TQ Systems Yocto Project Tutorial

Release 1.x-g476b0ea

TQ Systems Community BSP Team

January 23, 2014

1 Build and Boot your TQ Systems Yocto Image **3**

2 Yocto Folders **5**

3 Yocto Architecture **7**
3.1 Yocto Release History 7

4 Bitbake Metadata **9**
4.1 Layers 9
4.2 Configuration Data 9

5 Creating a new Layer **15**

6 Patching the Linux Kernel **17**

7 Building the Kernel Manually **19**

8 Contributing to the TQ Systems Yocto Project **21**

Summary

Version 1.x
Release 1.x-g476b0ea
Date January 23, 2014
Status some mature, some in progress
Contact Stephan Linz <linz@li-pro.net>
Organization Li-Pro.Net
Address Jena, Germany

Authors TQ Systems Community BSP Team
- Stephan Linz <linz@li-pro.net>
Abstract This document has the Yocto tutorial of the TQ Systems Community BSP 1.x which is a communty effort to start quick with it and learn the build and design workflow.
Target users, developers and integrators

Revision History

Release	Date	Name	Position	Modification
1.x	2013-11-06	Linz		Document created.

Involved Components

- TQ Systems embedded modules
- Yocto related
- General tutorial

Legal Notice

```
Creative Commons Attribution-ShareAlike 3.0 Unported
```

For details of the terms and definitions, representations, warranties and disclaimer see the file LICENSE that comes with the documentation and/or read the online version (CC-BY-SA-3.0).

You are free:

to Share —to copy, distribute and transmit the work

to Remix —to adapt the work

to make commercial use of the work

Under the following conditions:

Attribution —You must attribute the work in the manner specified by

the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike —If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

Waiver —Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain —Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights —In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Things Todo

Todo
The issue *Add the Layer* to the `build/conf/bblayers.conf` file (modify this file) and *versioning new custom layer with Git* should describe more precise in focus of the underlayed `repo` tool. **Maybe that this part of the tutorial is not proper!**

(The *original entry* is located in /hsj/projects/2014/Yocto/lipro/tqs-community-docs/src/yocto-tutorial/layer.rst, line 55.)

CHAPTER

ONE

BUILD AND BOOT YOUR TQ SYSTEMS YOCTO IMAGE

- Check required packages for your Linux Distribution and install them.
- Install the repo utility following these steps:

```
$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > \
  ~/bin/repo
$ chmod a+x ~/bin/repo
$ PATH=${PATH}:~/bin
```

- Download the BSP metadata (recipes + configuration files + classes):

```
$ mkdir tqs-community-bsp
$ cd tqs-community-bsp
tqs-community-bsp $ repo init \
                    -u https://github.com/lipro/tqs-community-bsp-platform \
                    -b dylan
tqs-community-bsp $ repo sync # Takes some minutes the first time
```

- Select your machine and prepare the bitbake's environment:

```
# To list all TQS related machines, type
tqs-community-bsp $ find sources/meta-tqs* -name "*.conf" | grep "conf/machine"
tqs-community-bsp $ MACHINE=<selected machine> . ./setup-environment build
# if MACHINE is not set, the default machine is 'qemuarmv6'
build $
```

- Choose an image and bake it!

```
build $ bitbake-layers show-recipes | grep image    # To list all possible images
build $ bitbake <selected image>                     # Bake! The first time can
                                                     # take several hours.
# e.g. bitbake core-image-minimal
```

- Boot (e.g. *core-image-minimal*) on the machine *qemuarmv6* with Yocto's **runqemu**:

```
build $ MACHINE=qemuarmv6 runqemu \
        tmp/deploy/images/zImage-qemuarmv6.bin \
        tmp/deploy/images/core-image-minimal-qemuarmv6.ext3
```

- Flash SD Card for machines other than *qemuarmv6*:

> **Warning:** The issue *Flash SD Card* needs to be evaluated! Do not yet apply this description!

```
# Insert your SD Card
# Type '$ dmesg | tail' to see the device node being used, e.g /dev/sdb)
# In case SD to be flash has already some partitions, the host system may have
# mounted these, so unmount them, e.g. '$ sudo umount /dev/sdb?'.
build $ ls -la 'tmp/deploy/images/*.sdcard'

# Flash the soft link one
build $ sudo dd \
        if=tmp/deploy/images/<selected image>-<select machine>.sdcard \
        of=/dev/sdX \
        bs=1M
build $ sync
```

- Place your SD Card in the correct board's slot and boot!

