX = /usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11-g++

QMAKE\_LINK = /usr/x86\_64-apple-darwin11/usr/bin/x86\_64-apple-darwin11-g++ -F/usr/x86\_64-apple-darwin11/usr/Frameworks

QMAKE\_LINK\_SHLIB = $$QMAKE\_CXX -F/usr/x86\_64-apple-darwin11/usr/Frameworks

QMAKE\_CXXFLAGS\_RELEASE\_WITH\_DEBUGINFO += $$QMAKE\_CFLAGS\_RELEASE\_WITH\_DEBUGINFO

QMAKE\_PCH\_OUTPUT\_EXT = .gch

QMAKE\_CFLAGS\_PRECOMPILE = -x c-header -c ${QMAKE\_PCH\_INPUT} -o ${QMAKE\_PCH\_OUTPUT}

QMAKE\_CFLAGS\_USE\_PRECOMPILE = -include ${QMAKE\_PCH\_OUTPUT\_BASE}

QMAKE\_CXXFLAGS\_PRECOMPILE = -x c++-header -c ${QMAKE\_PCH\_INPUT} -o ${QMAKE\_PCH\_OUTPUT}

QMAKE\_CXXFLAGS\_USE\_PRECOMPILE = $$QMAKE\_CFLAGS\_USE\_PRECOMPILE

*#include(../common/g++-macx.conf)*

*#*

*# Qmake configuration for the GNU C++ compiler on Mac OS X*

*#*

*# Before making changes to this file, please read the comment in*

*# gcc-base.conf, to make sure the change goes in the right place.*

*#*

*# To verify that your change has the desired effect on the final configuration*

*# you can use the manual test in tests/manual/mkspecs.*

*#*

QMAKE\_CFLAGS\_RELEASE\_WITH\_DEBUGINFO += $$QMAKE\_CFLAGS\_DWARF2

QMAKE\_CXXFLAGS\_RELEASE\_WITH\_DEBUGINFO += $$QMAKE\_CFLAGS\_DWARF2

QMAKE\_LFLAGS\_RELEASE\_WITH\_DEBUGINFO += -g $$QMAKE\_CFLAGS\_DWARF2

QMAKE\_LFLAGS\_STATIC\_LIB += -all\_load

QMAKE\_CFLAGS\_X86\_64 += -Xarch\_x86\_64 -mmacosx-version-min=10.5

QMAKE\_CFLAGS\_PPC\_64 += -Xarch\_ppc64 -mmacosx-version-min=10.5

QMAKE\_CXXFLAGS\_X86\_64 = $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_CXXFLAGS\_PPC\_64 = $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_OBJECTIVE\_CFLAGS\_X86\_64 = $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_OBJECTIVE\_CFLAGS\_PPC\_64 = $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_LFLAGS\_X86\_64 = $$QMAKE\_CFLAGS\_X86\_64

QMAKE\_LFLAGS\_PPC\_64 = $$QMAKE\_CFLAGS\_PPC\_64

QMAKE\_OBJCFLAGS\_PRECOMPILE = -x objective-c-header -c ${QMAKE\_PCH\_INPUT} -o ${QMAKE\_PCH\_OUTPUT}

QMAKE\_OBJCFLAGS\_USE\_PRECOMPILE = $$QMAKE\_CFLAGS\_USE\_PRECOMPILE

QMAKE\_OBJCXXFLAGS\_PRECOMPILE = -x objective-c++-header -c ${QMAKE\_PCH\_INPUT} -o ${QMAKE\_PCH\_OUTPUT}

QMAKE\_OBJCXXFLAGS\_USE\_PRECOMPILE = $$QMAKE\_CFLAGS\_USE\_PRECOMPILE

load(qt\_config)

Помните, в Mac OS X для линковки с библиотекой **.a**-файлы не нужны.

##### **Настройка QtCreator**

Сначала нужно добавить в список все установленные компиляторы. Инструменты — Параметры — Сборка и запуск — Инструментарии:  
  
QtCreator обычно нормально определяет ABI, но лучше перепроверить. Так же можно заметить, что системный x64 GCC в linux умеет генерировать и 32-битные приложения. Однако это не отменяет того, что также необходимы 32-битные версии библиотек.  
  
После компиляторов можно добавить профили Qt:  
  
Вот при добавлении профиля и пригодятся собранные ранее *qmake, uic, moc, rcc*, ведь нужно выбрать директорию с *qmake*. Жёлтый значок с восклицательным знаком слева от профиля означает warning, но QtCreator может использовать такой профиль Qt. А вот если значок красный, то профиль нерабочий. Такое может случиться при неправильной структуре каталогов. Или если удалить директорию, в которой компилировали Qt.  
  
