Script to build the above.

Requires:
https://github.com/swampdawg/go
https://github.com/swampdawg/qtc #(this repo)

	Prebuilt	Sources
Contents:	**QtCreator 11.0**	**QtCreator 14.01**
icu:	72.1	72.1
z3:	4.13.0	4.13.0
llvm:	*18.1.6*	18.1.8
cmake:	*3.24.3*	3.30.3
ninja:	*1.11.1*	1.12.1
mariadb:	11.4.2	11.4.2
qt:	*6.5*	6.7.2
qtc:	*11.0*	14.0.1
gcc:	14.2.0	14.2.0
node:	16.17.0	16.17.0
md4c:	0.4.8	0.4.8
doxygen:	1.9.5	1.9.5
openocd:	0.0.0 #?	0.0.0 #?
mosquitto:	2.0.15	2.0.15
qt6ct:	0.10	0.10
	Pico cross compiler	
xgcc:	(pico arm-none-eabi)	(pico arm-none-eabi)
binutils:	2.43	2.43
gcc:	14.2.0	14.2.0
newlib:	4.4.0.20231231	4.4.0.20231231
xgcc: (*1)	(pico riscv32-unknown-elf)	(pico riscv32-unknown-elf)
binutils:	2.43	2.43
gcc:	14.2.0	14.2.0
newlib:	4.4.0.20231231	4.4.0.20231231
	Source	
tarball:	n/a (*2)	all.tar

Italics above denote an older version.

(*1) Untested. Author does not own a pico2 at time of writing.
(*2) Not enough space to upload.

The prebuilt binary can be used (a) as-is or (b) used to build the sources. Alternatively (c), the sources can be built from scratch.

a) For those with no large project compilation experience.
b) For those who desire to get the sources built quicker.
c) For those who want to work it all out for themselves.

Pick your platform

x86_64:	qtc-(sdu-linuxmint-21.3-virginia)-11.0-bin.tar.xz
md5sum:	5feee878ad06422fd0a5131e3f3d107f
dev: (*1)	qtc-(sdu-linuxmint-21.3-virginia)-11.0.dev
ldd: (*2)	qtc-(sdu-linuxmint-21.3-virginia)-11.0.ldd
aarch64:	qtc-(pi24-debian-12-bookworm)-11.0-bin.tar.xz
md5sum:	45bffc8c8f63cbec871e2ea95c18e7d4
dev: (*1)	qtc-(pi24-debian-12-bookworm)-11.0.dev
ldd: (*2)	qtc-(pi24-debian-12-bookworm)-11.0.ldd

Source

all.tar	
md5sum:	92d64850d7f6249e04c8f4c9c63d8171

(*1) QtCreator will launch(*2) without these packages but will not be able to build much without at least some of these. This is simply a list of dev packages on the build box.
(*2) List of QT runtime dependencies. Search ‘sdldd’ below.

Installation

Binary unpacks to /usr/local/QT/6500r/. This can be changed when it is rebuilt. If you intend to use it as-is, aka (a) above, then do not move it.

Installation for (a): download qtc*-bin.tar.xz to some folder then..

```
$ sudo mkdir /usr/local/QT/
$ sudo chown `id -un`:`id -gn` /usr/local/QT/
$ tar -C / -xvJf [tarball]
$ pushd /usr/local/
$ sudo ln -s QT/6500r qt
$ popd
$ cd ~
$ mkdir -p bin
$ pushd bin
$ ln -s /usr/local/QT/6500r/bin/sd-qt
$ popd
```

You should refer to the installation via the /usr/local/qt/* symlink in your projects and so forth rather than /usr/local/QT/6500r/*. Both will work but should you decide to rebuild it yourself, aka (b) above, then the default installation path is /usr/local/qt/ and will save amending project paths at a later date.

Invocation:

```
#check QT works..
$ sd-qt assistant #wait for indexing to complete then quit
$ sd-qt designer  #then quit
```

Note:
QtCreator stores settings in ~/.config/QtProject/ should you desire to take steps to preserve settings from an older version.

Now for QtCreator itself..

```
$ sd-qt qtcreator &
```

The author likes..
Edit→Preferences→Environment[Interface]:Theme=”Flat Dark”.
The fonts may be ugly. Try..

```
$ sd-qt qt6ct
```

Sources

Some sources are verbatim. Some have been generated from github repos. Some are verbatim and merely repacked to fit the traditional tarball naming scheme.

Dependencies:
*.dev
*.ldd

As noted earlier, the "dev" list are those packages on the author's machine at the time. Some may not be relevant to the build. The omission of a dev package might be important. We are, after all, building "latest" code on a system with older stuff thus the existence of a system dev package can break things. It depends on the age gap between your build environment and how new what you're trying to build is. Larger the gap, more likely to be issues.

