# Creating a Custom Embedded Linux Distribution Using the Yocto Project

**Part 1**

General concepts, BitBake, Recipes, Troubleshooting

This section will introduce the Yocto Project main concepts

## GENERAL CONCEPTS

## Yocto Project Overview

- Collection of tools and methods enabling
- Rapid evaluation of embedded Linux on many popular off-the-shelf boards
- Easy customization of distribution characteristics
- Supports x86, ARM, MIPS, Power, RISC-V
- Based on technology from the OpenEmbedded Project
- Layer architecture allows for easy re-use of code

## What is the Yocto Project?

- **Umbrella organization under Linux Foundation**
- **Backed by many companies interested in making Embedded Linux easier for the industry**
- **Co-maintains OpenEmbedded Core and other  tools (including opkg)**

## Yocto Project Governance

- Organized under the Linux Foundation
- Split governance model
- Technical Leadership Team
- Advisory Board made up of participating  organizations

## Yocto Project Member Organizations

PLATINUM MEMBERS

intel arm cisco XILINX COMCAST

aws Microsoft facebook WNDRVR

GOLD MEMBERS

TEXAS INSTRUMENTS SIEMENS AUTOMOTIVE GRADE LINUX HUAWEI

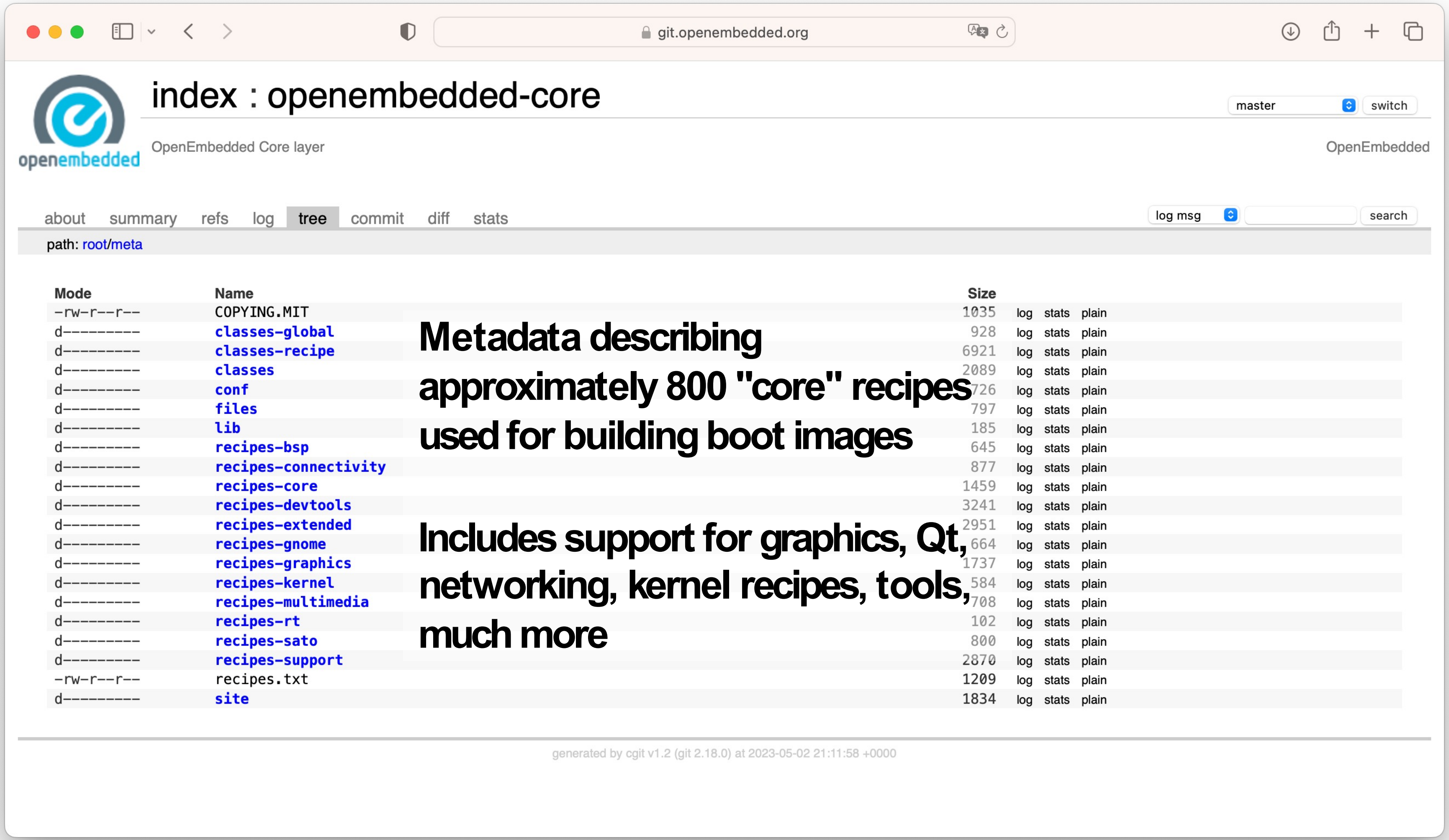
SILVER MEMBERS

ST life.augmented LINEO Solutions MontaVista DELL LG Electronics AMD NXP Linaro Savoir-faire LINUX RENESAS FOUNDRIES.IO

## Yocto Project Overview

- **YP builds packages – then uses these packages to build bootable images**
- **Supports use of popular package formats**
  - Including rpm, deb, ipk
- **Releases on a 6-month cadence**
- **App Development Tools including SDK,  toaster**
- **Latest (stable) kernel, toolchain and packages,  documentation**

## Yocto is based on OpenEmbedded-core



index : openembedded-core

OpenEmbedded Core layer

OpenEmbedded

path: root/meta

| Mode | Name | Size | | | |
|---|---|---|---|---|---|
| -rw-r--r-- | COPYING.MIT | 1035 | log | stats | plain |