Следующие настройки нужно делать в каждом создаваемом проекте.  
Для добавления конкретного профиля Qt надо при активном проекте зайти на вкладку «Проекты» (Ctrl+5):  
  
  
По умолчанию в списке «Изменить конфигурацию сборки» есть только системный профиль Qt. Зато в списке кнопки «Добавить» есть все профили Qt, добавленные в параметры сборки.  
  
В основных настройках сборки необходимо проверить пару библиотека-компилятор. Чтоб и то и другое было от одной и той же операционной системы.  
  
Этапы сборки «qmake» и «Сборка» QtCreator ставит по умолчанию. А вот особые этапы «upx» и «dmgbuild» я добавил вручную для своего проекта. Этап «upx» выполняется каждый раз при нажатии на кнопку «Собрать проект». Однако если исполняемый файл не был изменён, то *upx* вернёт ошибку, что файл им уже обработан. В случае ошибки следующий этап не вызывается, т.е. dmg-файл обновится только если upx отработал успешно.  
  
Для работы этапа *upx* он должен быть установлен в системе. Однако даже работая в linux-окружении и поставленный из пакетного менеджера *upx* умеет ужимать приложения: linux32/64, win32, macos32/64. Далеко не для всех проектов upx-сжатие реально нужно, этап показан скорее для примера.  
  
Для этапа «dmgbuild» я воспользовался скриптом [make\_dmg](http://confluence.concord.org/display/CCTR/Creating+MacOS+dmg+files+in+Linux). Ему нужны права *root*, поэтому добавил скрипт в файл */etc/sudoers*

##### **Изменения в проектном файле и использование сторонних библиотек**

В моём проекте используется *libusb*, а это далеко не часть Qt. Также необходимо было включить платформенно-зависимую реализацию HID. В проектный файл были добавлены строки:

macx {

INCLUDEPATH += $$PWD/libusbx/

SOURCES += BootLoader/HIDAPI/mac/hid.c

LIBS += -framework IOKit -framework CoreFoundation -lusb-1.0

ICON = AqPicFlash.icns

}

win32: {

INCLUDEPATH += $$PWD/libusbx/

LIBS += -lsetupapi -lole32

SOURCES += BootLoader/HIDAPI/windows/hid.cpp

RC\_FILE = WinIcon.rc

}

win32: !win64-x-g++ {

LIBS += -L$$PWD/libusbx/ -lusb-1.0-32.dll

}

win64-x-g++ {

LIBS += -L$$PWD/libusbx/ -lusb-1.0-64.dll

}

unix: !macx {

CONFIG += link\_pkgconfig

PKGCONFIG += libusb-1.0

SOURCES += BootLoader/HIDAPI/linux/hid-libusb.c

}

В Mac OS X и Linux линкуемся с системной libusb, в Windows в зависимости от разрядности линкуемся с *libusb-1.0-32.dll.a* или *libusb-1.0-64.dll.a*. Помним, что **.a**-файл может быть переименован, но зависеть приложение всё-равно будет от *libusb-1.0.dll*. В Linux параметры для libusb берём через системную утилиту pkgconfig. Кроме libusb подключаем для каждой операционной системы необходимые системные библиотеки и иконки.  
  
Удобно разнести итоговые файлы для разных операционных систем по директориям. Сделать это можно так:

macx {

DESTDIR = mac

OBJECTS\_DIR = mac

MOC\_DIR = mac

UI\_DIR = mac

RCC\_DIR = mac

}

unix: !macx {

DESTDIR = linux

OBJECTS\_DIR = linux

MOC\_DIR = linux

UI\_DIR = linux

RCC\_DIR = linux

}

win32 {

DESTDIR = windows/release

OBJECTS\_DIR = windows

MOC\_DIR = windows

UI\_DIR = windows

RCC\_DIR = windows

}

win64-x-g++ {

DESTDIR = win64/release

OBJECTS\_DIR = win64

MOC\_DIR = win64

UI\_DIR = win64

RCC\_DIR = win64

}

Цель *win64-x-g++* относится к *win32*, однако в проектном файле идёт последней и переписывает настройки.

