Pros and cons of various layer management tools

Yocto Summit 2020

Tomasz Żyjewski

3MDEB

Agenda

- Introduction
- Yocto layered model
- Presentation goal
- Description of example project
- Layer management - git submodules
- Layer management - repo
- Layer management - combo-layer
- Layer management - kas
- Summary and conclusions

/usr/bin/whoami

tomasz.zyjewski@3mdeb.com

- Embedded Systems Engineer
- over 1 year Embedded Linux / Yocto experience
- aspiring Yocto contributor

Company profile

3mdeb is a firmware and embedded systems development company founded by Piotr Król and headquartered in Gdańsk, Poland. We combine the collaborative creativity of the open-source community with the reliable strength of an enterprise-grade solutions provider.

Yocto layered model

The Yocto Project

- Initiated in 2010 by the Linux Foundation
- open source project used to develop custom Linux-based systems
- one of the most popular, the others are Buildroot or OpenWrt

Starting with the Yocto Project

- overwhelming at first
- lot of tools, informations, assumptions, variables

Yocto layered model

Starting a new project can be like

- prepare description of requirements
- transforming it into list of tools and packages
- in case of Yocto, every package is delivered by corresponding recipe

At this moment we can decide to provide all required recipes from our main repository or to search and reuse existing recipes from existing layers.

meta-aarch64	AArch64 (64-bit ARM) architecture support	Machine (BSP)	git://git.linaro.org/openembedded/meta-linaro.git
meta-acme	Acme Systems Yocto meta layer	Machine (BSP)	https://github.com/myfreescalewebpage/meta-acme
meta-allwinner-hx	Meta layer for all allwinner H2/H3/H5 boards	Machine (BSP)	https://gitlab.com/dlmtass/meta-allwinner-hx
meta-altera	Altera SoC BSP layer	Machine (BSP)	https://github.com/kraj/meta-altera
meta-amarula-engicam	Yocto Meta BSP layer for Engicam boards	Machine (BSP)	https://github.com/amarula/meta-amarula-engicam.git
meta-amd	AMD board support common layer (official)	Machine (BSP)	git://git.yoctoproject.org/meta-amd
meta-amdfalconx86	AMD Falcon board support layer (official)	Machine (BSP)	git://git.yoctoproject.org/meta-amd
meta-arduino	Board Support for the Arduino Yún	Machine (BSP)	https://gitlab.com/toganlabs/meta-arduino

Yocto layered model

So what are the layers in the Yocto Project?

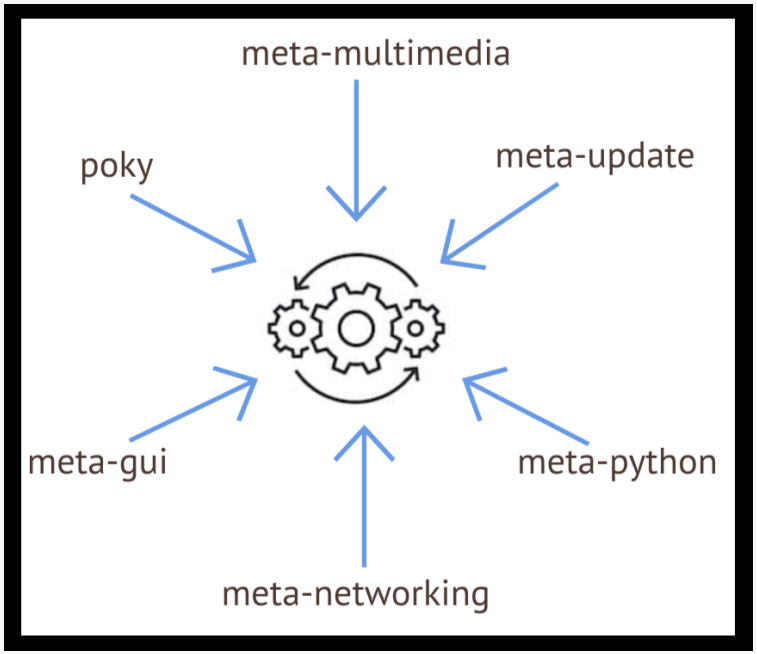
- one of the many features
- group related functionalities into separate bundles
- can be added to any project at any time (but some layers have dependencies)
- reduces the complexity and redundancy of a project

From the files perspective, layers are some repositories which contains meta-data like recipes or configuration files.

