Building Qt 5 from Git

From Qt Wiki
Jump to: navigation, search

En Ar Bg De El Es Fa Fi Fr Hi Hu It Ja Kn Ko Ms Nl Pl Pt Ru Sq Th Tr Uk Zh

This article provides hints for checking out and building the Qt 5 repositories. This is primarily for **developers who want to contribute to the Qt library itself,** or who want to try the latest unreleased code.

If you simply want to build a specific release of Qt from source to use the libraries in your own project, you can download the source code from the Official Releases (http://download.qt.io/official_releases/qt/) page or the Archive (http://download.qt.io/archive/qt/). Alternatively, commercial customers can download the Source Packages via the Qt Account (https://account.qt.io) portal.

To compile Qt 6, see Building Qt 6 from Git.

To compile Qt Creator, see Building Qt Creator from Git.

Contents

System Requirements

All desktop platforms

- Git (**>= 1.6.x**)
- Perl (**>=5.14**)
- Python (**>=2.6.x**)
- A working C++ compiler

For more detailed information, see Building Qt Sources (http://doc.qt.io/qt-5/build-sources.html)

Documentation (optional)

- Clang (**>=6.x**)

See Installing Clang for QDoc (https://doc.qt.io/qt-5/qdoc-guide-clang.html)

SSL (optional)

See Enabling and Disabling SSL Support (http://doc.qt.io/qt-5/ssl.html#enabling-and-disabling-ssl-support)

For Windows, *bison*, *flex* and *gperf* are provided with the source code at *c:\pathToQt\gnuwin32\bin*. Get Ruby from http://rubyinstaller.org/. You can download the

precompiled ICU packages from download.qt.io (http://download.qt.io/development_releases/prebuilt/icu/prebuilt/), or see Compiling-ICU to compile your own.

Linux/X11

apt-get build-dep

Ubuntu/Debian based systems have a convenient way of installing build-depends for any package:

```
sudo apt-get build-dep qt5-default
sudo apt-get install libxcb-xinerama0-dev
```

RPM-based distros with yum offer a similar tool called yum-builddep.

Note: if you see the following error message while running the commands above on Ubuntu:

```
E: You must put some 'source' URIs in your sources.list
```

Then you will need to enable the "Source code" option in Software and Updates > Ubuntu Software under the "Downloadable from the Internet" section. This setting can also be found by running **software-properties-gtk**.

Convenience packages (Ubuntu 11.10 -- 12.10 only)

For Ubuntu/Debian, Gabor Loki has provided a custom PPA with the sedkit-env-webkit (https://launchpad.net/~u-szeged/+archive/sedkit) meta package that installs all required dependencies for building Qt/Qt WebKit. You can add the PPA by calling:

```
sudo apt-add-repository ppa:u-szeged/sedkit &&\
sudo apt-get update &&\
sudo apt-get install sedkit-env-qtwebkit
```

For other distros, get the separate components below.

Build essentials

Ubuntu and/or Debian:	`sudo apt-get install build-essential perl python git`
Fedora 30	`su - -c "dnf install perl-version git gcc-c++ compat-openssl10-devel harfbuzz-devel double-conversion-devel libzstd-devel at-spi2-atk-devel dbus-devel mesa-libGL-devel"`
OpenSUSE:	`sudo zypper install git-core gcc-c++ make`

Libxcb

Libxcb (http://xcb.freedesktop.org/) is now the default window-system backend for platforms based on X11/Xorg, and you should therefore install libxcb and its accompanying packages. Qt5 should build with whatever libxcb version is available in your distro's packages (but you may optionally wish to use v1.8 or higher to have

threaded rendering support). The README (http://code.qt.io/cgit/qt/qtbase.git/tree/src/plugins/platforms/xcb/README) lists the required packages.