| d--------- | classes-global | 928 | log | stats | plain |
| d--------- | classes-recipe | 6921 | log | stats | plain |
| d--------- | classes | 2089 | log | stats | plain |
| d--------- | conf | 726 | log | stats | plain |
| d--------- | files | 797 | log | stats | plain |
| d--------- | lib | 185 | log | stats | plain |
| d--------- | recipes-bsp | 645 | log | stats | plain |
| d--------- | recipes-connectivity | 877 | log | stats | plain |
| d--------- | recipes-core | 1459 | log | stats | plain |
| d--------- | recipes-devtools | 3241 | log | stats | plain |
| d--------- | recipes-extended | 2951 | log | stats | plain |
| d--------- | recipes-gnome | 664 | log | stats | plain |
| d--------- | recipes-graphics | 1737 | log | stats | plain |
| d--------- | recipes-kernel | 584 | log | stats | plain |
| d--------- | recipes-multimedia | 708 | log | stats | plain |
| d--------- | recipes-rt | 102 | log | stats | plain |
| d--------- | recipes-sato | 800 | log | stats | plain |
| d--------- | recipes-support | 2870 | log | stats | plain |
| -rw-r--r-- | recipes.txt | 1209 | log | stats | plain |
| d--------- | site | 1834 | log | stats | plain |

generated by cgit v1.2 (git 2.18.0) at 2023-05-02 21:11:58 +0000

**Metadata describing approximately 800 "core" recipes used for building boot images**

**Includes support for graphics, Qt, networking, kernel recipes, tools, much more**

## Intro to OpenEmbedded

- **The OpenEmbedded Project co-maintains OE-core build system:**
  - **bitbake** build tool and **scripts**
  - **Metadata** and **configuration**
- **Provides a central point for new metadata**
- **(see the OE Layer index)**

## What is BitBake?

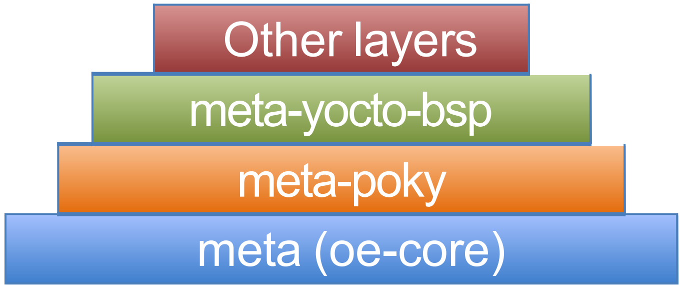
- **BitBake**
  - Powerful and flexible  build engine (Python)
  - Reads metadata
  - Determines  dependencies
  - Schedules tasks

- **Metadata**
  - A structured  collection of "recipes" which tell BitBake what to build, organized in layers