Found Errors? Subscribe and report it to the author <linz@li-pro.net> with subject ``[tqs-community-bsp] Error report: <your_msg>''.

CHAPTER

TWO

YOCTO FOLDERS

tqs-community-bsp Base (BASE) directory where all Yocto data resides (recipes, source code, built packages, images, etc).

BASE/sources Source (SOURCE) directory where metadata (layers) resides.

BASE/build Build (BUILD) directory where **bitbake** commands are executed.

BASE/build/tmp Target (TMP) directory for all bitbake commands.

BASE/build/tmp/work Working (WORKING) directory for recipes tasks.

BASE/build/tmp/deploy Deploy (DEPLOY) directory where bitbake's output data is found.

BASE/build/tmp/deploy/images Complete and partial images are found under this folder.

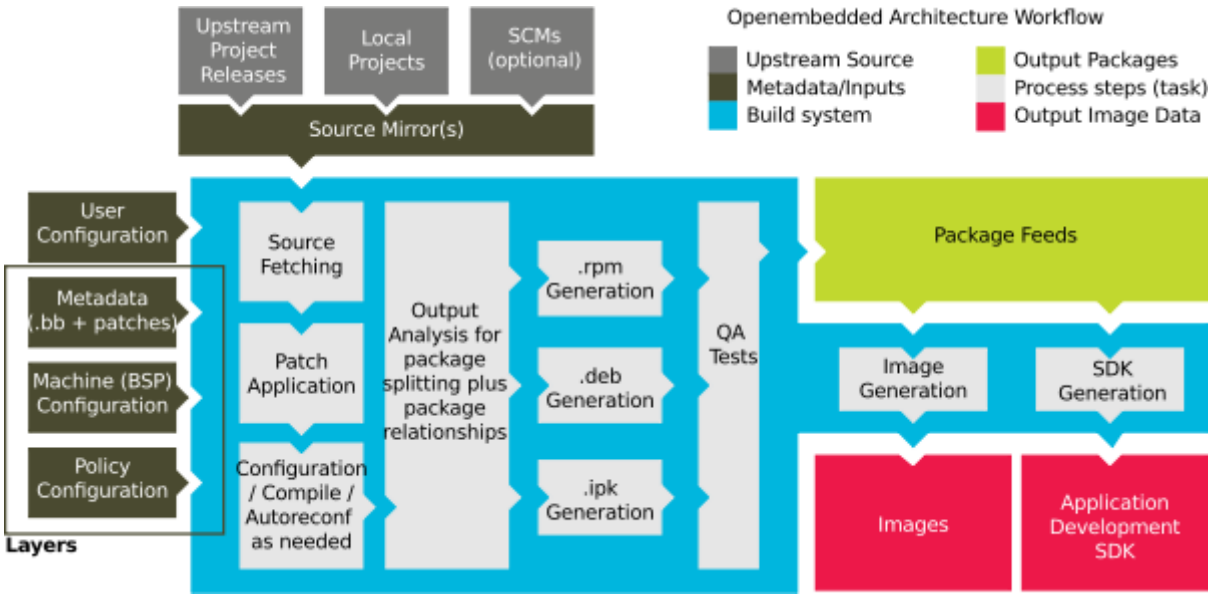
BASE/build/tmp/deploy/sdk Complete SDK installation binaries are found under this folder.