Ubuntu/Debian:	`sudo apt-get install '^libxcb.*-dev' libx11-xcb-dev libglu1-mesa-dev libxrender-dev libxi-dev libxkbcommon-dev libxkbcommon-x11-dev`
Fedora 30:	`su - -c "dnf install libxcb libxcb-devel xcb-util xcb-util-devel xcb-util-*-devel libX11-devel libXrender-devel libxkbcommon-devel libxkbcommon-x11-devel libXi-devel libdrm-devel libXcursor-devel libXcomposite-devel"`
OpenSUSE 12+:	`sudo zypper in xorg-x11-libxcb-devel xcb-util-devel xcb-util-image-devel xcb-util-keysyms-devel xcb-util-renderutil-devel xcb-util-wm-devel xorg-x11-devel libxkbcommon-x11-devel libxkbcommon-devel libXi-devel`
ArchLinux/Manjaro:	`sudo pacman -S --needed libxcb xcb-proto xcb-util xcb-util-image xcb-util-wm libxi`
Chakra Linux:	Install the ArchLinux packages, plus xcb-util-keysyms. It's available from CCR.
Mandriva/ROSA /Unity:	`urpmi 'pkgconfig(xcb)' 'pkgconfig(xcb-icccm)' 'pkgconfig(xcb-image)' 'pkgconfig(xcb-renderutil)' 'pkgconfig(xcb-keysyms)' 'pkgconfig(xrender)'`
Linux Mint:	`apt-get install libx11-xcb-dev libxcb-composite0-dev libxcb-cursor-dev libxcb-damage0-dev libxcb-dpms0-dev libxcb-dri2-0-dev libxcb-dri3-dev libxcb-glx0-dev libxcb-icccm4-dev libxcb-image0-dev libxcb-keysyms1-dev libxcb-present-dev libxcb-randr0-dev libxcb-render-util0-dev libxcb-render0-dev libxcb-shape0-dev libxcb-shm0-dev libxcb-sync-dev libxcb-util-dev libxcb-xfixes0-dev libxcb-xinerama0-dev libxcb-xkb-dev libxcb-xtest0-dev libxcb1-dev`

Centos 5/6	Install missing Qt build dependencies:
	`yum install libxcb libxcb-devel xcb-util xcb-util-devel`
	Install Red Hat DevTools 1.1 for CentOS-5/6 x86_64, they are required due to outdated GCC shipped with default CentOS:
	`wget http://people.centos.org/tru/devtools-1.1/devtools-1.1.repo -O /etc/yum.repos.d/devtools-1.1.repo`
	`yum install devtoolset-1.1`
	Initialise your newly installed dev tools:
	`scl enable devtoolset-1.1 bash`
	`# Test - Expect to see gcc version 4.7.2 (**not** gcc version 4.4.7)`
	`gcc -v`
	For more info on preparing the environment on **CentOS**, see this thread (http://forum.qt.io/topic/30140).

Centos 7	Update to gcc 7:
	`yum install centos-release-scl`
	`yum install devtoolset-7-gcc*`
	`scl enable devtoolset-7 bash`

	Install missing Qt build dependencies (Qt 5.13): `yum install libxcb libxcb-devel xcb-util xcb-util-devel mesa-libGL-devel libxkbcommon-devel`

OpenGL support

For Qt Quick 2, a graphics driver with native OpenGL 2.0 support is highly recommended.

Accessibility

It is recommended to build with accessibility enabled, install **libatspi 2** and **libdbus-1** development packages.

Qt WebKit

Ubuntu/Debian:	`sudo apt-get install flex bison gperf libicu-dev libxslt-dev ruby`
Fedora 30:	`su - -c "dnf install flex bison gperf libicu-devel libxslt-devel ruby"`
OpenSUSE:	`sudo zypper install flex bison gperf libicu-devel ruby`
Mandriva/ROSA /Unity:	`urpmi gperf`