## OK, so what is Poky?

- **Poky is a reference distribution**
- **Poky has its own git repo**
  - `git clone git://git.yoctoproject.org/poky`

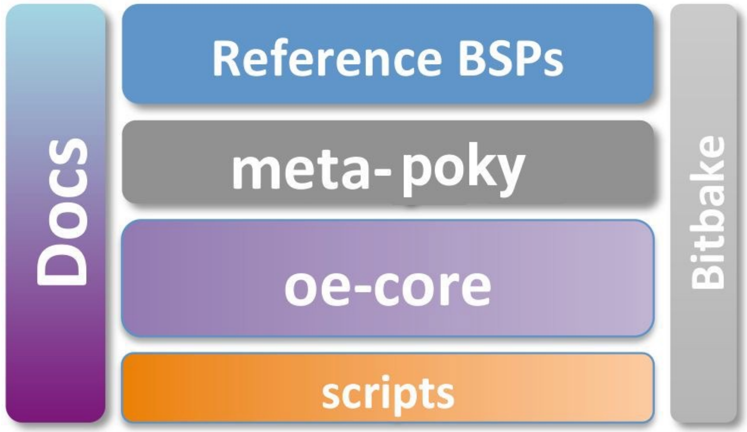
- **Primary Poky layers**
  - oe-core (`poky/meta`)
  - meta-poky (`poky/meta-poky`)
  - meta-yocto-bsp



- **Poky is the starting point for building things with the Yocto Project**

## Poky in Detail

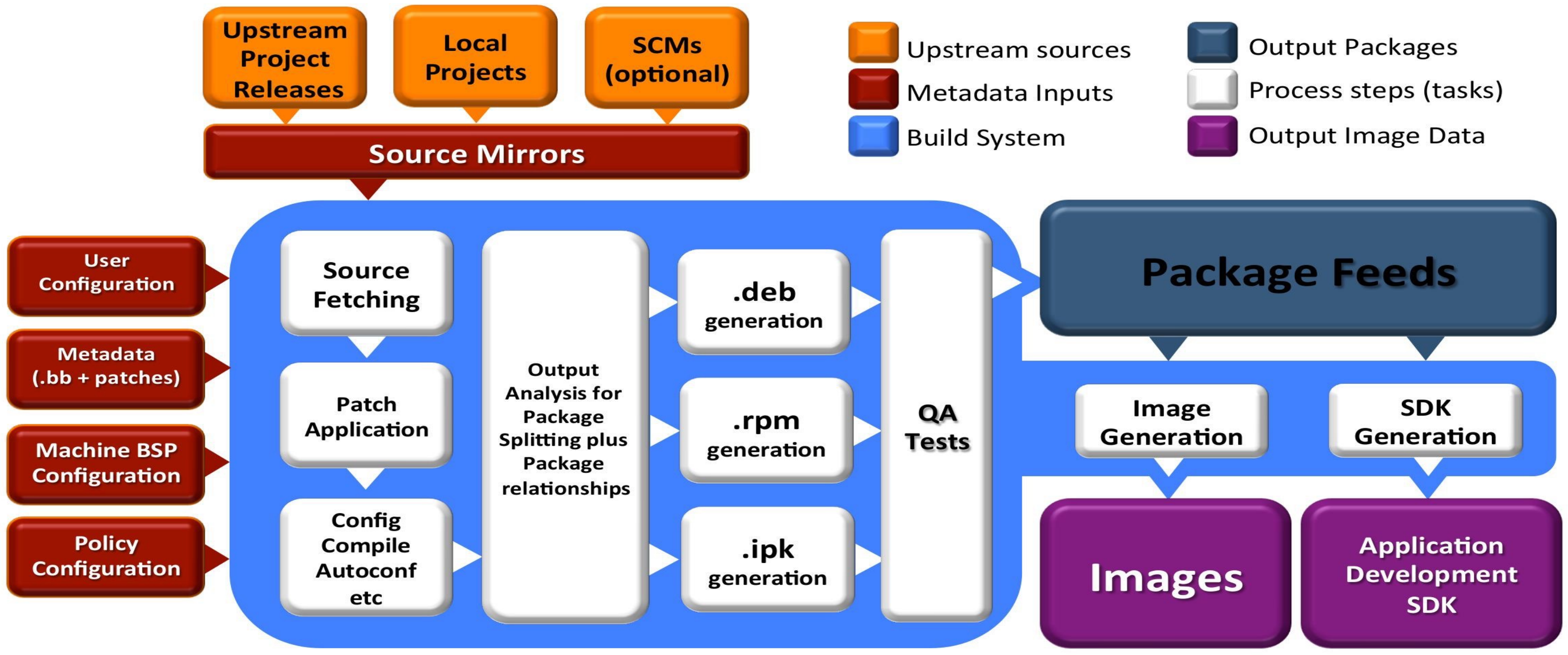
- **Contains core components**
  - Bitbake tool: A python-based build engine
  - Build scripts (infrastructure)
  - Foundation package recipes (oe-core)
  - meta-poky (Contains distribution policy)
  - Reference BSPs
  - Yocto Project documentation