Layers can and should be used to logically break down information about our build.

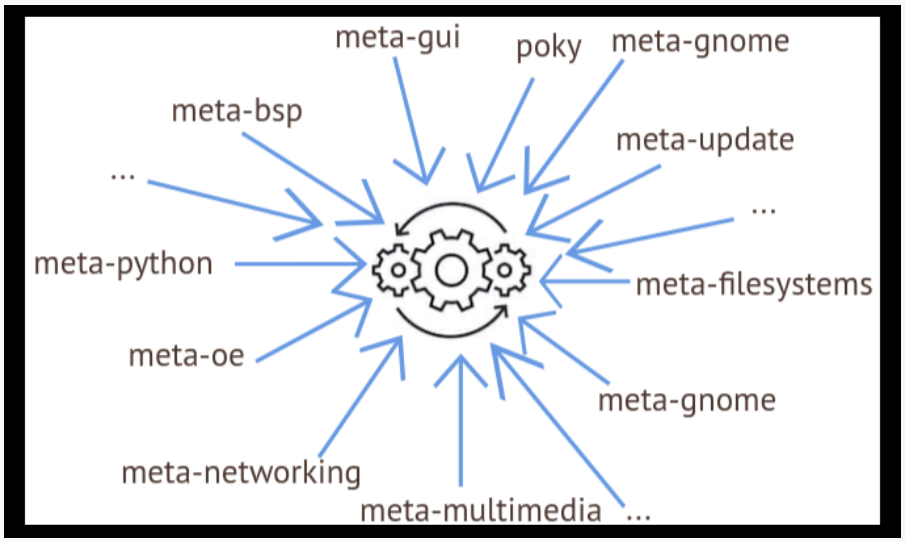
Yocto layered model

- A good prepared layer is a layer that contains the smallest set of metadata.
- This also means that the smallest project can consists of several layers.



Yocto layered model

- The more complex project, the more layers we need to use



- What should we do now? How to start?

Presentation goal

Fortunately, the Yocto layered model also allows to create user friendly tools for managing layers. In this presentation I would like to present four such tools

- use of `git submodules`
- use of the Google `repo` tool
- `combo-layer` script from poky
- `kas` tool from Siemens repository

The main goal of this presentation

- a list of advantages and disadvantages of given solutions
- give a chance to get to know several solutions to choose the most appropriate

Description of example project

The best lesson is to practice, therefore

- the four mentioned tools will be tested on a small project
- the project will be large enough to show features of given solution

Short high-end description of the project

- image for RPi0 with X server, allowing to run chromium browser

- changes in `local.conf` to set machine, distro and some variables

Description of example project

- used layers
  - `meta-openembedded` and `poky` - basic layers used in builds, each of them contains smaller layers and we will use some of them (e.g. meta-networking)
  - `meta-swupdate` - provide possibility to create files to update system
  - `meta-webkit` - some useful tools including benchmark test for browser
  - `meta-raspberrypi` - RPi layer, provides, among others, machine config for the used hardware
  - `meta-presentation` - custom layer that allows you to make changes to existing recipes (e.g. dnsmasq) to be applied to the target image
- the last task will be to patch one of the used layers

Description of example project

Presentation goal summary

- the key goal is to compare various of tools on the same project
- listed project requirements cover most of the popular tasks that developer need to face when preparing layers
- order of tested tools is random, each of them will end with a pros and cons of given solution
- every custom configuration file or script needs to be placed in out custom layer which is `meta-presentation`

Layer management - git submodules

- `git submodule` - one of the git's internal tools
- description from https://git-scm.com/

```
It often happens that while working on one project, you need to use another
project from within it. Perhaps it’s a library that a third party developed or
that you’re developing separately and using in multiple parent projects. A
common issue arises in these scenarios: you want to be able to treat the two
projects as separate yet still be able to use one from within the other.
```

Doesn't this description fit perfectly with what we deal with in YP with layers?