Qt WebEngine

Ubuntu/Debian:	`sudo apt-get install libxcursor-dev libxcomposite-dev libxdamage-dev libxrandr-dev libxtst-dev libxss-dev libdbus-1-dev libevent-dev libfontconfig1-dev libcap-dev libpulse-dev libudev-dev libpci-dev libnss3-dev libasound2-dev libegl1-mesa-dev gperf bison nodejs`
Fedora/RHEL:	`sudo dnf install freetype-devel fontconfig-devel pciutils-devel nss-devel nspr-devel ninja-build gperf cups-devel pulseaudio-libs-devel libcap-devel alsa-lib-devel bison libXrandr-devel libXcomposite-devel libXcursor-devel libXtst-devel dbus-devel fontconfig-devel alsa-lib-devel rh-nodejs12-nodejs rh-nodejs12-nodejs-devel`
OpenSUSE:	`sudo zypper install alsa-devel dbus-1-devel libXcomposite-devel libXcursor-devel libXrandr-devel libXtst-devel mozilla-nspr-devel mozilla-nss-devel gperf bison nodejs10 nodejs10-devel`

Qt Multimedia

You'll need at least alsa-lib (**>= 1.0.15**) and gstreamer (**>=0.10.24**) with the base-plugins package.

Ubuntu/Debian:	`sudo apt-get install libasound2-dev libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev libgstreamer-plugins-good1.0-dev libgstreamer-plugins-bad1.0-dev`

Fedora 30:	`dnf install pulseaudio-libs-devel alsa-lib-devel gstreamer1-devel gstreamer1-plugins-base-devel wayland-devel`

QDoc Documentation Generator Tool

Ubuntu/Debian:	`sudo apt install libclang-6.0-dev llvm-6.0`
Fedora 30:	`su -c 'dnf install llvm-devel'`

macOS

Install the latest Xcode from the App Store. Verify that your Xcode install is properly set up for command line use:

```
xcodebuild -version && xcodebuild -showsdks
```

This should give you eg:

```
Xcode 6.2
Build version 6C131e
```

```
OS X SDKs:
 OS X 10.9                      -sdk macosx10.9
 OS X 10.10                     -sdk macosx10.10
```

```
iOS SDKs:
 iOS 8.2                        -sdk iphoneos8.2
```

```
iOS Simulator SDKs:
 Simulator - iOS 8.2            -sdk iphonesimulator8.2
```

You can verify that the right Xcode is being used by running:

```
xcode-select --print-path
```

If this points to /Developer you're probably using an older Xcode version. Switch to the latest one by running:

```
sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
```

Windows

Windows Graphics Drivers

The full feature set of Qt Quick 2 requires OpenGL 2.1 or higher or OpenGL ES 2.0 to work. Several options are available:

1. Using the native OpenGL driver for your graphics card (Note: The stock Windows driver only supports OpenGL 1.1, which is insufficient).
2. Using the ANGLE-library (http://code.google.com/p/angleproject/) to translate OpenGL calls into *DirectX*. A copy of ANGLE is bundled in Qt 5.
3. Using a software rasterizer shipped in the Qt SDK. See MesaLlvmpipe for instructions on how that is built.

Starting with Qt 5.8, many Qt Quick applications can also be run with the built-in software backend (previously known as the Qt Quick 2D Renderer). This is not the same as running through a full OpenGL software rasterizer, and is a lot more lightweight. See the documentation (https://doc-snapshots.qt.io/qt5-dev/qtquick-visualcanvas-adaptations.html) on how to request a specific backend.

Qt can be built to dynamically choose the OpenGL implementation (see Dynamically Loading Graphics Drivers (http://doc.qt.io/qt-5/windows-requirements.html#dynamically-loading-graphics-drivers)) by passing the option `-opengl dynamic` to `configure`. To build Qt to always use the native OpenGL driver, pass `-opengl desktop`. To build Qt to always use ANGLE, pass `-opengl es2`.

ANGLE dependencies

- **MSVC**
  - Qt < 5.12: Building ANGLE with Windows SDKs prior to Windows Kit 8 requires the DirectX SDK (http://msdn.microsoft.com/en-us/directx/default.aspx) to be installed, and the path to the `d3d_compiler<xx>.dll` to be added to the `PATH` variable.
  - Qt >= 5.12: Qt needs at least a Windows 10 SDK in order to build ANGLE. No additional manual setup is needed
- **MinGW**
  - Building ANGLE with Windows SDKs prior to Windows Kit 8 requires the DirectX SDK (http://msdn.microsoft.com/en-us/directx/default.aspx) to be installed, and the path to the `d3d_compiler<xx>.dll` to be added to the `PATH` variable.