## Putting It All Together

- Yocto Project is a large collaboration project
- OpenEmbedded is providing most metadata
- Bitbake is the build tool
- Poky is the Yocto Project's reference distribution
- Poky contains a version of bitbake and oe-core from which you can start your project

# Build System Workflow

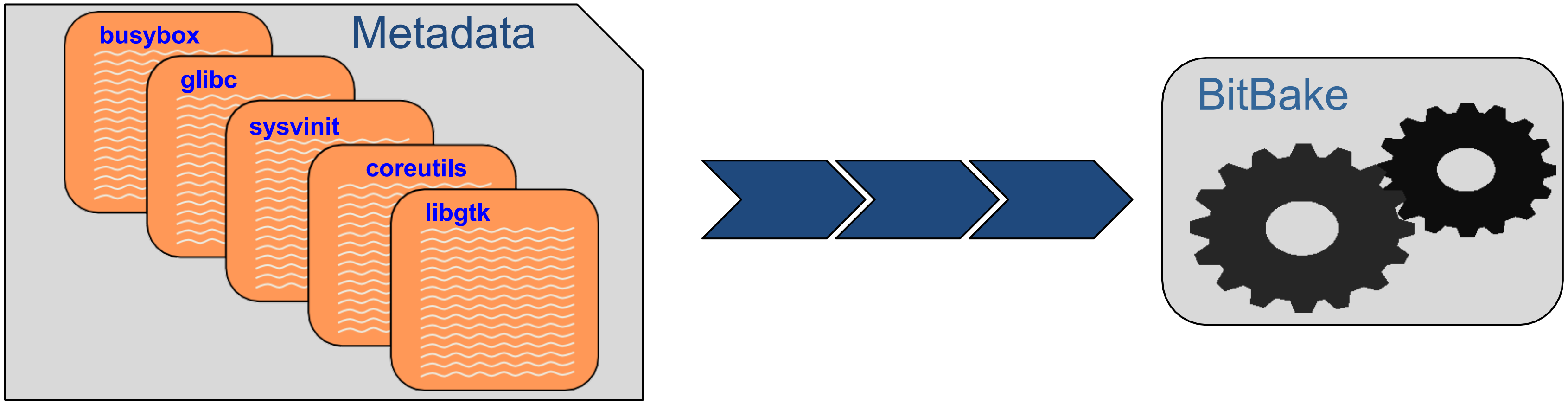


This section will introduce the concept of the bitbake build tool  and how it can be used to build recipes

## BITBAKE

# Metadata and BitBake

- Most common form of metadata: the recipe
- A recipe provides a “list of ingredients” and “cooking instructions”
- Defines settings and a set of tasks used by bitbake to build binary packages