CHAPTER

THREE

YOCTO ARCHITECTURE

See introducing the Yocto Project and the general Yocto Project Development Environment figure for more detailed informations:



3.1 Yocto Release History

See Yocto Project Releases for more detailed (and latest) informations:

Name	Date	Yocto	Poky	BB	Links
	2014-04-25	1.6 [2]			Features, Schedule, Status, Test Plan, Test Report
Dora	2013-10-18	1.5	10.0	1.20	Features, Schedule, Status, Test Plan, Test Report
Dylan	2013-04-26	1.4	9.0	1.18	Features, Schedule, Status, Test Plan, Test Report
Danny	2012-10-26	1.3	8.0	1.16	Features, Schedule, Status, Test Plan, Test Report
Denzil	2012-04-27	1.2	7.0	1.15	Features, Schedule, Status, Test Plan, Test Report
Edison	2011-10-06	1.1	6.0	1.13	Features, Schedule, Status, Test Plan, Test Report
Bernard	2011-04-06	1.0	5.0	1.11	Features, Schedule, Status, Test Plan
Laverne		0.9	4.0	1.11	
Purple			3.2		
Pinky			3.1		
Blinky			3.0		
Clyde			2.0		
Inky			1.0		

[1] in planning
[2] in planning

CHAPTER

FOUR

BITBAKE METADATA

BitBake handles the parsing and execution of the data files. The data itself is of various types:

- Recipes: Provides details about particular pieces of software.
- Class Data: Abstracts common build information (e.g. how to build a Linux kernel).
- Configuration Data: Defines machine-specific settings, policy decisions, and so forth. Configuration data acts as the glue to bind everything together.

4.1 Layers

- Metadata is organized into multiple *layers*.
- Layers allow you to isolate different types of customizations from each other.
- DO NOT do your modifications in existing layers, instead create a layer and create recipes (`.bb` files) or modified existing ones (`.bbappend` files)

4.2 Configuration Data

- build/conf/local.conf: Local User Configuration for your build environment
- build/conf/site.conf: Local Shared (site wide) Configuration for your build environment
- build/conf/bblayers.conf: Define layers, which are directory trees, traversed by BitBake.
- sources/meta-*/conf/layer.conf: Layer configuration file
- sources/meta-*/conf/machine/*.conf: Machine configuration files

4.2.1 Build's local configuration file `build/conf/local.conf`

```
MACHINE ??= 'tqma35'
DISTRO ?= 'poky'
#PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES = "debug-tweaks"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS = "\
    STOPTASKS,${TMPDIR},1G,100K \
```

```
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K"
CONF_VERSION = "1"

BB_NUMBER_THREADS = '4'
PARALLEL_MAKE = '-j 4'
ACCEPT_FSL_EULA = ""
```

Important variables:

- MACHINE: Indicates the target machine, *qemuarmv6* is the default.
- BB_NUMBER_THREADS and PARALLEL_MAKE: Indicate the max number of threads when baking and compiling.

4.2.2 Build's local configuration file `build/conf/site.conf`

```
SCONF_VERSION = "1"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"
DL_DIR = "${BSPDIR}/downloads/"
```

Important variables:

- DL_DIR: Tarball repository. Several users can share the same folder, so data can be reused.

4.2.3 Build's layer configuration file `build/conf/bblayers.conf`

- Automatically created by the **setup-environment** script (see section *Build and Boot your TQ Systems Yocto Image*)
- Only modified when adding a new layer:

```
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"

BBFILES ?= ""
BBLAYERS = " \
  ${BSPDIR}/sources/poky/meta \
  ${BSPDIR}/sources/poky/meta-yocto \
  \
  ${BSPDIR}/sources/meta-fsl-arm \
  \
  ${BSPDIR}/sources/meta-tqs-arm \
"
```

4.2.4 Layer configuration file source/meta-tqs-arm/conf/layer.conf

```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have a packages directory, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "tqs-arm"
BBFILE_PATTERN_tqs-arm := "^${LAYERDIR}/"
BBFILE_PRIORITY_tqs-arm = "6"
```

Important variables:

- BBFILES: Indicates where to look for .bb* files
- BBFILE_PRIORITY_tqs-arm: Indicates layer's priority
- MIRRORS: Indicates where to get the source code