Any project that uses run-time shader compilation must have D3DCOMPILER_xx.DLL copied to the local executable path for the project. This DLL is available in this sub-directory of the Windows SDK installation under `%ProgramFiles(x86)%\Windows Kits\<version>\Redist\D3D\<arch>` where <arch> is x86 and x64 (see [1] (https://msdn.microsoft.com/en-us/library/windows/desktop/ee663275.aspx)).

Supported Compilers on Windows

- Visual Studio 2019
- Visual Studio 2017 (might need a recent service pack, especially for QtWebEngine)
- MinGW-w64 based compiler with *g++* version 7.3 or higher (e.g. MinGW-builds (http://sourceforge.net/projects/mingwbuilds/), see also MinGW-64-bit).

Notes:

- Add the compiler to the PATH environment variable. Visual Studio usually ships .bat files named `vcvarsall.bat` or similar that can be called with a command line

parameter for choosing the toolchain (/x86, /x64, etc) to set up the environment.
- Windows SDK v6.0A/v7.0A contains the same compiler as Visual Studio 2008/2010.
- Windows SDK 8.0 and later do not include a compiler.
- As of 16.3.2012, if you wish to install both Visual Studio 2010 and the standalone SDK, you need to follow this order (see readme.html provided with the service pack):
    1. Install Visual Studio 2010
    2. Install Windows SDK 7.1. See also the Cannot Install Windows SDK page.
    3. Install Visual Studio 2010 SP1
    4. Install Visual C++ 2010 SP1 Compiler Update for the Windows SDK 7.1

Windows Build environment

We recommend creating a command prompt that provides the build environment (see the Qt Creator README (http://code.qt.io/cgit/qt-creator/qt-creator.git/tree/README.md)). In this environment, *Python* (e.g. *Active Python (http://www.python.org/download/) 2.7* later) and *Perl* (e.g. *StrawberryPerl (http://strawberryperl.com/) 5.12* or later) should be in the *PATH*. Ruby (http://rubyinstaller.org/downloads/) is required for WebKit.

Hint: If you installed git with the non-recommended setting to add git's entire Unix environment to PATH, make sure that *Perl* is added to the path in front of *git*, since that ships an incompatible build of perl, which would cause the scripts to fail. Also, MinGW builds of Qt become Msys builds due to the presence of sh.exe; and those who've tried have had more success configuring and building for MinGW in MS's `cmd.exe` Command Prompt than in the Git Bash shell.

Multicore building: To speed up building when using *nmake*, the compiler can be instructed to use all available CPU cores in one of the following ways:

- Pass the option *-mp* to Qt's *configure*
- Set the environment variable *CL* (specifying Visual Studio compiler options) to */MP* (On the command line: `set CL=/MP`)
- Use the tool jom instead of nmake. (Using jom instead of nmake reduces compile time quite a bit)

Configuring Visual Studio 2013 on Windows 8, 8.1 & 10

Setting the environment variables to properly build Qt can be done by following the steps below:

- Create a file called qt5vars.bat, paste the following inside it and save it

Hint: Remember to change <arch> to your desired platform and double-check that the paths are correct for Qt and Visual Studio

```
@echo off

REM Set up \Microsoft Visual Studio 2015, where <arch> is \c amd64, \c x86, etc.
CALL "C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\vcvarsall.bat" <arch>

REM Edit this location to point to the source code of Qt
SET _ROOT=C:\qt5
```

```
SET PATH=%_ROOT%\qtbase\bin;%_ROOT%\gnuwin32\bin;%PATH%

REM Uncomment the below line when using a git checkout of the source repository
SET PATH=%_ROOT%\qtrepotools\bin;%PATH%

REM Uncomment the below line when building with OpenSSL enabled. If so, make sure the directory points
REM to the correct location (binaries for OpenSSL).
REM SET PATH=C:\OpenSSL-Win32\bin;%PATH%

REM When compiling with ICU, uncomment the lines below and change <icupath> appropriately:
REM SET INCLUDE=<icupath>\include;%INCLUDE%
REM SET LIB=<icupath>\lib;%LIB%
REM SET PATH=<icupath>\lib;%PATH%

REM Contrary to earlier recommendations, do NOT set QMAKESPEC.

SET _ROOT=

REM Keeps the command line open when this script is run.
cmd /k
```

