Bitbake 101

Running the Yocto Project workflow

BitBake for the busy developer

Slides online

https://a4z.gitlab.io/talks/embed/bitbake101/

Name: Harald Achitz
Profession: Freelancer, currently at Tobii

Keywords: Developer
Architect
Build / Automation Engineer
C++ Community organizer

https://a4z.gitlab.io/about

For more info, please find me on linkedin

BitBake

The Yocto

Command **L**ine **I**nterface

Yocto in one sentence

The Yocto Project provides a modular construction system to create a custom Linux distribution

Yocto in one sentence

for the busy developer

When Raspian isn’t enough anymore

Motivation

> *Using the Yocto Project is fairly easy, until **something goes wrong**. Without an understanding of how the build process works, you'll find yourself trying to troubleshoot "a black box".*

What I wish I'd known about Yocto Project — Yocto documentation

BitBake

A generic task execution engine

that follows **recipes** and **configurations**

in a specific format in order to perform defined tasks

BitBake

A generic task execution engine

that follows **recipes** and **configurations**

in a specific format in order to perform defined tasks

A practical guide to BitBake

https://a4z.gitlab.io/docs/BitBake/guide.html

(search for bitbake tutorial)

Creating packages for Linux

> *Using the Yocto Project is fairly easy, until something goes wrong. Without an* ***understanding of how the build process works****, you'll find yourself trying to troubleshoot "a black box".*

What I wish I'd known about Yocto Project — Yocto documentation

Creating packages for Linux

- Context
- Properties
- Tasks

Creating packages for Linux

Context

- Compiler
- File locations
- General compilation flags
- Development machine / target host
- Recipes
- …

Creating packages for Linux

Properties

- Name
- Version
- Download locations
- License information
- Dependencies (build/run)
- …

Creating packages for Linux

Tasks

- Download source
- Extract the source
- Patch
- Configure
- Compile
- Install
- Package
- Test or check the package

Creating packages for Linux

- Context
- Properties
- Tasks

Context

```
cd yocto/poky

source oe-init-build-env [build-fold]
```

Context

Add some more context

```
bitbake-layers add-layer ../meta-raspberrypi/
```

Properties

e.g. setup properties of the context

```
echo "MACHINE = raspberrypi3" >> conf/local.conf
```

Properties

```
MACHINE = "raspberrypi3"
DL_DIR ?= "${TOPDIR}/../build-state/downloads"
SSTATE_DIR ?= "${TOPDIR}/../build-state/sstate-cache
BB_NUMBER_THREADS = "7"
PARALLEL_MAKE = "-j 10"
ENABLE_UART = "1"
```

Tasks

The actual work

Tasks

```
Currently 11 running tasks (2994 of 3973)  75% |
####################################################
0: linux-raspberrypi-1_5.15.34+gitAUTOINC+e1b976ee4
1: glibc-locale-2.35-r0 do_package_write_rpm - 8m0s
2: openssl-3.0.2-r0 do_compile - 7m41s (pid 707593)
3: perl-5.34.1-r0 do_compile - 2m13s (pid 1022255)
4: libarchive-3.6.1-r0 do_compile - 1m0s (pid 109482
5: libidn2-2.3.2-r0 do_configure - 34s (pid 1119642)
6: coreutils-9.0-r0 do_configure - 30s (pid 1122702)
7: icu-70.1-r0 do_compile - 27s (pid 1125520)
8: libxrandr-1_1.5.2-r0 do_configure - 2s (pid 11291
9: util-linux-2.37.4-r0 do_compile_ptest_base - 2s (
10: libpciaccess-0.16-r0 do_compile - 0s (pid 114405
```

Tasks

- Tasks are defined in recipes
- There are many tasks
- There are many recipes

Tasks

Many tasks are repetitive

They do always the same

```
wget ...
tar ...
cd ...
configure ...
make -j $(nproc)
make install DESTDIR ...
package ...
...
```

Tasks

Many tasks are repetitive

They do always the same

except when they don’t

```
wget ... or git ?
tar ... or bzip2, gunzip ...
cd ... to where
configure ... or cmake, or just make ?
make .. or is it rust, go, python ... ?
```

