OpenEmbedded and Yocto Project best practices

Alexandre Belloni
alexandre.belloni@bootlin.com
Alexandre Belloni

- Embedded Linux engineer at Bootlin
  - Embedded Linux expertise
  - Development, consulting and training
  - Bootloader, Linux kernel, Yocto Project, Buildroot
  - Complete Linux BSP development
  - Hardware support in bootloader/Linux
  - Strong open-source focus: upstreaming and contributions
  - Freely available training materials

- Open-source contributor
  - Maintainer for the Linux kernel RTC subsystem
  - Co-Maintainer of kernel support for Microchip (ARM and MIPS) processors
Distributions
Poky

- Yocto Project is an entity, not something you can use.
- Poky is the reference distribution, the code that is downloaded and used.
- As a reference distribution, it is not tailored to your system (e.g. it always includes opengl)
- It can generate demo images but is not meant to be used as-is on production systems.
- The included features are not stable (e.g. it switched from xorg to wayland)
- Poky bundles Openembedded-core, bitbake and two very small layers:
  - meta-yocto-bsp is a **BSP** layer for reference boards from the Yocto Project members
  - meta-poky is a **distro** layer with four distributions: poky, poky-tiny, poky-bleeding, poky-altcfg
For your project, not using Poky has some advantages:

- when reporting bugs, it is necessary to reproduce with a **nodistro** build
- it is easier to start from **nodistro** and create a distribution than tuning a distribution including **poky.conf**
- it is easier to work with the oe-core repository when sending patches upstream
- Confidentiality, Poky defines **PREMIRRORS** that point to
  
  http://downloads.yoctoproject.org/mirror/sources/, it will leak the name of everything that is fetched using version control.

- The main drawback is having to match the oe-core and bitbake branches manually.
Creating your own distribution

- Not that difficult, simply have `conf/distro/<distro_name>.conf`
- Used to define the distribution wide policies:
  - Toolchain (including libc) selection
  - init selection
  - `DISTRO_FEATURES`
  - `PREFERRED_PROVIDERS`
  - `PACKAGE_CLASSES`
  - QA checks with `WARN_QA` and `ERROR_QA`
local.conf
local.conf

- local.conf is really for local configuration (CPU number, disk space).
- Avoid the numerous tutorials saying otherwise
- The main reason is distribution of the changes and reproducibility of the build.
- Also huge drawback, a change in local.conf makes bitbake parse all the recipes again.
- It is fine to carry changes in local.conf for development/testing.
site.conf is for site wide configuration (proxies, mirrors, shared sstate-cache location).

Unfortunately, it suffers from the same local.conf distribution drawback.
The most abused variable in `local.conf` is `IMAGE_INSTALL_append` (seen in tutorials from SoM vendors).

This is not even easy for beginners due to parse order.

The solution is simply to create your own image recipe as soon as the `core-image-*.bb` recipes are not enough anymore.
All the machine related variables should go in the machine configuration:

- `PREFERRED_PROVIDER_virtual/kernel`
- `PREFERRED_PROVIDER_virtual/bootloader`
- `PREFERRED_VERSION_linux-*`
- `IMAGE_FSTYPES`
- In a few cases, `IMAGE_INSTALL_append`, for example, to actually install the kernel in the root filesystem.
The other variables should go in the distro configuration:

- PREFERRED_PROVIDER_*
- PREFERRED_VERSION_*
- PACKAGECONFIG_pn-*
- INCOMPATIBLE_LICENSE
- LICENSE_FLAGS_WHITELIST
Release management
There are multiple tasks that OE/bitbake based projects let you do on your own to ensure build reproducibility:

- Code distribution and project setup.
- Release tagging

A separate tool is needed for that, usual solutions are:

- combo-layer, as done by Poky:
  https://wiki.yoctoproject.org/wiki/Combo-layer
- git submodules + setup script. Great example in YOE:
  https://github.com/YoeDistro/yoe-distro
- repo and templateconf or setup script
- kas
repo is used in Android to distribute its source code, which is split into many git repositories. It’s a wrapper to handle several git repositories at once.

The repo configuration is stored in manifest file, usually available in its own git repository.

It could also be in a specific branch of your custom layer.