Complete the steps below after you have cloned Qt5 from Git

- After having cloned Qt5 from Git (assuming it's at C:\Qt5), move qt5vars.bat to the Qt5 folder
- Double-click on the script to run it. Once it runs the script, the command prompt will stay open to enter additional commands.
- Navigate to C:\Qt5 in the command prompt

The working directory should be c:\Qt5.

All the required environment variables are now correctly set up and building Qt5 with nmake should work now. You have to run qt5vars.bat and use the command prompt every time you build Qt5 with nmake.

The above steps also work for setting up nmake to build jom.

Configuring MinGW on Windows 8, 8.1 & 10

The configuration process is similar to above except for a few minor changes in the qt5vars.bat script:

```
@echo off

REM Edit this location to point to the source code of Qt
SET _ROOT=C:\qt5

SET PATH=%_ROOT%\qtbase\bin;%_ROOT%\gnuwin32\bin;%PATH%

REM Uncomment the below line when using a git checkout of the source repository
SET PATH=%_ROOT%\qtrepotools\bin;%PATH%

REM Uncomment the below line when building with OpenSSL enabled. If so, make sure the directory points
REM to the correct location (binaries for OpenSSL).
REM SET PATH=C:\OpenSSL-Win32\bin;%PATH%

REM When compiling with ICU, uncomment the lines below and change <icupath> appropriately:
REM Note that -I <icupath>\include and -L <icupath>\lib need to be passed to
REM configure separately (that works for MSVC as well).
REM SET PATH=<icupath>\lib;%PATH%

REM Contrary to earlier recommendations, do NOT set QMAKESPEC.

SET _ROOT=
```

```
REM Keeps the command line open when this script is run.
cmd /k
```

ICU on Windows

Qt 5 can make use of the ICU (http://site.icu-project.org/) library for UNICODE and Localization support. This is **required** for building Qt WebKit. You can use pre-compiled versions of ICU with a Visual Studio 2010 dependency from the website, get pre-compiled versions for different compilers from download.qt.io (http://download.qt.io/development_releases/prebuilt/icu/prebuilt/), or compile ICU on your own.

The absolute paths of *include* and *lib* folders of the ICU installation must be passed with -I and -L option to Qt's configure. In addition, `uic.exe` needs to find the ICU DLLs during compilation, for which the *lib* folder of the ICU installation must be added to the *PATH* environment variable.

At run-time, the ICU DLLs need to be found. This can be achieved by copying the DLLs to the application folder or adding the *lib* folder of the ICU installation to the *PATH* environment variable.

Getting the source code

First clone the top-level Qt 5 git repository:

```
$ git clone git://code.qt.io/qt/qt5.git
```

or (if you're behind a firewall and want to use the https protocol):

```
$ git clone https://code.qt.io/qt/qt5.git
```

Then check out the target branch (see Branch Guidelines):

```
$ cd qt5
$ git checkout 5.12
```

Getting the submodule source code

As described in the README.git (http://code.qt.io/cgit/qt/qt5.git/tree/README.git), initialize the repository using the `init-repository` script, which clones the various sub-modules of Qt 5.

Relevant options for `init-repository`:

- `--module-subset=default,-qtwebengine` : **Consider skipping the web module** (Qt WebEngine) by passing this option. It is quite big and takes a long time to compile (and is often a source of compile errors), so it is recommended to only download it if you intend to use it. You can always re-run init-repository later on to add it.