- layers should be separated
- it should be easy to use more than one layer at a time

Layer management - git submodules

- we should add other layers as submodules to our custom layer
- adding submodules (meta layers) is done via the command `git submodule <URL>`, to add `poky` as submodule type

```
$ git submodule add https://git.yoctoproject.org/git/poky
```

- after adding all of them, commit changes

```
$ git commit -s -m "add needed meta-layers as submodules"
[git_submodules 0615242e8318] add needed meta-layers as submodules
  6 files changed, 20 insertions (+)
  create mode 100644 .gitmodules
  create mode 160000 meta-openembedded
  create mode 160000 meta-raspberrypi
  create mode 160000 meta-swupdate
  create mode 160000 meta-webkit
  create mode 160000 poky
```

- note special `160000` mode for files

Layer management - git submodules

- by default, added submodules will be checked out on master
- to set correct refspec, `cd` to any one of them, make `git checkout` and commit changes

```
$ cd poky
$ git checkout 88c6be81a5fbed098999fbef5576c5e0bb90cc21
$ cd ..
$ git add .
$ git commit -s -m "set correct refspec"
[git_submodules ca0f9be2a11d] set correct refspec
 1 file changed, 1 insertions(+), 1 deletions(-)
$ git show
commit ca0f9be2a11d5d6de351dfef6b69884617041542 (HEAD -> git_submodules)
Author: Tomasz Żyjewski <tomasz.zyjewski@3mdeb.com>
Date:   Mon Oct 26 09:36:21 2020 +0100

    set correct refspec

    Signed-off-by: Tomasz Żyjewski <tomasz.zyjewski@3mdeb.com>
diff --git a/poky b/poky
index 4e4a302e37ac..88c6be81a5fb 160000
--- a/poky
+++ b/poky
@@ -1 +1 @@
-Subproject commit 4e4a302e37ac06543e9983773cdb4caf7728330d
+Subproject commit 88c6be81a5fbed098999fbef5576c5e0bb90cc21
```

Layer management - git submodules

- apply custom patch on one of the used layers (`poky`)
- needs to be done manually, `git submodules` do not support any kind of patching layers from local files

```
$ cp patches/0001-add-removed-classes.patch poky/
$ cd poky/
$ git apply 0001-add-removed-classes.patch
$ git st
HEAD detached at 88c6be81a5fb
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   meta/classes/image_types_wic.bbclass
        modified:   scripts/lib/wic/plugins/imager/direct.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        meta/classes/bluetooth.bbclass
        meta/classes/gnome.bbclass
```

Layer management - git submodules

- modification of `bblayers.conf` and `local.conf` files
  - `git submodules` does not allow to make those changes automatically
- to use custom configuration files we need to
  - run `oe-init-build-env` first to create build environment and copy our custom configuration files **OR**
  - run `oe-init-build-env` with special `TEMPLATECONF` variable pointed to our custom configuration files
- either way, this needs to be done every time someone start working with project, possibility to make errors

Layer management - git submodules

Summary of `git submodules`

Pros

- does not require additional dependencies, most developers use git, right?
- easy to use

Cons

- limited to downloading selected layers and revision settings
- preparation of the environment requires manual work or some trickery related to writing scripts

Layer management - repo

- `repo` - tool built on top of Git
- helps to manage many repositories, set correct revisions
- `repo` command is a Python script, the easiest way to use it is to download and add it to PATH variable

```
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 39134  100 39134    0     0   150k      0 --:--:-- --:--:-- --:--:--  151k
$ chmod a+x ~/bin/repo
$ repo --help
usage: repo COMMAND [ARGS]

repo is not yet installed.  Use "repo init" to install it here.

The most commonly used repo commands are:

  init      Install repo in the current working directory
  help      Display detailed help on a command

For access to the full online help, install repo ("repo init").
```