##### **Результат**

Теперь чтобы собрать приложение под ту или иную операционную систему достаточно выбрать тип сборки, как показано на самом первом скриншоте, и нажать «Собрать проект»

**Теги:**

* [qt](https://habr.com/ru/search/?q=%5Bqt%5D&target_type=posts)
* [qtcreator](https://habr.com/ru/search/?q=%5Bqtcreator%5D&target_type=posts)
* [cross compile](https://habr.com/ru/search/?q=%5Bcross%20compile%5D&target_type=posts)
* [crossplatform](https://habr.com/ru/search/?q=%5Bcrossplatform%5D&target_type=posts)
* [linux](https://habr.com/ru/search/?q=%5Blinux%5D&target_type=posts)
* [windows](https://habr.com/ru/search/?q=%5Bwindows%5D&target_type=posts)
* [mac os x](https://habr.com/ru/search/?q=%5Bmac%20os%20x%5D&target_type=posts)
* [upx](https://habr.com/ru/search/?q=%5Bupx%5D&target_type=posts)
* [dmg](https://habr.com/ru/search/?q=%5Bdmg%5D&target_type=posts)

**Хабы:**

* [Программирование](https://habr.com/ru/hub/programming/)
* [Qt](https://habr.com/ru/hub/qt_software/)

# Кросс-компиляция Qt5 под Linux для Win x32/x64/static/shared

* [PostgreSQL](https://habr.com/ru/hub/postgresql/),
* [C++](https://habr.com/ru/hub/cpp/),
* [Qt](https://habr.com/ru/hub/qt_software/)
* [Из песочницы](https://habr.com/ru/sandbox/)

# Цель статьи

Документирование получения системы кросс-компиляции под Linux для Windows x32/x64/static/shared и сборка последней на момент описания Qt 5.4.1 в лайт-версии (для указанных четырех целей). Для себя, глубоко-обожаемого, ну и для пользы обществу.

# Назначение

Многие разработчики приходят к выводу, что использование \*nix (в частности Linux) более предпочтительно для разработки приложений, используя фрэймворк Qt. И тому есть причины. Qt изначально ориентирована на \*nix инструментарий, типа autotool, make, perl… И второй момент, под никсами есть прекрасный инструмент — [valgrind](http://ru.wikipedia.org/wiki/Valgrind), под виндой порта пока его не видел. Ну и последняя причина: просто удобно иметь набор инструментария для создания приложений под различные целевые платформы — в одном месте.

# Почему лайт-версия Qt5

Фрэймворк-Qt имеет модульную структуру, но, увы, не совсем совершенную. Некоторые зависимости от внешних библиотек «вешаются» не на модуль, требующий эти библиотеки, а на Qt5Core. Иными словами, нужна вам эта библиотека, или нет — вынь да положЪ в дистрибутив. Пример тому «монстрик» — библиотека [ICU](http://site.icu-project.org/). Весит она почти 25 метров! Зависимость вешается, как я упоминал выше, на Qt5Core… а требует ее модуль Qt5WebKit (который по сути в 99% случаев не используется, по крайней мере мной). Что делаем? Вырезаем и отрезаем. Хотите получить фулл-версию Qt5? Об этом упомяну в заключении. Поехали.

## **0. Сценарий сборки**

Все шаги делаем последовательно. Желательно не объединять все скрипты в последовательную сборку по причине необходимости промежуточного «человечного» контроля. Разные дистрибутивы Линуха, разные среды исполнения, наборы инструментариев… Простой алгоритм: сделал очередной шаг, убедился в отсутствии ошибок, пошел делать следующий. Итак, сам сценарий:

* 1. Предварительная подготовка
* 2. Установка среды кросс-компиляции [MXE](http://mxe.cc/)
* 3. Загрузка и настройка Qt 5.4.1
* 4. Сборка комплектов Qt 5.4.1 для четырех целей (см. сабж)
* 5. Прописка собранного в QtCreator

## **1. Предварительная подготовка**

У вас установлен дистрибутив Линукса. Желательно все это делать на на продакшен-компе (не на живом Линуксе), а установленном в виртуальную машину. Я например, пользуюсь VMWare, но это дело вкуса. Выбор дистрибутива Линукса — так же дело вкуса. Лично я предпочитаю [Gentoo Linux](http://www.gentoo.org/), собственно под ним всю эту кухню и настраиваю. Если есть сложности в настройке, у меня есть небольшая статейка по этому вопросу: [«Установка и настройка Linux Gentoo под VMWare»](http://majestio.tk/viewtopic.php?id=29).  
  
Итак, у вас есть настроенный Линукс и вы работаете **не под рутом!** Для дальнейшей работы вам нужно проверить присутствие следующих установленных пакетов, или доустановить:

$ sudo emerge \

app-arch/bzip2 \

app-arch/unzip \

app-arch/xz-utils \

app-shells/bash \

dev-lang/ruby \

dev-libs/libffi \

dev-libs/openssl \

dev-perl/XML-Parser \

dev-util/cmake \

dev-util/intltool \

dev-util/pkgconfig \

dev-util/scons \

dev-vcs/git \

sys-devel/autoconf \

sys-devel/automake \

sys-devel/bison \

sys-devel/flex \

sys-devel/gettext \

sys-devel/libtool \

sys-devel/patch \

net-misc/ntp \

www-client/lynx

Вся дальнейшая установка будет производиться в каталог **$HOME/dev**. Если у вас таковой присутствует — либо переименовываете его, либо внимательно правите скрипты, которые будут приведены далее. Для всех манипуляций со скачиваемыми внешними файлами/скриптами/библиотеками будет использован каталог **$HOME/setup**. Все замечания выше относительно $HOME/dev — в силе и к этому каталогу.