- `--branch` : Check out the branch tips instead of the SHA1s of the latest successful integration build.
- `--codereview-username <Jira/Gerrit username>` : If you plan to contribute to Qt, you may specify your codereview username (pay attention to capitalization!) so that the git remotes are properly set up. Note that it is recommended to adjust your ssh configuration instead.

```
$ cd qt5
$ perl init-repository
```

In order to build a specific release of Qt, you can checkout the desired tag:

```
$ cd qt5
$ git checkout 5.12.0
$ perl init-repository
```

Note: `init-repository` is currently unable to initialize tags that are too old. An alternative way to build a specific release or branch of Qt5 (although without linking of the gerrit account for code reviewing) is to use `git submodule update --init` in place of the `init-repository` script. That translates to:

```
$ git clone https://code.qt.io/qt/qt5.git                    # cloning the repo
$ cd qt5
$ git checkout v5.8.0                                        # checking out the specific release or branch
$ git submodule update --init --recursive                    # updating each submodule to match the
supermodule
```

More information can be found in Get The Source.

Configuring and Building

The Qt5 build system should be fairly resilient against any "outside distractions" - it shouldn't matter whether you have other Qt versions in PATH, and QTDIR is entirely ignored. However, make sure that you have no qmake-specific environment variables like QMAKEPATH or QMAKEFEATURES set, and the qmake -query output does not refer to any other Qt versions ($HOME/.config/Qt/QMake.conf should be empty).

For more configure options, see Qt Configure Options (http://doc.qt.io/qt-5/configure-options.html).

Note: To build qdoc and Qt documentation in future you should set LLVM_INSTALL_DIR environment variable pointing to directory where LLVM is installed (it should be top level directory, the configuration scripts use relative path tracing from this directory). For example, in Linux with LLVM installed in isolated directory (/usr/llvm), at a bash prompt:

```
$ export LLVM_INSTALL_DIR=/usr/llvm
```

A build script called `configure` (or `configure.bat` for Windows) will be in the directory that you git cloned the source code into (`~/qt5` if you followed the directions above). You will want to call that script from a different, parallel-level directory, because (unless you are using a Qt Autotest Environment (https://wiki.qt.io/Qt_Autotest_Envir

onment#Shadow_builds)) you do not want to build Qt in the directory that holds the source code. Instead, you should use a "shadow build," meaning you should not build into the source directory.

For Linux / macOS, to install in `~/qt5-build` (an arbitrary name), assuming you are in ~:

```
$ mkdir qt5-build
$ cd qt5-build
$ ../qt5/configure -developer-build -opensource -nomake examples -nomake tests
```

For Windows, again assuming you are in ~:

```
$ mkdir qt5-build
$ cd qt5-build
$ ..\\qt5\configure -developer-build -opensource -nomake examples -nomake tests
```

The `-developer-build` option causes more symbols to be exported in order to allow more classes and functions to be unit tested than in a regular Qt build. It also defaults to a 'debug' build, and installs the binaries in the current directory, avoiding the need for 'make install'. '-opensource' sets the license to be GPL/LGPL. The `-nomake examples` and `-nomake tests` parameters make sure examples and tests aren't compiled by default. You can always decide to compile them later by hand.

Some Hints

1. You can add `-confirm-license` to get rid of the question whether you agree to the license.
2. On Windows, you might not be able to build if `sh.exe` is in your *PATH* (for example due to a *git* or *msys* installation). Such an error is indicated by `qt5-src\qtbase\bin\qmake.exe: command not found` and alike. In this case, make sure that `sh.exe` is *not* in your path. You will have to re-configure if your installation is already configured.
3. If you are planning to make contributions with git, it is advised to configure and build the source in a separate directory so the binaries are not seen by git. This can be done by navigating to the desired build directory and calling configure/make in that directory. (But shadow builds are discouraged in Qt Autotest Environment (https://wiki.qt.io/Qt_Autotest_Environment#Shadow_builds).)
    1. *Note:* If you do a shadow build, follow the instructions above for creating a build directory that is parallel to the source directory. Do **NOT** make the build directory a subdirectory of the source tree. The build will probably fail in weird ways if the build directory is not exactly parallel to the source directory.