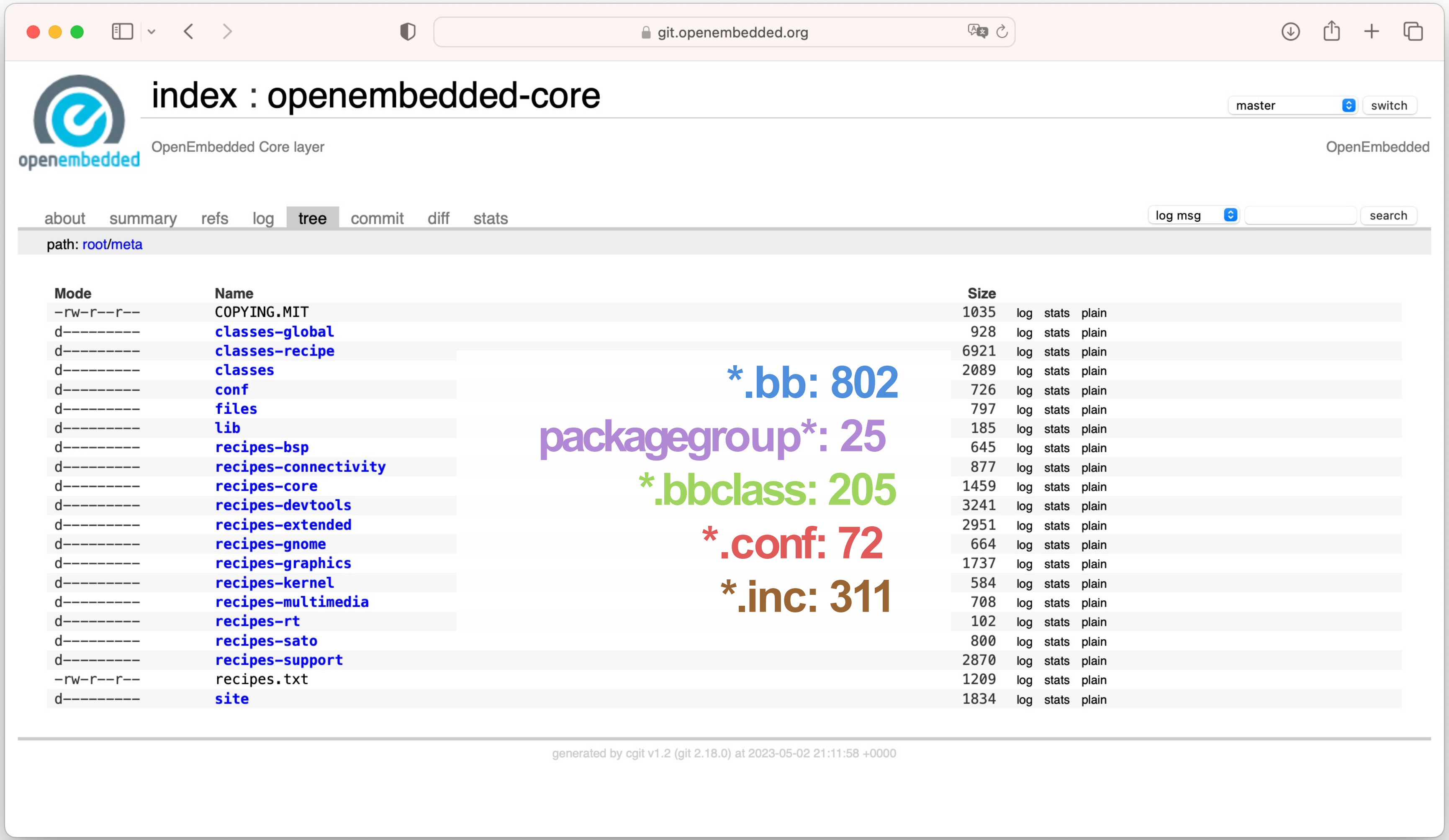
## What is Metadata?

- **Metadata exists in four general categories:**
- **Recipes (*.bb)**
  - Usually describe build instructions for a single package
- **PackageGroups (special *.bb)**
  - Often used to group packages together for a FS image
- **Classes (*.bbclass)**
  - Inheritance mechanism for common functionality
- **Configuration (*.conf)**
  - Drives the overall behaviour of the build process

## Other Metadata

- **Append files (*.bbappend)**
  - Define additional metadata for a similarly named .bb file
  - Can add or override previously set values
- **Include files (*.inc)**
  - Files which are used with the include directive
  - Also, can be included with require (mandatory include)
  - Include files are typically found via the BBPATH variable