Tasks

```
bitbake zlib -c listtask
```

bitbake zlib -c listtask

```
do_build , do_checkuri , do_clean , do_cleanall
do_cleansstate , do_compile
do_compile_ptest_base , do_configure
do_configure_ptest_base
do_deploy_source_date_epoch
do_deploy_source_date_epoch_setscene
do_devshell , do_fetch , do_install
do_install_ptest_base , do_listtasks , do_package
do_package_qa , do_package_qa_setscene
do_package_setscene , do_package_write_rpm
do_package_write_rpm_setscene , do_packagedata
do_packagedata_setscene , do_patch
do_populate_lic , do_populate_lic_setscene
do_populate_sysroot , do_populate_sysroot_setscene
do_prepare_recipe_sysroo , do_pydevshell do_unpack
```

bitbake zlib-native -c listtask

do_addto_recipe_sysroot , do_build , do_checkuri
do_clean , do_cleanall , do_cleansstate
do_compile , do_configure
do_deploy_source_date_epoc
do_deploy_source_date_epoch_setscene
do_devshell , do_fetch , do_install
do_listtasks , do_patch , do_populate_lic
do_populate_lic_setscene , do_populate_sysroot
do_populate_sysroot_setscene
do_prepare_recipe_sysroo
do_pydevshell , do_unpack

Tasks

Many tasks are repetitive

They do always the same

except when they don’t

Recipes are written in a DSL

BitBake DSL

BitBake DSL

The balance act

Remove repetitive boilerplate

Make it convenient to build software

(it’s never convenient to build software, imho)

BitBake DSL

Does a good job

Pretty good readable with knowing some basics

BitBake DSL

The best case recipe

```
SUMMARY = "LibCool is awesome"
DESCRIPTION = "LibCool does cool stuff"
HOMEPAGE = "https://libcool.org/"
SECTION = "libs/devel"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://License;md5=123456789abc
SRC_URI = "https://libcool.org/libcool-${PV}.tar.gz
SRC_URI[sha256sum] = "abcdef12345678912345678 9abcde
S = "${WORKDIR}/libcool-${PV}"

inherit autotools
```

BitBake DSL

The *best case* rarely happens

- Define dependencies
- Extend tasks
- Do tasks different
- Start a service
- Define extra flags
- Patch source
- …

BitBake DSL

- Context
- Properties
- Tasks

BitBake DSL

- Files (and file locations)
- Variables
- Functions

BitBake DSL, Files

File types

- Configuration
- Recipes
- Include and Append files
- Classes

BitBake DSL, Files

File locations

```
build-folder (working dir)
|-- conf/local.conf
|-- conf/layer.conf
...
meta
|-- classes/
|-- conf/
|-- recipes-core/
meta-poky/
|-- classes/
|-- conf/
|-- recipes-core/
```

BitBake DSL, Variables

BitBake DSL, Variables

```
VALUE = "123"
MESSAGE = "value ${VALUE}" (1)
VALUE = "456"

do_build(){
  echo "Message ${MESSAGE}"
}
```

```
bitbake -c build recipe

⇒ Message value 456
```

(1) Lazy evaluation

BitBake DSL, Variables

```
VALUE = "123"
MESSAGE := "value ${VALUE}"   (1)
VALUE = "456"

do_build(){
  echo "Message ${MESSAGE}"
}
```

```
bitbake -c build recipe

⇒ Message value 123
```

(1) Greedy evaluation

BitBake DSL, Variables

Appending and prepending with space

```
MESSAGE = "value"
MESSAGE += "123"
MESSAGE =+ "Message"
```

⇒ `Message value 123`

BitBake DSL, Variables

Appending and prepending without space

```
MESSAGE = "value"
MESSAGE .= "123"
MESSAGE =. "Message"
```

⇒ `Messagevalue123`

BitBake DSL, Variables

Default values

```
VALUE ?= "123"
VALUE ?= "456"

do_build(){
  echo "Message ${VALUE}"
}
```