It only handles fetching code, handling local.conf and bblayers.conf is done separately.
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
    <default sync-j="4" revision="dunfell"/>

    <remote fetch="https://github.com/openembedded" name="oe"/>
    <remote fetch="https://github.com/Freescale" name="freescale"/>
    <remote fetch="ssh://git@server.com" name="private"/>

    <project remote="freescale" name="meta-freescale" path="sources/meta-freescale"/>
    <project remote="oe" name="openembedded-core" path="sources/openembedded-core"/>
    <project remote="oe" name="bitbake" path="sources/openembedded-core/bitbake"
             revision="1.46" />
    <project remote="oe" name="meta-openembedded" path="sources/meta-openembedded"/>

    <project remote="private" name="meta-custom" path="sources/meta-custom">
        <copyfile dest="setup-environment" src="buildconf/setup-environment"/>
    </project>
</manifest>
To tag a release, a few steps have to be taken:

- Optionally tag the custom layers
- For each project entry in the manifest, set the revision parameter to either a tag or a commit hash.
- Commit and tag this version of the manifest.
Specific tool developed by Siemens for OpenEmbedded: https://github.com/siemens/kas

- Will fetch layers and build the image in a single command
- Uses a single JSON or YAML configuration file part of the custom layer
- Can generate and run inside a Docker container
- Can setup local.conf and bblayers.conf
kas configuration

header:
  version: 8
machine: mymachine
distro: mydistro
target:
  - myimage

repos:
  meta-custom:

  bitbake:
    url: "https://git.openembedded.org/bitbake"
    refspec: "1.46"

  openembedded-core:
    url: "https://git.openembedded.org/openembedded-core"
    refspec: dunfell
layers:
  meta:
kas configuration

meta-freescale:
  url: "https://github.com/Freescale/meta-freescale"
  refspec: dunfell

meta-openembedded:
  url: http://git.openembedded.org/meta-openembedded
  refspec: dunfell
  layers:
    meta-oe:
    meta-python:
    meta-networking:
Network access

Another task when creating a release is to ensure all the code is available internally, either on the local build machine or on local mirrors.

- Ensure there is no `SRCREV = ${AUTOREV}` in any recipe.
- Set `BBGENERATE_MIRROR_TARBALLS = "1"` to generate tarballs of the git repositories in `DL_DIR`.
- Fetch all the source (e.g. using `bitbake -c fetchall <target>`).
- Archive `DL_DIR`, make the tarballs available internally.
- Optionally build once with `BB_NO_NETWORK = "1"` to check for missing tarballs or remaining `AUTOREV`.
- Point bitbake to your internal mirrors, using `PREMIRRORS` or `INHERIT += "own-mirrors"` with `SOURCE_MIRROR_URL`.
- Build the release, from scratch using `BB_FETCH_PREMIRRORONLY = "1"`. 
Build optimization
Sharing the sstate-cache

It is possible to share the shared state cache across multiple build machines:

▶ Set up CI or nightly builds.
▶ Use the `DL_DIR` to populate the `PREMIRRORS`.
▶ Share the sstate-cache (`SSTATE_DIR`) over NFS or HTTP.
▶ Setup `SSTATE_MIRRORS` to point to that share

This works well if all the hosts are similar as this influence checksums. Containers will help.
Cleaning the sstate-cache

The sstate-cache is growing over time. It is possible to clean old data with:

```
$ ./scripts/sstate-cache-management.sh --remove-duplicated -d \
   --cache-dir=<SSTATE_DIR>
```
License compliance
OpenEmbedded will generate a manifest of all the licenses of the software present on the target image in LICENSE_DIRECTORY/IMAGE_NAME/license.manifest

PACKAGE NAME: busybox
PACKAGE VERSION: 1.31.1
RECIPE NAME: busybox
LICENSE: GPLv2 & bzip2-1.0.6

PACKAGE NAME: dropbear
PACKAGE VERSION: 2019.78
RECIPE NAME: dropbear
LICENSE: MIT & BSD-3-Clause & BSD-2-Clause & PD
To include the license text in the root filesystem either:

- Use `COPY_LIC_DIRS = "1"` and `COPY_LIC_MANIFEST = "1"`
- or use `LICENSE_CREATE_PACKAGE = "1"` to generate packages including the license and install the required license packages.
OpenEmbedded provides the archiver class to generate tarballs of the source code:

- Use `INHERIT += "archiver"`
- Set the `ARCHIVER_MODE` variable, the default is to provide patched sources. To provide configured sources:

```
ARCHIVER_MODE[src] = "configured"
```
Questions? Suggestions? Comments?

Alexandre Belloni
alexandre.belloni@bootlin.com

Slides under CC-BY-SA 3.0