## **2. Установка среды кросс-компиляции MXE**

Предварительное замечание об MXE. Это отличнейшая система сборки тулчейнов для кросс-компиляции. Но есть одно «но». В данный момент не существует стабильной ветки. Авторы до поры до времени вели две ветки в своем git-репозитарии — стабильную и «разработческую». Сейчас ветки объединены. Разработка идет ну очень активно — изменения сбрасываются чуть ли не раз 1-3 дня. А это чревато тем, что «то работает сборка, то не работает». Некоторые важные для меня библиотеки, в частности клиентская часть PostgreSQL, собираются без ошибок, но в нерабочем состоянии. Потратил неделю не исследование явных косяков. Исправляем эти «недочеты». Итак:

$ mkdir -p $HOME/setup

$ cd $HOME/setup

$ wget http://majestio.tk/stuff/setup-scripts.tar.xz

$ tar -xf setup-scripts.tar.xz

$ rm -f setup-scripts.tar.xz

Должны получить в каталоге $HOME/setup следующий набор скриптов:

1.setup-first.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 1. Создание структуры каталогов для разворачивания системы ##*

*## кросс-компиляции средствами MXE (http://mxe.cc), ##*

*## собственно загрузку и первоначальную сборку ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Создать в $HOME подкаталог, например $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# Структура подкаталогов будет в $HOME/dev*

*# Набор тулчейнов MXE ищем в $HOME/dev/cross/mxe*

*#*

*# .........................................................................*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

RETDIR=`pwd`

MXEHOME=$HOME/dev/cross/mxe

*# создаем структуру каталогов и вынимаем MXE из git-репозитария -----------*

mkdir -p $HOME/dev/{cross,projests,src,stuff,tools}

cd $HOME/dev/cross

*# можно так:*

*# git clone https://github.com/mxe/mxe.git*

*# но лучше так:*

*# wget http://majestio.tk/stuff/mxe-empty-26.05.2015.tar.xz*

*# tar -xf mxe-empty-26.05.2015.tar.xz*

*# rm -f mxe-empty-26.05.2015.tar.xz*

wget http://majestio.tk/stuff/mxe-empty-26.05.2015.tar.xz

tar -xf mxe-empty-26.05.2015.tar.xz

rm -f mxe-empty-26.05.2015.tar.xz

*# патчим файлик определения ABI проекта MXE -------------------------------*

echo "#!/bin/sh" > $HOME/dev/cross/mxe/ext/config.guess

echo "echo \"x86\_64-pc-linux-gnu\"" >> $HOME/dev/cross/mxe/ext/config.guess

*# делаем первоначальную сборку MXE с типом нитей win32 --------------------*

cd $MXEHOME

make MXE\_TARGETS='x86\_64-w64-mingw32.shared '`

`'x86\_64-w64-mingw32.static '`

`'i686-w64-mingw32.shared '`

`'i686-w64-mingw32.static ' gcc gmp winpthreads -j4 JOBS=4

*# патчим mxe/src/gcc.mk на предмет добавления posix-threads ---------------*

GCCMK=$MXEHOME/src/gcc.mk

PLTMP=$$-$RANDOM.pl

echo "open(F,\"$GCCMK\") || die \"Holy shit!\\n\";" > $PLTMP

echo "@In = <F>;" >> $PLTMP

echo "close(F);" >> $PLTMP

echo "open(F,\">$GCCMK\") || die \"Holy shit!\\n\";" >> $PLTMP

echo "foreach \$I(@In) {" >> $PLTMP

echo " \$I =~ s/^(\\\$\(PKG\)\_DEPS.+)\$/\$1 winpthreads/;" >> $PLTMP

echo " \$I =~ s/^(.+?--libdir='\\\$\(PREFIX\).+)\$/\$1\n --enable-shared=libstdc\+\+ \\\/;" >> $PLTMP

echo " \$I =~ s/^(.+?--enable-threads=)win32(.\*)\$/\$1posix\$2/;" >> $PLTMP

echo " print F \$I;" >> $PLTMP

echo "}" >> $PLTMP

echo "close(F);" >> $PLTMP

perl $PLTMP && rm -f $PLTMP

*# пересобираем GCC с поддержкой posix-threads -----------------------------*

cd $MXEHOME

make MXE\_TARGETS='x86\_64-w64-mingw32.shared '`

`'x86\_64-w64-mingw32.static '`

`'i686-w64-mingw32.shared '`

`'i686-w64-mingw32.static ' gcc -j4 JOBS=4

*# делаем промежуточный архив "пустой" системы кросс-компиляции ------------*

echo "Делаем промежуточную архивацию ..."