4.2.5 Machine configuration file: meta-tqs-arm/conf/tqma35.conf

```
#@TYPE: Machine
#@NAME: TQ System i.MX35 Embedded module (tqma35)
#@SOC: i.MX35
#@DESCRIPTION: Machine configuration for TQ System i.MX35 Embedded module (tqma35)

include conf/machine/include/soc-family.inc
include conf/machine/include/imx-base.inc
include conf/machine/include/tune-arm1136jf-s.inc

SOC_FAMILY = "mx3:mx35"

PREFERRED_VERSION_udev_mx3 = "172"

PREFERRED_PROVIDER_virtual/kernel_mx3 = "linux-tqs"
PREFERRED_PROVIDER_u-boot = "u-boot-tqs"

UBOOT_MACHINE = "TQMa35_config"
UBOOT_SUFFIX = "bin"
UBOOT_MAKE_TARGET = "u-boot.${UBOOT_SUFFIX}"

SERIAL_CONSOLE = "115200 ttymxc0"

MACHINE_FEATURES += "ext2 ext3 screen"
```

[conf/machine/include/imx-base.inc] (from the *meta-fsl-arm* layer)

Important variables:

- IMAGE_FSTYPES: Located on imx-base.inc. Defines the type of outputs for the Root Filesystem. Default is: `"tar.bz2 ext3 sdcard"`. On the TQMa35 we have to evaluate: `"ubi jffs2 tar.bz2"`.
- UBOOT_ENTRYPOINT_*: Located on imx-base.inc. Defines where the Kernel is loaded by U-Boot.

- SOC_FAMILY: Defines machine's family. Only recipes with the same SOC_FAMILY (defined with the recipe's variable COMPATIBLE_MACHINE) are taken into account when baking for a particular machine.
- UBOOT_MACHINE: Define the U-Boot configuration file
- PREFERRED_PROVIDER_*: Defines which package name (PN) of the recipe you want to give precedence.
  - PREFERRED_PROVIDER_virtual/kernel_mx3. Default located on imx-base.inc. Defines the Freescale community supported Linux kernel (*linux-fslc*). On the TQMa35 we force to use the TQ Systems supported Linux kernel (*linux-tqs*).
  - PREFERRED_PROVIDER_u-boot. Default located on fsl-default-providers.inc. Defines the Freescale community supported U-Boot (*u-boot-fslc*). On the TQMa35 we force to use the TQ Systems supported U-Boot (*u-boot-tqs*).
- PREFERRED_VERSION_*: Defines which package version (PV) of the recipe you want to give precedence.
  - PREFERRED_VERSION_udev_mx3: Default is nowhere located on and is always (and implicitly) defined by the head udev recipe version in the Poky distribution (see *poky/meta* layer). On the TQMa35 we force to use the older but with the TQ Systems supported Linux kernel more compatible version 172.

4.2.6 Machine configuration file: `meta-tqs-arm/conf/qemuarmv6.conf`

```
#@TYPE: Machine
#@NAME: arm_versatile_1136jfs
#@DESCRIPTION: arm_versatile_1136jfs

require conf/machine/include/qemu.inc
require conf/machine/include/tune-arm1136jf-s.inc

PREFERRED_VERSION_udev = "172"

PREFERRED_PROVIDER_virtual/kernel = "linux-tqs"

KERNEL_IMAGETYPE = "zImage"

SERIAL_CONSOLE = "115200 ttyAMA0"
```

[conf/machine/include/qemu.inc] (from the *poky/meta* layer)

Important variables:

- IMAGE_FSTYPES: Located on qemu.inc. Defines the type of outputs for the Root Filesystem. Default is: `"tar.bz2 ext3"`. *Ext3* can than used by **runqemu** command.
- EXTRA_IMAGEDEPENDS: Located on qemu.inc. Defines the extra dependent tasks to host's native Qemu tools. Default is: `"qemu-native qemu-helper-native"`
- KERNEL_IMAGETYPE: Define the Linux kernel image binary format. *zImage* can than used by **runqemu** command.
- SERIAL_CONSOLE: Define the serial console (*baud rate* and *device name*) for getty.

- PREFERRED_PROVIDER_virtual/kernel. Default located on qemu.inc. Defines the Freescale community supported Linux kernel (*linux-yocto*). On the QemuARMv6 we force to use the TQ Systems supported Linux kernel (*linux-tqs*).
- PREFERRED_VERSION_udev: Default is nowhere located on and is always (and implicitly) defined by the head udev recipe version in the Poky distribution (see *poky/meta* layer). On the TQMa35 we force to use the older but with the TQ Systems supported Linux kernel more compatible version 172.