# OE-CORE Breakdown



index : openembedded-core

OpenEmbedded Core layer

OpenEmbedded

path: root/meta

| Mode | Name | Size | | | |
|---|---|---|---|---|---|
| -rw-r--r-- | COPYING.MIT | 1035 | log | stats | plain |
| d--------- | classes-global | 928 | log | stats | plain |
| d--------- | classes-recipe | 6921 | log | stats | plain |
| d--------- | classes | 2089 | log | stats | plain |
| d--------- | conf | 726 | log | stats | plain |
| d--------- | files | 797 | log | stats | plain |
| d--------- | lib | 185 | log | stats | plain |
| d--------- | recipes-bsp | 645 | log | stats | plain |
| d--------- | recipes-connectivity | 877 | log | stats | plain |
| d--------- | recipes-core | 1459 | log | stats | plain |
| d--------- | recipes-devtools | 3241 | log | stats | plain |
| d--------- | recipes-extended | 2951 | log | stats | plain |
| d--------- | recipes-gnome | 664 | log | stats | plain |
| d--------- | recipes-graphics | 1737 | log | stats | plain |
| d--------- | recipes-kernel | 584 | log | stats | plain |
| d--------- | recipes-multimedia | 708 | log | stats | plain |
| d--------- | recipes-rt | 102 | log | stats | plain |
| d--------- | recipes-sato | 800 | log | stats | plain |
| d--------- | recipes-support | 2870 | log | stats | plain |
| -rw-r--r-- | recipes.txt | 1209 | log | stats | plain |
| d--------- | site | 1834 | log | stats | plain |

*.bb: 802

packagegroup*: 25

*.bbclass: 205

*.conf: 72

*.inc: 311

generated by cgit v1.2 (git 2.18.0) at 2023-05-02 21:11:58 +0000

## Introduction to Bitbake

- **Bitbake is a task executor and scheduler**
- **By default, the *build* task for the specified recipe is executed**

```
$ bitbake myrecipe
```

- **You can indicate which task you want run**

```
$ bitbake -c clean myrecipe
$ bitbake -c cleanall myrecipe
```

- **You can get a list of tasks with**

```
$ bitbake -c listtasks myrecipe
```

## Building Recipes

- **By default, the highest version of a recipe is built**
  - Can be overridden  with DEFAULT_PREFERENCE or PREFERRED_VERSION metadata

```
$ bitbake myrecipe
```

- **You can specify the version of the package you want built (version of upstream source)**

```
$ bitbake myrecipe-1.0
```

- **You can also build a particular revision of the package metadata**

```
$ bitbake myrecipe-1.0-r0
```

- **Or you can provide a recipe file to build**

```
$ bitbake -b mydir/myrecipe.bb
```

## Running bitbake for the First Time

- **Running bitbake normally will stop on the first error found**

```
$ bitbake core-image-minimal
```

- **When running a long build (e.g. overnight) you want as much of the build done as possible before debugging issues**
- **Running with `--continue` (`-k`) means bitbake will proceed as far as possible after finding an error**

```
$ bitbake -k core-image-minimal
```

## Bitbake is a Task Scheduler

- **Bitbake builds recipes by scheduling build tasks in parallel**

  ```
  $ bitbake recipe
  ```

  - This looks for `recipe.bb` in BBFILES
- **Each recipe defines build tasks, each which can depend on other tasks**
- **Recipes can also depend on other recipes, meaning more than one recipe may be built**
- **Tasks from more than one recipe are often executed in parallel at once on multi-cpu build machines**

## Recipe Basics – Default Tasks*

| | |
|---|---|
| `do_fetch` | Locate and download source code |
| `do_unpack` | Unpack source into working directory |
| `do_patch` | Apply any patches |
| `do_configure` | Perform any necessary pre-build configuration |
| `do_compile` | Compile the source code |
| `do_install` | Installation of resulting build artifacts in WORKDIR |
| `do_populate_sysroot` | Copy artifacts to sysroot |
| `do_package_*` | Create binary package(s) |

Note: to see the list of all possible tasks for a recipe, do this:

```
$ bitbake -c listtasks <recipe_name>
```

*Simplified for illustration

## Simple recipe task list*

```
$ bitbake hello

NOTE: Running task 337 of 379 (ID: 4, hello_1.0.0.bb, do_fetch)
NOTE: Running task 368 of 379 (ID: 0, hello_1.0.0.bb, do_unpack)
NOTE: Running task 369 of 379 (ID: 1, hello_1.0.0.bb, do_patch)
NOTE: Running task 370 of 379 (ID: 5, hello_1.0.0.bb, do_configure)
NOTE: Running task 371 of 379 (ID: 7, hello_1.0.0.bb, do_populate_lic)
NOTE: Running task 372 of 379 (ID: 6, hello_1.0.0.bb, do_compile)
NOTE: Running task 373 of 379 (ID: 2, hello_1.0.0.bb, do_install)
NOTE: Running task 374 of 379 (ID: 11, hello_1.0.0.bb, do_package)
NOTE: Running task 375 of 379 (ID: 3, hello_1.0.0.bb, do_populate_sysroot)
NOTE: Running task 376 of 379 (ID: 8, hello_1.0.0.bb, do_packagedata)
NOTE: Running task 377 of 379 (ID: 12, hello_1.0.0.bb, do_package_write_ipk)
NOTE: Running task 378 of 379 (ID: 9, hello_1.0.0.bb, do_package_qa)
```