Now trigger the build from within the build directory by running:

```
$ make -j$(nproc)
```

For Windows (MSVC), choose one of the following, depending on your setup/environment:

```
$ nmake
```

or

```
$ jom
```

or

```
$ mingw32-make
```

Or only build a specific module, e.g. declarative, and modules it depends on:

```
$ make module-qtdeclarative
```

Building Qt WebKit

Windows

WebKit.org (http://trac.webkit.org/wiki/BuildingQtOnWindows) has instructions for building WebKit on Windows. ICU (http://site.icu-project.org/) is required for building.

The tools *bison*, *flex* and *gperf* which are required for building are provided for convenience in the folder *gnuwin32\bin*. If you are using shadow builds, you must add this directory to your `PATH`, else no special actions need to be done manually in order to use them.

Installing (Linux / OS X)

- **Note:** Installation is only needed if you haven't used the configure options `-developer-build` or `-prefix "%PWD%/qtbase"`. Otherwise, you can just use Qt from the build directory.

To install, run

```
$ make install
```

Cleaning

To get a **really** clean tree use:

```
$ git submodule foreach --recursive "git clean -dfx" && git clean -dfx
```

since make confclean no longer works from the top-level of the repo.

Getting updates

To update both the qt5.git repo as well as the submodules to the list of revisions that are known to work, run

```
$ git pull
$ perl init-repository -f
```

In addition, you should pass the same parameters to init-repository as you did in Getting the source code.

Unlike a "normal" git submodule update, this ensures that any changes to the module structure are automatically pulled as well.

If you are planning to do nightly builds, consider using the script *qt5_tool* that lives in *qtrepotools/bin*. It provides options for updating the repository, cleaning and building. For example, `qt5_tool -u -c -b` can be used to clean, update and build. `qt5_tool -p -c -b` would be used to pull all modules to the head of their master branches.

Depending upon what changed in the source since it was last updated you might have to run configure again. The safe thing to do is to always run `config.status -recheck-all` in the build directory after updating.

- **Hint1:** The submodule update does a checkout in submodules, potentially hiding any local commits you've done. If the latter happened to you (and you haven't been working with branches anyhow), `git reflog` is your friend.
- **Hint2:** When creating scripts for updates on Windows, note that `git clean` often fails if some process locks a file or folder.

Using latest branches in the submodules

By default, the checkout will not contain the latest stable/dev branches of each individual submodule repository, but a combination of versions that are known to work together. If you want to get the absolute latest stuff you can do so on a per-module basis, e.g.

```
$ cd qtdeclarative
$ git fetch
$ git checkout -b 5.9 origin/5.9
```

or tell init-repository to check out branches in all repositories:

```
$ perl init-repository -f --branch
```

Some Unix shell tricks for developing Qt can be useful when you are making or reviewing changes in multiple modules.

Issues

Linux

configure fails with "No QPA platform plugin enabled!" (Linux)

You should install the libxcb and it's accompanying packages, see 'System Requirements'.

configure fails with errors like "cannot stat file ..."

Your perl version is too old, Qt 5 beta1 needs at least 5.14.

configure fails to enable qdoc on Debian

Debian allows parallel installs of libclang-<version>-dev in /usr/lib/llvm-<version> so LLVM_INSTALL_DIR should be /usr/lib/llvm-<version> to build qdoc. Not /usr/llvm.

qmlscene segfaults "Cannot create platform GL context, none of GLX, EGL, DRI2 is enabled" (Linux)

Try installing the libx11-xcb-dev package:

```
$ sudo apt-get install libx11-xcb-dev
```

afterwards you have to re-run configure and force qtbase/src/plugins/platforms/xcb to recompile.