cd ..

tar -cJf mxe-winpthreads-empty.tar.xz ./mxe

*# собираем дополнительные либы --------------------------------------------*

cd $MXEHOME

make MXE\_TARGETS='x86\_64-w64-mingw32.shared '`

`'x86\_64-w64-mingw32.static '`

`'i686-w64-mingw32.shared '`

`'i686-w64-mingw32.static ' zlib xz sqlite dbus \

libiconv libodbc++ -j4 JOBS=4

*# патчим и собираем PostgreSQL --------------------------------------------*

cd $MXEHOME/src

wget http://majestio.tk/stuff/postgresql-3-socket.patch

PQSQL=$MXEHOME/src/postgresql.mk

PLTMP=$$-$RANDOM.pl

echo "open(F,\"$PQSQL\") || die \"Holy shit!\\n\";" > $PLTMP

echo "@In = <F>;" >> $PLTMP

echo "close(F);" >> $PLTMP

echo "open(F,\">$PQSQL\") || die \"Holy shit!\\n\";" >> $PLTMP

echo "foreach \$I(@In) {" >> $PLTMP

echo " \$I =~ s/^(.+?--disable-rpath.+)\$/\$1\n --disable-thread-safety \\\/;" >> $PLTMP

echo " print F \$I;" >> $PLTMP

echo "}" >> $PLTMP

echo "close(F);" >> $PLTMP

perl $PLTMP && rm -f $PLTMP

cd $MXEHOME

make MXE\_TARGETS='x86\_64-w64-mingw32.shared '`

`'x86\_64-w64-mingw32.static '`

`'i686-w64-mingw32.shared '`

`'i686-w64-mingw32.static ' postgresql -j4 JOBS=4

for PREFIX in "x86\_64-w64-mingw32.shared" "x86\_64-w64-mingw32.static" \

"i686-w64-mingw32.shared" "i686-w64-mingw32.static" ; do

echo "Description: PostgreSQL libpq library" > $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Url: http://www.postgresql.org/" >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Version: 9.2.4" >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Requires: " >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Requires.private: " >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Cflags: -I$MXEHOME/usr/$PREFIX/include" >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Libs: -L$MXEHOME/usr/$PREFIX/lib -lpq" >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

echo "Libs.private: -lssl -lcrypto -lshell32 -lwsock32 -lws2\_32 -lsecur32" >> $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

chmod 0644 $MXEHOME/usr/$PREFIX/lib/pkgconfig/libpq.pc

done

cd $RETDIR

2.setup-qt5.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 2. Выкачка Qt 5.4.1 + небольшие косметические правки ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Зайти в каталог установки $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# На выходе будет каталог qt5*

*#*

*# -------------------------------------------------------------------------*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

wget http://download.qt.io/official\_releases/qt/5.4/5.4.1/single/qt-everywhere-opensource-src-5.4.1.tar.gz

tar -xf qt-everywhere-opensource-src-5.4.1.tar.gz

mv qt-everywhere-opensource-src-5.4.1 qt5

cd qt5/qtbase/mkspecs

wget http://majestio.tk/stuff/win32-g++.static.tar.xz

tar -xf win32-g++.static.tar.xz

rm -f win32-g++.static.tar.xz

cd ../..

wget http://majestio.tk/stuff/qt-5.4.1.patch

patch -p1 -u < qt-5.4.1.patch

rm -f qt-5.4.1.patch

cd ..

3.1.build-qt-x32-shared.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 3.1 Создание сборки Qt 5.4.1 для Win x32 (shared) ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Зайти в каталог установки $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# На выходе будет готовая сборка, размещенная в %HOME/dev/stuff*

*#*

*# -------------------------------------------------------------------------*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

TARGET="i686-w64-mingw32.shared"

MXE="$HOME/dev/cross/mxe/usr"

PREFIX="$HOME/dev/stuff/qt5-$TARGET"

export PATH=$MXE/bin:$PATH

PWD="`pwd`"

LOG="$PWD/$TARGET-qt5.log\_\_\_\_\_\_\_\_\_\_\_"

mkdir "$TARGET-qt5.build"

cd "$TARGET-qt5.build"

clear

OPENSSL\_LIBS="`''"$TARGET"'-pkg-config' --libs-only-l openssl`" \

PSQL\_LIBS="-lpq -lsecur32 `''"$TARGET"'-pkg-config' --libs-only-l openssl` -lws2\_32" \