`bitbake -c build recipe`

⇒ `Message 123`

First wins

BitBake DSL, Variables

Default values

```
VALUE ??= "123"
VALUE ??= "456"

do_build(){
  echo "Message ${VALUE}"
}
```

`bitbake -c build recipe`

⇒ `Message 456`

Last wins

BitBake DSL, Variables

Inline Python Variable Expansion

```
DATE = "${@time.strftime('%Y%m%d',time.gmtime())}"
```

BitBake DSL, Functions

BitBake DSL, Functions

Shell script

The natural choice for building software

Python

When shell scripts feel not natural anymore

BitBake DSL, Functions

Shell functions

BitBake DSL, Functions

Shell functions

```
do_install () {
  autotools_do_install
  install -d ${D}${bindir}/
  install -m 0755 libtool ${D}${bindir}/
}
```

BitBake DSL, Functions

Extend some task

```
do_install:append() {
  install -d ${D}${sysconfdir}/udhcpc.d
  install ${WORKDIR}/00avahi-autoipd \
          ${D}${sysconfdir}/udhcpc.d
  install ${WORKDIR}/99avahi-autoipd \
          ${D}${sysconfdir}/udhcpc.d
}
```

BitBake DSL, Functions

Extend some task

```
do_configure:prepend () {
  # Remove any existing libtool m4 since old
  # stale versions would break any upgrade
  rm -f ${STAGING_DATADIR}/aclocal/libtool.m4
  rm -f ${STAGING_DATADIR}/aclocal/lt*.m4
}
```

BitBake DSL, Functions

Python functions

BitBake DSL, Functions

Python functions

```
python __anonymous() {
   if not bb.utils.contains('DISTRO_FEATURES',
                            'sysvinit', True, False, d):
      d.setVar("INHIBIT_UPDATERCD_BBCLASS", "1")
}
```

BitBake DSL, Functions

Python functions

```
python () {
    if d.getVar("BB_CURRENT_MC") == "mc_2":
        bb.fatal("Multiconfig is mc_2")
}
```

BitBake DSL, Functions

Python functions

```
python do_configure() {
    bb.build.exec_func('build_efi_cfg', d)
}
```

BitBake DSL, Functions

Add add a task

```
python do_display_banner() {
    bb.plain("**********************");
    bb.plain("*                    *");
    bb.plain("*  Example recipe *");
    bb.plain("*                    *");
    bb.plain("**********************");
}

addtask display_banner before do_build
```

BitBake DSL

Variable and function tagging

BitBake DSL, tagging

First time seen, this might surprise

```
VARIABLE = "var value"
VARIABLE[tagname] = "tag value"
```

BitBake DSL, tagging

Can later be accessed via the datastore

```
VARIABLE = "var value"
VARIABLE[tagname] = "tag value"

print (d.getVar("VARIABLE"),
       d.getVarFlags("VARIABLE")["tagname"])
```

BitBake DSL, tagging

Predefined example `cleandirs`

Ensure an empty directory exists before task execution

```
do_populate_sdk[cleandirs] = "${SDKDEPLOYDIR}"
```

BitBake DSL

Include, append and class files

BitBake DSL, include files

Include and/or require include files

```
include file1.inc

require from/my-layer/file2.inc
```

BitBake DSL, include files

Use case for include files:

- recipe.inc
- recipe_1.2.3.bb (includes recipe.inc)
- recipe_4.5.6.bb (includes recipe.inc)

BitBake DSL, append files

*.bbappend files

Customize recipes from an other layer

BitBake DSL, append files

Yocto / poky does not have many *.bbappend files

Other, specialized layers might have some to customize recipes from yocto/poky

BitBake DSL, Classes

Create a *is-a* relation and import functionality

BitBake DSL, Classes

Usage: I am a cmake build

```
inherit cmake
```

BitBake DSL, Classes

Multiple inheritance is possible

```
inherit pkgconfig systemd
```

BitBake DSL, Classes

inherit pkgconfig

```
cat meta/classes/pkgconfig.bbclass
```

```
1  DEPENDS:prepend = "pkgconfig-native "
2
```

BitBake DSL, Classes

Classes are defined in a *.bbclass file.