CHAPTER

FIVE

CREATING A NEW LAYER

It is suggested to create a layer when creating or modifying any metadata file (recipe, configuration file or class). The main reason is simple: modularity. In the other hand, make sure your new metadata has not already be implemented (layer, recipe or machine), so before proceeding check this link.

- To have access to Yocto scripts, setup the enviroment from the BASE folder:

  ```
  tqs-community-bsp $ . setup-environment build
  ```

- Move to the place you want to create your layer and choose a name (e.g. `meta-tqs-custom`):

  ```
  sources $ yocto-layer create meta-tqs-custom
  # Answer the questions. Make sure the priority is set correctly
  # (higher numbers, higher priorities). Set the priority equal to
  # the lowest already present, except when you have introduce a
  # new recipe with the same name as other and want to shadow the
  # original one.
  ```

- Add any metadata content. Suggestion: Version the layer with Git and upload your local git repo to a server.
- Edit and add the layer to the `build/conf/bblayers.conf` file.

  Todo

  The issue *Add the Layer* to the `build/conf/bblayers.conf` file (modify this file) and *versioning new custom layer with Git* should describe more precise in focus of the underlayed `repo` tool. **Maybe that this part of the tutorial is not proper!**

- To verify that your layer is *seen* by BitBake, run the following command under the BUILD folder:

  ```
  build $ bitbake-layers show-layers
  ```

CHAPTER

SIX

PATCHING THE LINUX KERNEL

The Linux Kernel is just another recipe for Yocto, so learning to patch it you learn to patch any other package. In the other hand, Yocto **should not** be used for package development, but in those rare cases follow the steps below. It is assumed that you have already build the package you want to patch.

- Create the patch or patches. In this example we are patching the Linux kernel for tqma35 machine; in other words, the value of MACHINE on the `build/conf/local.conf` is `MACHINE ??= 'tqma35_'`. In case you already have the patches, make sure these can be nicely applied with the commands `git apply --check` <PATCH_NAME>, and jump this step.

```
build $ cd tmp/work/tqma35-poky-linux-gnueabi/\
linux-tqs/2.6.34.14+gitAUTOINC+6b4ea726b39f32041ac4d2dd03cf056c57b638ac-r32.1/git
build $ # Edit any files you want to change
build $ git add <modified file 1> <modified file 2> ..  # Select the files you
                                                        # want to commit
build $ git commit -s -m '<your commit's title>'        # Create the commit
build $ git format-patch -1                             # Create the patch

# e.g. 0001-calibrate-Add-printk-example.patch
```

- Create a new layer (see section *Creating a new Layer*)
- On the new layer (e.g. `meta-tqs-custom`), create the corresponding subfolders and the `.bbappend` file:

```
sources $ mkdir -p meta-tqs-custom/recipes-kernel/linux/linux-tqs-2.6.34.14
sources $ cat > meta-tqs-custom/recipes-kernel/linux/linux-tqs_git.bbappend
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}-${PV}:"
SRC_URI += "file://0001-calibrate-Add-printk-example.patch"
PRINC := "${@int(PRINC) + 1}"
^d
```

- Move the patch to the new layer

```
sources $ mv ../build/tmp/work/tqma35-poky-linux-gnueabi/linux-tqs/\
2.6.34.14+gitAUTOINC+6b4ea726b39f32041ac4d2dd03cf056c57b638ac-r32.1/\
git/0001-calibrate-Add-printk-example.patch \
  meta-tqs-custom/recipes-kernel/linux/linux-tqs-2.6.34.14
```

- Setup the enviroment and clean previous package's build data (sstate)

```
tqs-community-bsp $ . setup-environment build
build $ bitbake -c cleansstate linux-tqs
```

- Compile and Deploy

```
build $ bitbake -f -c compile linux-tqs
build $ bitbake -c deploy linux-tqs
```

- Insert the SD into your host and copy the `uImage` into the first partition. Do not forget to unmount the partition before removing the card!

> **Warning:** The issue *SD Card preperation* with new Linux kernel needs to be evaluated! Do not yet apply this description!

```
build $ sudo cp tmp/deploy/images/uImage /media/boot
```

- Insert the SD into your board and test your change.