../qt5/configure -prefix $PREFIX \

-opensource \

-confirm-license \

-release \

-shared \

-opengl desktop \

-force-pkg-config \

-no-use-gold-linker \

-no-glib \

-openssl \

-dbus \

-plugin-sql-psql \

-plugin-sql-sqlite \

-plugin-sql-odbc \

-qt-libpng -qt-libjpeg -qt-pcre -zlib -qt-freetype -no-kms -no-pch \

-nomake examples -nomake tests -silent -skip webkit \

-xplatform win32-g++ \

-device-option CROSS\_COMPILE=$TARGET- \

-device-option PKG\_CONFIG="$TARGET-pkg-config" 2>&1 | tee $LOG

make -j4 2>&1 | tee -a $LOG

make install 2>&1 | tee -a $LOG

3.2.build-qt-x32-static.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 3.2 Создание сборки Qt 5.4.1 для Win x32 (static) ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Зайти в каталог установки $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# На выходе будет готовая сборка, размещенная в %HOME/dev/stuff*

*#*

*# -------------------------------------------------------------------------*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

TARGET="i686-w64-mingw32.static"

MXE="$HOME/dev/cross/mxe/usr"

PREFIX="$HOME/dev/stuff/qt5-$TARGET"

PWD="`pwd`"

LOG="$PWD/$TARGET-qt5.log\_\_\_\_\_\_\_\_\_\_\_"

mkdir "$TARGET-qt5.build"

cd "$TARGET-qt5.build"

clear

export PATH=$MXE/bin:$PATH

*#../qt5/configure --help|less*

*#exit*

OPENSSL\_LIBS="`''"$TARGET"'-pkg-config' --libs-only-l openssl`" \

PSQL\_LIBS="-lpq -lsecur32 `''"$TARGET"'-pkg-config' --libs-only-l openssl` -lws2\_32" \

../qt5/configure -prefix $PREFIX \

-opensource \

-confirm-license \

-release \

-static \

-opengl desktop \

-force-pkg-config \

-no-use-gold-linker \

-no-glib \

-openssl-linked \

-dbus-linked \

-qt-sql-psql \

-qt-sql-sqlite \

-qt-sql-odbc \

-qt-libpng -qt-libjpeg -qt-pcre -zlib -qt-freetype -no-kms -no-pch \

-nomake examples -nomake tests -silent -skip webkit \

-xplatform win32-g++ \

-device-option CROSS\_COMPILE=$TARGET- \

-device-option PKG\_CONFIG="$TARGET-pkg-config" 2>&1 | tee $LOG

make -j4 2>&1 | tee -a $LOG

make install 2>&1 | tee -a $LOG

3.3.build-qt-x64-shared.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 3.3 Создание сборки Qt 5.4.1 для Win x64 (shared) ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Зайти в каталог установки $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# На выходе будет готовая сборка, размещенная в %HOME/dev/stuff*

*#*

*# -------------------------------------------------------------------------*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

TARGET="x86\_64-w64-mingw32.shared"

MXE="$HOME/dev/cross/mxe/usr"

PREFIX="$HOME/dev/stuff/qt5-$TARGET"

export PATH=$MXE/bin:$PATH

PWD="`pwd`"

LOG="$PWD/$TARGET-qt5.log\_\_\_\_\_\_\_\_\_\_\_"

mkdir "$TARGET-qt5.build"

cd "$TARGET-qt5.build"

clear

OPENSSL\_LIBS="`''"$TARGET"'-pkg-config' --libs-only-l openssl`" \

PSQL\_LIBS="-lpq -lsecur32 `''"$TARGET"'-pkg-config' --libs-only-l openssl` -lws2\_32" \

../qt5/configure -prefix $PREFIX \

-opensource \

-confirm-license \

-release \

-shared \

-opengl desktop \

-force-pkg-config \

-no-use-gold-linker \

-no-glib \

-openssl \

-dbus \

-plugin-sql-psql \

-plugin-sql-sqlite \

-plugin-sql-odbc \

-qt-libpng -qt-libjpeg -qt-pcre -zlib -qt-freetype -no-kms -no-pch \

-nomake examples -nomake tests -silent -skip webkit \

-xplatform win32-g++ \

-device-option CROSS\_COMPILE=$TARGET- \

-device-option PKG\_CONFIG="$TARGET-pkg-config" 2>&1 | tee $LOG

make -j4 2>&1 | tee -a $LOG

make install 2>&1 | tee -a $LOG

3.4.build-qt-x64-static.sh

#!/bin/sh

*###########################################################################*

*## ##*

*## Скрипт 3.4 Создание сборки Qt 5.4.1 для Win x64 (static) ##*

*## ##*

*## Majestio (C) http://majestio.tk ##*

*###########################################################################*

*#*

*# Порядок установки:*

*#*

*# 1. Зайти в каталог установки $HOME/Setup*

*# 2. Скопировать туда этот скрипт и запустить*

*# 3. Дождаться завершения*

*#*

*# На выходе будет готовая сборка, размещенная в %HOME/dev/stuff*

*#*

*# -------------------------------------------------------------------------*

if [ "$(whoami)" == 'root' ]; then

echo "Ахтунг: под учетной записью рута работать отказываюсь!"