The "ldd" list exists to give an idea of the dependencies to be aiming for. The 'sdldd' tool looked at every qtc-*bin.tar.xz executable and dynamic lib and attempted to build a list of all system dependencies. It's not bad but isn't wholly exhaustive. It is also the one item for which there is no source. This is low down to the "2do" list. For those who don't like it, delete it. Its only function is to produce the above two files.

Instructions for, aka (b), above. Assuming you've done (a) then remove the symlink and replace it with a directory for it is the default build installation target..

```
$ pushd /usr/local/
$ sudo rm -v qt
$ sudo mkdir qt
$ sudo chown `id -un`:`id -gn` qt
$ popd
# install all the dev packages as hinted above.
$ B_QT=/usr/local/QT/6500r D_QT=/usr/local/qt ./go all-bootstrap
#^^^the above takes a very long time and is a waste of time out of the gate.
#^^^only do it once you know the build will work. hint: 'screen' ;-)
```

More methodically:
Edit 'go' and change..

```
: ${D_QT:="/usr/local/QT/6500r"}
```

..to..

```
: ${D_QT:="/usr/local/qt"}
```

Now you can..

```
$ B_QT=/usr/local/QT/6500r ./go bootstrap
#^^^will build a minimal build environment. If this fails you'll be missing
#^^^something fundamental. Start with understanding fcp_bootstrap function.
#^^^look in relevant log/*/*.log file.
```

eg: 'llvm' fails:

```
$ B_QT=/usr/local/QT/6500r ./go llvm del obj
#^^^delete OBJ
$ B_QT=/usr/local/QT/6500r ./go llvm cfg
#^^^repeat until prerequisites solved.
$ B_QT=/usr/local/QT/6500r ./go llvm mak -j$NPROC
#^^^ditto then manually install..
$ B_QT=/usr/local/QT/6500r ./go llvm ins
$ B_QT=/usr/local/QT/6500r ./go llvm del all
#^^^delete OBJ and SRC
```

Check it runs to completion..

```
$ B_QT=/usr/local/QT/6500r ./go llvm all
```

Once all the phases are complete, check it works from scratch..

```
$ B_QT=/usr/local/QT/6500r ./go trash-target
$ B_QT=/usr/local/QT/6500r ./go bootstrap
```

With "bootstrap" working flawlessly it is then possible to move onto..

```
$ ./go all
```

..basically repeating the debug approach outlined above. Finally it is possible to return to..

```
$ B_QT=/usr/local/QT/6500r ./go all-bootstrap
#^^^only once this succeeds is it possible to remove /usr/local/QT/6500r/ and
#^^^rely on /usr/local/qt/ alone.
```

Footnotes

1) Linux version used is 64Gb mint 21.3 virginia with 1Gb paging.
2) Raspberry Pi version used is 8Gb rpi5 aarch64 bookworm with 16Gb paging.
   It was possible to use 8Gb rpi4 bullseye but expect an "all-bootstrap"
   to take a week. Takes about 12hrs on author's PC. Approx 3days on rpi5.
3) Most common cause of (seemingly) random build failures is gcc exhausting
   memory, especially when linking collides. Most of the build strives to use
   the clang compiler.
4) It may be tempting to disable F_GO_TMPFS but just set up 16Gb of paging
   before commencing. Your solid state device will thank you. At a push it is
   possible to plug in a thumbdrive and use that as transient swap. Not a
   recommended method because it can actually be quite hard to diagnose when a
   thumbdrive fails - they go bad in weird ways.
5) Pico2 riscv compiler is untested. Author doesn't even have a pico2 atm!
6) The usual workflow, when "all-bootstrap" works, is:

a) Use /usr/local/QT/*/ to build /usr/local/qt/

b) Destroy /usr/local/QT/*/ and use to B_QT=/usr/local/qt to start on next version of /usr/local/QT/*/

c) When /usr/local/QT/*/ is verified, backup /usr/local/qt/, wipe it and rebuild /usr/local/QT/*/ into /usr/local/qt/.

e) eg:

B_QT=/usr/local/QT/6500r D_QT=/usr/local/qt ./go trash-target
B_QT=/usr/local/QT/6500r D_QT=/usr/local/qt ./go all-bootstrap
(look in ./BIN/ and rename/archive that elsewhere)
mkdir /usr/local/QT/6700r
(upgrade packages)
B_QT=/usr/local/qt D_QT=/usr/local/QT/6700r ./go trash-target
B_QT=/usr/local/qt D_QT=/usr/local/QT/6700r ./go all-bootstrap

7) Installation folders are currently 18Gb each.
8) The use of "tar -xvJf" to produce .xz is painfully painfully slow but it
   does substantially reduce the size of the binary so we're going to live
   with it.

~~2do
---
1) Experiment with "install-strip".
2) Say something about PKG-VER.patch.???
3) Learn some *.md stuff!
^^^scratch that, make a pdf instead!
4) picotool