Classes should be under meta[-name]/classes folder

BitBake DSL, Classes

The 'public interface' of a class

```
EXPORT_FUNCTIONS do_configure do_compile do_install
```

BitBake DSL, Classes

Yocto defines many classes

BitBake DSL, Classes

There needs to be one base.bbclass

Yocto defines this class
(meta/classes/base.bbclass)

BitBake DSL

Pretty good readable with knowing some basics

BitBake DSL

Basic machinery is setup

(download, patch, configure, ….)

Fill in the gaps and do some customization

BitBake DSL

Don’t use bashism, sh is used. Use Python.

BitBake DSL

There is of course more …

the busy developer should find the kickstart here

Where to go from here?

Where to go from here?

- Download a yocto distribution
- Follow **Yocto Project Quick Build**
- Read some configs and recipes
- Explore the environment

Where to go from here?

- Get your first board
- Find out how to build for that
- Do it

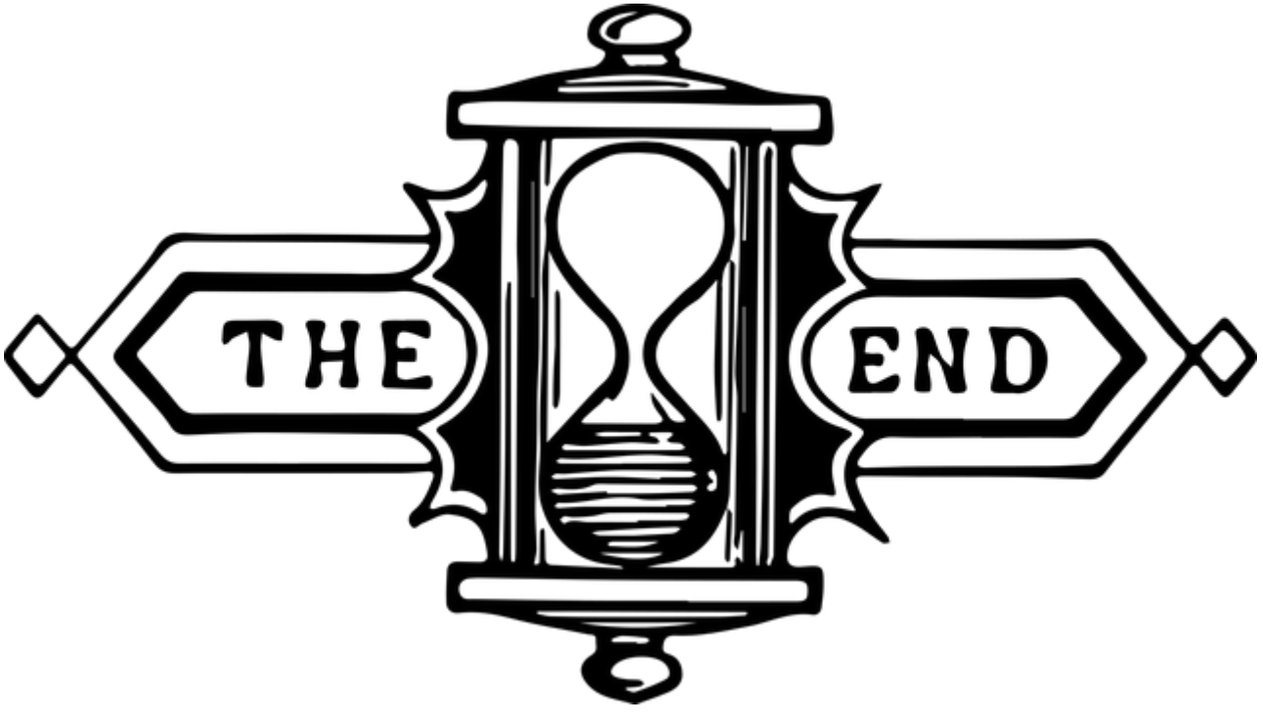
If your first board is a Raspberry, search for *how to enable UART*.

Use proper usb-serial debug cable.

Get familiar with minicom.

Where to go from here?

Listen to all the other wonderful talks of this conference!



Thanks for listening