exit 1

fi

TARGET="x86\_64-w64-mingw32.static"

MXE="$HOME/dev/cross/mxe/usr"

PREFIX="$HOME/dev/stuff/qt5-$TARGET"

PWD="`pwd`"

LOG="$PWD/$TARGET-qt5.log\_\_\_\_\_\_\_\_\_\_\_"

mkdir "$TARGET-qt5.build"

cd "$TARGET-qt5.build"

clear

export PATH=$MXE/bin:$PATH

*#../qt5/configure --help|less*

*#exit*

OPENSSL\_LIBS="`''"$TARGET"'-pkg-config' --libs-only-l openssl`" \

PSQL\_LIBS="-lpq -lsecur32 `''"$TARGET"'-pkg-config' --libs-only-l openssl` -lws2\_32" \

../qt5/configure -prefix $PREFIX \

-opensource \

-confirm-license \

-release \

-static \

-opengl desktop \

-force-pkg-config \

-no-use-gold-linker \

-no-glib \

-openssl-linked \

-dbus-linked \

-qt-sql-psql \

-qt-sql-sqlite \

-qt-sql-odbc \

-qt-libpng -qt-libjpeg -qt-pcre -zlib -qt-freetype -no-kms -no-pch \

-nomake examples -nomake tests -silent -skip webkit \

-xplatform win32-g++ \

-device-option CROSS\_COMPILE=$TARGET- \

-device-option PKG\_CONFIG="$TARGET-pkg-config" 2>&1 | tee $LOG

make -j4 2>&1 | tee -a $LOG

make install 2>&1 | tee -a $LOG

Скрипты не будут иметь атрибута «исполняемый» просто в целях безопасности — предотвращения «непреднамеренного случайного» запуска. Запускаем первый скрипт **1.setup-first.sh**. Он создаст структуру подкаталогов $HOME/dev…, выкачает систему кросс-компиляции mxe, делает двойную сборку gcc тулчейнов (для обеспечения поддержки posix-threads), соберет основные базовые библиотеки, пропатчит и соберет клиентскую часть библиотеки PostgreSQL.  
  
**Важно:** Если ваша хостовая система по ABI отличается от «x86\_64-pc-linux-gnu», в скрипте вручную удалите патч ABI. Ищем по комментарию "# патчим файлик определения ABI проекта MXE". А лучше пропишете ваш.

$ sh 1.setup-first.sh

## **3. Загрузка и настройка Qt 5.4.1**

Запускаем скрипт **2.setup-qt5.sh**. Его задача выкачать исходники Qt 5.4.1, развернуть их в подкаталог qt5/ и пропатчить нужное.

$ sh 2.setup-qt5.sh

## **4. Сборка комплектов Qt 5.4.1 для четырех целей**

Для, собственно, сборки четырех видов Qt 5.4.1 предназначены остальные скрипты:

* 3.1.build-qt-x32-shared.sh
* 3.2.build-qt-x32-static.sh
* 3.3.build-qt-x64-shared.sh
* 3.4.build-qt-x64-static.sh

Запуск производиться точно так же как и в предыдущем пункте:

$ sh скрипт

Однако есть важное замечание, могущее помочь вам сэкономить время на сборку. Если у вас многоядерный компьютер — имеет смысл запускать сборки параллельно. Для этого вы открываете второй(третий, четвертый) терминал, и в терминале запускаете очередной скрипт. Лично мне удобно все действия проводить в дополнительном ssh-клиенте. Как показала практика, на моем Core i7 (4 ядра, с гипертрейдингом) — прирост общего времени сборки наблюдается при двух параллельно работающих сборках. Если более — производительность не возрастает, а чаще всего падает.  
  
Итак, все четыре скрипта отработали. Что мы получили в результате?  
  
$HOME/dev/cross/mxe/usr/bin — тут собраны исполняемые модули всех четырех тулчейнов  
$HOME/dev/cross/mxe/usr/тулчейн — тут собраны заголовки и либы тулчейнов  
$HOME/dev/stuff/сборки — тут собраны все виды Qt для последующей кросс-компиляции  
  
Осталось все это подключить к использованию.