WebKit doesn't compile, missing ICU

Currently there is no configure time check for ICU, so install it through the package manager through

on Ubuntu/Debian:

```
$ sudo apt-get install libicu-dev
```

on Fedora:

```
$ su - -c "yum install libicu-devel"
```

- You can also compile Qt without Qt WebKit by deleting / renaming the qtwebkit, qtwebkit-examples-and-demos directories.
- The --no-webkit option of configure added, see QTBUG-20577 (https://bugreports.qt.io/browse/QTBUG-20577) issue.

Qt D-Bus fails to build due to "inconsistent user-defined literal suffixes"

This occurs when you attempt to build Qt 5 with GCC 4.7 while D-Bus < 1.4.20 is present on your system. (For example, the default Fedora 17 installation is prone to this error.) The error message is this:

```
qdbusinternalfilters.cpp:124:36: error: inconsistent user-defined literal suffixes
```

```
‘DBUS_INTROSPECT_1_0_XML_PUBLIC_IDENTIFIER’ and ‘DBUS_INTROSPECT_1_0_XML_SYSTEM_IDENTIFIER’ in string
literal
```

Note: The error is in the header files of D-Bus itself, and it has been fixed upstream, see https://bugs.freedesktop.org/show_bug.cgi?id=46147.

Solution: either upgrade to a newer version of D-Bus or edit that one line of the header file manually.

...::isNull is not defined (from qvariant_p.h)

C++11 support is detected while your GCC doesn't properly support it. Fixed by passing `-no-c11` to the configure options.

cc1: fatal error: .pch/release-shared/QtGui: No such file or directory

Currently unresolved bug with the build of assembly files, see discussion at http://comments.gmane.org/gmane.comp.lib.qt.devel/5933
Fixed by passing -no-pch to the configure options.

ld: hidden symbol `void QQmlThread::postMethodToThread<QQmlDataBlob*, QQmlDataBlob*, QQmlDataLoaderThread>(void (QQmlDataLoaderThread::)(QQmlDataBlob), QQmlDataBlob* const&)' isn't defined

Bug with GCC versions < 4.4.x, see bug report at https://bugzilla.redhat.com/show_bug.cgi?id=493929. Fixed by adding `QMAKE_CXXFLAGS_RELEASE *= -fno-inline` in *qtdeclarative/src/qml/qml.pro*.

Touchscreen (or Wacom tablet) doesn't work

Qt depends on having libxi (including development headers), supporting XInput protocol 2.2 or higher, available at build time in order to have multi-touch support. Otherwise configure will fall back to XInput 2.0 which does not support touchscreens. To prove that is the problem, try this:

```
$ export QT_XCB_DEBUG_XINPUT_DEVICES=1
```

and try your Qt application again. At startup, Qt will enumerate the input devices available, like this

```
XInput version 2.2 is available and Qt supports 2.2 or greater
input device … (keyboard, mouse etc.) …
input device Advanced Silicon S.A CoolTouch™ System
has valuator "Abs MT Position X" recognized? true
has valuator "Abs MT Position Y" recognized? true
has touch class with mode 1
it's a touchscreen with type 0 capabilities 0x21 max touch points 10
```

If it does not say Qt supports 2.2 or greater, it means the headers weren't available when Qt was built, so the support for touch was not included. If you do have 2.2 or greater but it doesn't say it's a touchscreen at the end, there may be some other problem such that the touchscreen is not recognized, and you may want to write up a bug, after verifying that touch works in other X11 applications.

Windows

Note that if you're shadow-building Qt, the source directory and build directory must be on the same drive, nested equally deeply. Also, make sure there are *no* old build artifacts in the source directory.

Debugging OpenGL issues (Windows)

Set the environment variable `QT_QPA_VERBOSE=gl:1` and run the application with DebugView (http://technet.microsoft.com/en-us/sysinternals/bb896647) installed. The log will show the requested vs obtained OpenGL version. If the log tells you that it only has OpenGL 1.1, Qt Quick 2 will not work. Note that *qmlscene* will not report errors about unsupported OpenGL versions.

Questions and comments

Please raise questions & comments about this article in this forum thread (http://forum.qt.io/topic/6861/qt5-development-primer).