Layer management - repo

- `repo` works with manifest files
- our custom layer should contain `default.xml` file with list of needed meta layers
- it could look as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <default remote="poky" revision="default"/>

  <remote name="poky" fetch="git://git.yoctoproject.org" />
  <remote name="rpi" fetch="git://git.yoctoproject.org" />
  <remote name="pres" fetch="ssh://github.com/Tomasz" />

  <project name="poky"
  revision="88c6be81a5fbed098999fbef5576c5e0bb90cc21"
  remote="poky"/>
  <project name="meta-raspberrypi"
  revision="9d0935c9bb1309431e62b8f8341eb503653e5ff5"
  remote="rpi"/>
  <project name="meta-presentation"
  revision="6cb191164259ee179163bfccb13a1dbac3c136ba"
  remote="pres" />
</manifest>
```

- ssh may be needed when fetching from private repositories

Layer management - repo

- to initialize new project, create new directory and call `repo init`

```
$ mkdir yocto_project && cd yocto_project
$ repo init -u <URL> -b <branch>
```

- after that run `repo sync`, all layers will be cloned and checked out to the correct revision
- custom patch again need to be applied manually
- the same applies to modifying the configuration files, the only option remains is to copy them from the custom layer or a special call to `oe-init-build-env`
- this can cause some errors and need to be done every time

Layer management - repo

Summary of `git submodules`

Pros

- has many built-in options that allow you to control what you want to clone, set the revision etc.
- is based on a manifest file downloaded from anywhere (it can be a custom meta-layer), which briefly and accurately describes the elements (layers) included in the project
- popular solution, often used, which means that it is also well tested

Cons

- requires some preparation before use, for example installing a script to PATH
- as with `git submodules`, modifying anything other than the list of layers in the appropriate revision is impossible, requires writing additional scripts

Layer management - combo-layer

- `combo-layer` - command-line utility
- creates one mega combo layer that is combination of several layers
- script available from `poky`

```
$ ./combo-layer -h
Usage: combo-layer [options] action
Create and update a combination layer repository from multiple component repositories.

Action:
  init                initialise the combo layer repo
  update [components] get patches from component repos and apply them to the combo repo
  pull [components]   just pull component repos only
  splitpatch [commit] generate commit patch and split per component, default commit is HEAD
Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -c CONFFILE, --conf=CONFFILE
                        specify the config file (conf/combo-layer.conf is the
                        default).
  -i, --interactive     interactive mode, user can edit the patch list and
                        patches
  -D, --debug           output debug information
  -n, --no-pull         skip pulling component repos during update
  --hard-reset          instead of pull do fetch and hard-reset in component
                        repos
  -H, --history         import full history of components during init
```

Layer management - combo-layer

- to use, add `combo-layer` script and `combo-layer.conf` file to custom layer
- entry for each layer can looks as follow

```
[meta-webkit]
src_uri = https://github.com/Igalia/meta-webkit.git
local_repo_dir = /home/tzyjewski/yocto_project/sources/meta-webkit
dest_dir = meta-webkit
branch = master
last_revision = 7bba8b0f806b10912f5d427d19867d017fc3aa34
file_exclude = recipes-browser recipes-devtools recipes-extended recipes-flatpak recipes-graphics
```

- mandatory variables
  - src_uri
  - local_repo_dir
  - last_revision
- optional variables
  - branch
  - file_exclude
  - more...

Layer management - combo-layer

- to prepare mega layer, copy repository with configuration file and run `init`

```
$ ./meta-presentation/scripts/combo-layer init -c meta-presentation/conf/combo-layer.conf
```

- as mentioned earlier, `combo-layer` not only clone given layers, it adds them to one repository, now we can commit all of them in one repo
- no options to automatically apply local patches to cloned layers
- `combo-layer` similar to the previous tools, limited to setting layers, unable to customize `local.conf` or `bblayers.conf`

Layer management - combo-layer

Summary of `combo-layer`

Pros

- transparent configuration file for selecting layers and their contents
- powerful in case of customizing used layers
- self-contained repository

Cons

- self-contained repository
- reviews might be time consuming, if there are many changes in many layers

Layer management - kas

- kas - provides an easy mechanism to setup bitbake based projects
- set layers, create default settings, launch build environment, initiate bitbake build process
- download kas-docker script to PATH (need docker installed)

