Activity Six

Licensing 2.0

Beth Flanagan, Paul Barker

Who Are We?

Togán Labs Ltd

- Ireland based Embedded Linux Consultancy
- Developers of Oryx Linux
- OpenChain Partner, strong focus on license compliance

Beta Five Ltd

- Nottingham, UK based
- Open Source Consultancy
- Linux-based projects from Embedded to Cloud

Contact details

- For any follow up questions or enquiries

- Beth Flanagan, Togán Labs Ltd
  - pidge@toganlabs.com

- Paul Barker, Beta Five Ltd
  - paul@betafive.co.uk

Overview

- Mirror creation
- Providing license manifests & text
- Providing recipes

How downloads work in bitbake

- Sources are fetched into the downloads directory
- Git sources are cloned into bare local repositories
- Other version control systems are handled similarly
- For each successful fetch a ‘.done’ file is created

Why create a mirror?

- Best approach for open source distros & BSPs
- Makes life easy for downstream Yocto users
- Saves you from disappearing sources

Generating mirror tarballs

- Bitbake supports mirrorring git sources as a tarball of the bare repository
  - Again, works similarly for other version control systems

- We can create these tarballs automatically during fetch
  - Set the following variable in local.conf or your distro conf file:
    - `BB_GENERATE_MIRROR_TARBALLS = "1"`

Ensuring downloads is populated

- Need to be careful here, sources may not be re-downloaded if a recipe is built from sstate
  - This applies even if downloads is empty
- Must explicitly run the fetch task for all recipes in our image, SDK or other targets
- Thankfully this can be done via the following methods:
  - 2.5 “sumo” or later: `bitbake <target> --runall=fetch`
  - 2.4 “rocko” or earlier: `bitbake <target> -c fetchall`

Collecting mirror files

- We don’t need the ‘.done’ files in our mirror
- We also don’t need the uncompressed bare git repositories and similar directories for other version control systems
- We use the following magic:

```
      mkdir -p mirror
      for f in `find downloads -maxdepth 1 -type f -not -
name *.done`; do ln -f $f mirror/`basename $f`; done
```

- These hard links save space but are easy to copy

Serving your mirror

- Internally
  - Local directory
  - NFS share

- Publically
  - HTTP server

Using the mirror

- **Local path:**

```
PREMIRRORS_prepend = " \
ftp://.*/.*      file://${TOPDIR}/mirror/ \n \
http://.*/.*     file://${TOPDIR}/mirror/ \n \
https://.*/.*    file://${TOPDIR}/mirror/ \n \
git://.*/.*      file://${TOPDIR}/mirror/ \n"
```

- **Public mirror:**

```
PREMIRRORS_prepend = " \
ftp://.*/.*      https://example.com/mirror/ \n \
http://.*/.*     https://example.com/mirror/ \n \
https://.*/.*    https://example.com/mirror/ \n \
git://.*/.*      https://example.com/mirror/ \n"
```

Testing your mirror

- **Set the following in local.conf:**
  - `BB_FETCH_PREMIRRORONLY = “1”`
- **The build will then use only the configured mirror**

The own-mirrors class

- **Intended for local testing only**
- **You can set the following in local.conf:**

  -
    ```
    INHERIT += "own-mirrors"
    SOURCE_MIRROR_URL =
      "https://example.com/mirror/"
    ```
- **Do not use this in a distro conf as it supports only one SOURCE_MIRROR_URL value**

License manifests

- **Useful to have a simple list of packages installed and their licenses**
- **This is created automatically during an image build**
- **See tmp/deploy/licenses/<image>-<machine>-<timestamp>**
- **For example:**
- `tmp/deploy/licenses/core-image-base-qemux86-20180926120707/`

License manifests (2)

- Files created:
- `package.manifest`
  - Simple list of installed packages
- `license.manifest`
  - Packages, versions, recipe names and licenses
- `image_license.manifest`
  - As above for dependencies not directly installed in the image (e.g. bootloader)

License Text

- For each recipe you will also find a directory in tmp/deploy/licenses.
- This contains license texts
- Also contains a `recipeinfo` file summarising the license and recipe version

Including license text in images

- Simple way to ensure end users receive license text
- In local.conf or a distro conf you can set:
  - `COPY_LIC_DIRS = “1”`
    - **Places license text for each package into /usr/share/common-licenses**
  - `COPY_LIC_MANIFEST = “1”`
    - **Places previously discussed license.manifest into /usr/share/common-licenses**

Including license text in images (2)

- One caveat…
- `COPY_LIC_DIRS` and `COPY_LIC_MANIFEST` only cover packages installed during image creation
- Licenses for packages installed via on-target package management are not handled by these methods

Creating license packages

- **Another variable you can set:**
  - `LICENSE_CREATE_PACKAGE = “1”`
- **For each recipe this creates a** `${PN}-lic` **package**
  - **E.g.** `busybox-lic`
- **Adds this as an RRECOMMENDS for the base package**
- **Installs licenses into** `/usr/share/licenses/${PN}`
  - **E.g.** `/usr/share/licenses/busybox`

Providing recipes

- The archiver can be used to provide recipes
  - Creates tarball of the bb file, bbappends & includes
- However, this makes it difficult for users to rebuild images
- It can be argued from the GPL that providing full layers is required
  - “scripts used to control compilation and installation”
  - I’m not a lawyer!

Providing recipes (2)

- The best way to handle this is to release your layers
- Also ensure you snapshot bitbake and third party layers used to build release images
- Recommend you also provide bblayers.conf, local.conf and any other customisation

Releasing your layer

- Releasing publically as an open source layer is easiest
  - You can add your layer to http://layers.openembedded.org/
- However, you can also release privately to customers
  - Give people a source archive or a download link with your product or images

Providing the correct versions

- Please don’t just point people at a layer repository or branch
- Make sure they get the same exact versions of bitbake and metadata which was used to build your image
- Many ways to do this
  - Tarball
  - Git submodules
  - Repo tool

Avoid AUTOREV for releases

- Setting `SRCREV = “${AUTOREV}”` can be great in development
- Terrible for releases
- People receiving your layer may need to rebuild months or years later and could get a different git commit
- Always explicitly set SRCREV when building releases

Don’t be clever