CHAPTER

SEVEN

BUILDING THE KERNEL MANUALLY

- To setup the Yocto environment, from the BASE folder run:

```
tqs-community-bsp $ . setup-environment build
```

- Build the toolchain:

```
build $ bitbake meta-toolchain
# Other toolchains:
#   Qt Embedded toolchain build: bitbake meta-toolchain-qte
#   Qt X11 toolchain build: bitbake meta-toolchain-qt
```

- Install it on your PC:

```
build $ sudo sh \
  tmp/deploy/sdk/poky-eglibc-x86_64-arm-toolchain-<version>.sh
```

- Setup the toolchain environment:

```
build $ source \
  /opt/poky/<version>/environment-setup-armv6-vfp-poky-linux-gnueabi
```

- Get the Linux Kernel's source code:

```
$ git clone git://github.com/lipro/linux-tqs.git linux-tqs
$ cd linux-tqs
```

- Create a local branch:

```
linux-tqs $ BRANCH=tqma35 # Change to any branch you want,
                          # Use 'git branch -a' to list all
linux-tqs $ git checkout -b ${BRANCH} origin/${BRANCH}
```

- Export ARCH and CROSS_COMPILE:

```
linux-tqs $ export ARCH=arm
linux-tqs $ export CROSS_COMPILE=arm-poky-linux-gnueabi-
linux-tqs $ unset LDFLAGS
```

- Choose configuration and compile:

```
linux-tqs $ make tqma35_defconfig
linux-tqs $ make uImage
```

- To Test your changes, insert the SD into your host and copy the `uImage` into the first partition:

> **Warning:** The issue *SD Card preperation* with new Linux kernel needs to be evaluated! Do not yet apply this description!

```
linux-tqs $ sudo cp arch/arm/boot/uImage /media/boot
```

- If case you want your changes to be reflected on your Yocto Framework, create the patches following the section *Patching the Linux Kernel*.

CHAPTER

EIGHT

CONTRIBUTING TO THE TQ SYSTEMS YOCTO PROJECT

The Yocto Project is open-source, so anyone can contribute. No matter what your contribution is (bug fixing or new metadata), contributions are sent through patches to a community list. Many eyes will look into your patch and at some point it is either rejected or accepted. Follow these steps to contribute:

- Make sure you have previously configured your personal info:

```
$ git config --global user.name "Your Name Here"
$ git config --global user.email "your_email@example.com"
```

- Download `master` branches:

```
tqs-community-bsp $  repo init \
  -u git://github.com/lipro/tqs-community-bsp-platform \
  -b master
```

- Update:

```
tqs-community-bsp $ repo sync
```

- Create local branches so your work is *not* done on master:

```
tqs-community-bsp $ repo start <branch name> --all
```

Where <`branch name`> is any name you want to give to your local branch (e.g. `fix_uboot_recipe`, `new_gstreamer_recipe`, etc.).

- Make your changes in any TQ Systems related folder (e.g. sources/meta-tqs-arm). In case you modified a recipe (`.bb`) or include (`.inc`) file, do not forget to *bump* (increase the value by one) either the PR or `INC_PR` value.
- Commit your changes using `git`. In this example we assume your change is on `meta-tqs-arm` folder:

```
sources/meta-tqs-arm $ git add <file 1> <file 2>
sources/meta-tqs-arm $ git commit
```

On the commit's log, the title must start with the filename change or introduced, then a brief description of the patch's goal, following with a long description. Make sure you follow the standards (type **git log --pretty=oneline** to see previous commits).

- Create a patch:

```
sources/meta-tqs-arm $ git format-patch -s \
  --subject-prefix='meta-tqs-arm][PATCH' -1
```

Where the last parameter (`-1`) indicate to patch last commit. In case you want to create patches for older commits, just indicate the correct index. If your patch is done in other folder, just make sure you change the `--subject-prefix` value.

- Send your patch or patches with:

Where <`patch`> is the file created by **git format-patch**.

- Keep track of patch's responses on the mailing list. In case you need to rework your patch, repeat the steps but this time the patch's subject changes to `--subject-prefix='meta-tqs-*][PATCH v2'`.
- Once your patch has been approved, you can delete your working branches:

  ```
  tqs-community-bsp $ repo abandon <branch name>
  ```