## **5. Прописка собранного в QtCreator**

Самостоятельно выкачиваете Qt 5.4.1 для вашего линуха (ссылка на [x32](http://download.qt.io/official_releases/qt/5.4/5.4.1/qt-opensource-linux-x86-5.4.1.run) или [x64](http://download.qt.io/official_releases/qt/5.4/5.4.1/qt-opensource-linux-x64-5.4.1.run)), самостоятельно производите установку. Проверяете работоспособность.  
  
После этого регистрируете собранное вами. Открываем QtCreator, выбираем в главном меню «Инструменты» ► «Параметры» ► «Сборка и запуск». Далее, вся регистрация предусматривает три шага:  
  
1) Регистрация компиляторов (Закладка «Компиляторы»)  
2) Регистрация профилей Qt (Закладка «Профили Qt»)  
3) Регистрация комплектов (Закладка «Комплекты»)  
  
Шаг-1  
  
Добавляя очередной компилятор, выбирайте тип «Особый». Нужный компилятор ищем в $HOME/dev/cross/mxe/usr/bin типа bla-bla-bla-g++. Ниже указываем путь к системной утилите make, типа /usr/bin/make. Далее выставляем ABI. Если выбрали static-версию, внизу добавьте mkspec — «win32-g++.static» (без кавычек). Остальные поля несущественны. Зарегали все четыре вида компиляторов, нажимаем «Применить»  
  
Шаг-2  
  
Переключаемся на закладку «Профили Qt». Добавляя профили ищем их в $HOME/dev/stuff/bla-bla-bla/bin/qmake. Зарегали все четыре профиля, нажимаем «Применить»  
  
Шаг-3  
  
Переключаемся на закладку «Комплекты». Добавляем очередной комплект, выбирая нужный компилятор + профиль Qt. В комплектах, где статическая сборка желательно прописать и mkspecs как на шаге-1. Зарегали все четыре комплекта, нажимаем «Применить»

# Заключение

Вполне отдаю себе отчет — собранное может содержать определенные ошибки, т.к. собиралось далеко не автором ни Qt, ни PostgreSQL, даже не участником проекта MXE. Да и отсутствие простейших проверок во время сборки, не говоря уж о юнит-тестах, предполагает, что вы осознаете «сырость» сделанного. Тем не менее, после получения данного набора тулчейнов, я протестировал свои проекты, примеры из поставки Qt. В интересующих меня модулях (UI, PostgreSQL, http, QtScript, и ряда других) ошибок не обнаружил.  
  
… Обещанное про фулл-сборку. Если вам очень повезет, и MXE, на момент вашего интереса будет в стабильном состоянии, получить фулл сборку очень просто:

$ mkdir -p $HOME/dev/{cross,src,stuff,projects}

$ cd $HOME/dev/cross

$ git clone https://github.com/mxe/mxe.git

$ make MXE\_TARGETS='x86\_64-w64-mingw32.shared' gcc -j4 JOBS=4

$ make MXE\_TARGETS='x86\_64-w64-mingw32.static' gcc -j4 JOBS=4

$ make MXE\_TARGETS='i686-w64-mingw32.shared' gcc -j4 JOBS=4

$ make MXE\_TARGETS='i686-w64-mingw32.static' gcc -j4 JOBS=4

*#*

*# чтобы PostgreSQL-либа была рабочей - положите в каталог $HOME/dev/cross/mxe/src*

*# патчик, полученный из http://majestio.tk/stuff/postgresql-3-socket.patch*

*#*

$ make MXE\_TARGETS='x86\_64-w64-mingw32.shared' qt5 -j4 JOBS=4

$ make MXE\_TARGETS='x86\_64-w64-mingw32.static' qt5 -j4 JOBS=4

$ make MXE\_TARGETS='i686-w64-mingw32.shared' qt5 -j4 JOBS=4

$ make MXE\_TARGETS='i686-w64-mingw32.static' qt5 -j4 JOBS=4

Вангую, по этому варианту сборки вас ждет разочарование с вероятностью 87.14%, но если все будет без косяков, то полученные сборки Qt5 будут размещены в **$HOME/dev/cross/mxe/usr/<тулчейн>/qt5**  
  
Всем удачного кодинга.

**Теги:**

* [Qt](https://habr.com/ru/search/?q=%5BQt%5D&target_type=posts)
* [cross-compile](https://habr.com/ru/search/?q=%5Bcross-compile%5D&target_type=posts)
* [Gentoo Linux](https://habr.com/ru/search/?q=%5BGentoo%20Linux%5D&target_type=posts)
* [toolchain](https://habr.com/ru/search/?q=%5Btoolchain%5D&target_type=posts)

**Хабы:**

* [PostgreSQL](https://habr.com/ru/hub/postgresql/)
* [C++](https://habr.com/ru/hub/cpp/)
* [Qt](https://habr.com/ru/hub/qt_software/)