```
DESCRIPTION = "Node.js modules"
LICENSE = "MIT & ISC & Apache-2 & FIPL-1.0 & BSD-2-Clause"
DEPENDS = "nodejs-native glfw glew cairo pango jpeg libpng"
DEPENDS_class-native = "nodejs-native"
PROVIDES = "nodejs-modules"
PR = "r2
S = "${WORKDIR}/${PN}-${PV}"
PD= "${PN}-${PV}/packages"
require packages.inc
```

Don’t be clever

```
DESCRIPTION = "Node.js modules"
LICENSE = "MIT & ISC & Apache-2 & FIPL-1.0 & BSD-2-Clause"
DEPENDS = "nodejs-native glfw glew cairo pango jpeg libpng"
DEPENDS_class-native = "nodejs-native"
PROVIDES = "nodejs-modules"
PR = "r2
S = "${WORKDIR}/${PN}-${PV}"
PD= "${PN}-${PV}/packages"
require packages.inc
```

Wait? Wot?

Don’t be clever

```
SRC_URI+= "http://registry.npmjs.org/put/-/put-0.0.6.tgz;name=0017put;unpack=yes;downloadfilename=put-
0.0.6.tgz;subdir=${PD}/0017-put-0.0.6"
LIC_FILES_CHKSUM += "file://real/put-0.0.6/package/LICENSE;md5=b2d989bc186e7f6b418a5fdd5cc0b56b"

SRC_URI+= "http://registry.npmjs.org/sax/-/sax-1.2.1.tgz;name=0018sax;unpack=yes;downloadfilename=sax-
1.2.1.tgz;subdir=${PD}/0018-sax-1.2.1"
LIC_FILES_CHKSUM += "file://real/sax-1.2.1/package/LICENSE;md5=326d5674181c4bb210e424772c60fa80"

SRC_URI+= "http://registry.npmjs.org/through/-/through-
2.3.8.tgz;name=0019through;unpack=yes;downloadfilename=through-2.3.8.tgz;subdir=${PD}/0019-through-2.3.8"
LIC_FILES_CHKSUM += "file://real/through-
2.3.8/package/readme.markdown;md5=6ff48d70322f9b54b7f36536954bca06"
LIC_FILES_CHKSUM += "file://real/through-
2.3.8/package/LICENSE.APACHE2;md5=ffcf739dca268cb0f20336d6c1a038f1"
LIC_FILES_CHKSUM += "file://real/through-2.3.8/package/LICENSE.MIT;md5=e0f70a42adf526e6f5e605a94d98a420"

SRC_URI+=
```

Trust but verify

- meta-license-tools + fossup + fossology
- Patched archiver scans
- SLOW!!! But finds issues
- A lot of knowledge needed about what you’re actually distributing.

```
USER_CLASSES += "license archiver"
COPYLEFT_LICENSE_INCLUDE = "GPL* AGPL* LGPL* MPL*"
COPYLEFT_LICENSE_EXCLUDE = "CLOSED Proprietary"
ARCHIVER_MODE[src] = "patched"
ARCHIVER_MODE[diff] = "0"
ARCHIVER_MODE[dumpdata] = "0"
ARCHIVER_MODE[recipe] = "1"
COPYLEFT_RECIPE_TYPES = "target"
INHERIT += "fossology”
VM_SPRINT_NUMBER = "054"
```

Trust but verify

Lets do a FOSSOLOGY CLEARANCE!