```
$ wget -O ~/bin/kas-docker https://raw.githubusercontent.com/siemens/kas/1.0/kas-docker
$ chmod +x ~/bin/kas-docker
```

- kas-docker script

```
$ kas-docker
Usage: /home/tzyjewski/bin/kas-docker [OPTIONS] { build | shell } [KASOPTIONS] KASFILE
       /home/tzyjewski/bin/kas-docker [OPTIONS] clean
Positional arguments:
build                   Check out repositories and build target.
shell                   Run a shell in the build environment.
clean                   Clean build artifacts, keep downloads.
Optional arguments:
--isar                  Use kas-isar container to build Isar image.
--docker-args           Additional arguments to pass to docker forrunning the build.
-v                      Print operations.
--ssh-dir               Directory containing SSH configurations,
                        avoid /home/tzyjewski/.ssh unless you fully trust the container.
--no-proxy-from-env     Do not inherit proxy settings from environment.
```

Layer management - kas

- `kas-docker` runs with kas configuration files, they describe used meta layers and changes to the configuration files
- entry for a given layer

```
meta-raspberrypi:
    url: https://git.yoctoproject.org/git/meta-raspberrypi
    refspec: 9d0935c9bb1309431e62b8f8341eb503653e5ff5
```

- kas configuration files can be stored in custom layer, to start new project we usually create new directory, clone custom layer and start `kas-docker` with command

```
kas-docker build meta-presentation/kas.yml
```

- all layers will be cloned and checked out inside `yocto_project` directory

Layer management - kas

- kas configuration files allows to create default build settings
- it is possible via `bblayers_conf_header:` and `local_conf_header:` sections, first will modify top of `bblayers.conf` file and the second `local.conf` file
- `BBLAYERS` will be generated based of the layer entries from kas configuration file
- distro and machine can be set via `distro` and `machine` variables in kas file

- generated `bblayers.conf`

```
$ cat build/conf/bblayers.conf
# standard
POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
LICENSE_FLAGS_WHITELIST = "commercial"
BBLAYERS ?= " \
/repo/ \
/work/meta-openembedded/meta-multimedia \
/work/meta-openembedded/meta-networking \
...
```

- generated `local.conf`

```
local_conf_header:
standard: |
CONF_VERSION = "1"
PACKAGE_CLASSES = "package_rpm"
SDKMACHINE = "x86_64"
debug-tweaks: |
EXTRA_IMAGE_FEATURES = "debug-tweaks"
MACHINE ??= "raspberrypi0-wifi"
DISTRO ??= "pres-x11"
```

Layer management - kas

- kas configuration file also allows to patch cloned meta layers
- responsible fragment of the configuration file

```
poky:
  url: https://git.yoctoproject.org/git/poky
  refspec: 88c6be81a5fbed098999fbef5576c5e0bb90cc21
  layers:
    meta:
    meta-poky:
    meta-yocto-bsp:
  patches:
    removed-classes:
      repo: meta-pres
      path: patches/0001-add-removed-classes.patch
```

Layer management - kas

Summary of `kas`

Pros

- allows a large customization of the build environment
- supports bundling the build configuration with a layer
- actually the only tool that properly prepares `local.conf` and `bblayers.conf` files
- prepares build environments

Cons

- when using the `kas-docker` script, installation of the docker is required
- there are problems with file ownership set to 1000, so sometimes the image recipe needs to be cleaned and rebuilt

Summary and conclusions

- every tool was presented with pros and cons
- which one to use? you decide!
- ... but first, check all of them!
- personnaly?
  - `kas`
  - `repo`
  - `git submodules`
  - `combo-layer`

Thank you

Q&A