*Simplified for illustration

## SSTATE CACHE

- **Several bitbake tasks can use past versions of build artefacts if there have been no changes since the last time you built them**

| | |
|---|---|
| do_packagedata | Creates package metadata used by the build system to generate the final packages |
| do_package | Analyzes the content of the holding area and splits it into subsets based on available packages and files |
| do_package_write_rpm | Creates the actual RPM packages and places them in the Package Feed area |
| do_populate_lic | Writes license information for the recipe that is collected later when the image is constructed |
| do_populate_sysroot | Copies a subset of files installed by do_install into the sysroot in order to make them available to other recipes |

## Simple recipe build from sstate cache*

```
$ bitbake -c clean hello
$ bitbake hello
```

```
NOTE: Running setscene task 69 of 74 (hello_1.0.0.bb, do_populate_sysroot_setscene)
NOTE: Running setscene task 70 of 74 (hello_1.0.0.bb, do_populate_lic_setscene)
NOTE: Running setscene task 71 of 74 (hello_1.0.0.bb, do_package_qa_setscene)
NOTE: Running setscene task 72 of 74 (hello_1.0.0.bb, do_package_write_ipk_setscene)
NOTE: Running setscene task 73 of 74 (hello_1.0.0.bb, do_packagedata_setscene)
```

*Simplified for illustration

This section will introduce the concept of metadata and recipes and how they can be used to automate the building of packages

## RECIPES

## What is a Recipe?

- **A recipe is a set of instructions for building packages, including:**
  - Where to obtain the upstream sources and which patches to apply (this is called "fetching")
  - **SRC_URI**
- **Dependencies (on libraries or other recipes)**
  - **DEPENDS**, **RDEPENDS**
- **Configuration/compilation options**
  - **EXTRA_OECONF**, **EXTRA_OEMAKE**
- **Define which files go into what output packages**
  - **FILES_***

# Example Recipe – ethtool_3.15.bb

```
SUMMARY = "Display or change ethernet card settings"
DESCRIPTION = "A small utility for examining and tuning the settings of your
ethernet-based network interfaces."
HOMEPAGE = "http://www.kernel.org/pub/software/network/ethtool/"
SECTION = "console/network"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=b234ee4d69f5fce4486a80fdaf4a4263 \
file://ethtool.c;beginline=4;endline=17;md5=c19b30548c582577fc6b443626fc1216"

SRC_URI = "${KERNELORG_MIRROR}/software/network/ethtool/ethtool-${PV}.tar.gz \
           file://run-ptest \
           file://avoid_parallel_tests.patch \
           "

SRC_URI[md5sum] = "fbb24aa414ab9a11ff66d351b5b8493b"
SRC_URI[sha256sum] =
"0b25a46b61bc6e26f56dcb973fc28acea159e2c32c9d6da42c0fa8d1a6339829"

UPSTREAM_CHECK_URI = "https://www.kernel.org/pub/software/network/ethtool/"
…
```

## What can a Recipe Do?

- **Build one or more packages from source code**
  - Host tools, compiler, utilities
  - Bootloader, Kernel, etc.
  - Libraries, interpretors, etc.
  - User-space applications
- **Package Groups**
- **Full System Images**

## Recipe Operators

- **BitBake files have their own syntax**
  - The syntax has similarities to several other languages but also has some unique features

| | | |
|---|---|---|
| **A = “foo”** | **(late assignment)** | |
| **B ?= “ot”** | **(default value)** | |
| **C ??= “abc”** | **(late default)** | |
| **D := “xyz”** | **(immediate assignment)** | |
| **A .= “bar”** | **(append)** | **-> “foobar”** |
| **B =. “wo”** | **(prepend)** | **-> “woot”** |
| **C += “def”** | **(append)** | **-> “foo bar”** |
| **D =+ “uvw”** | **(prepend)** | **-> “uvw xyz”** |

## More Recipe Operators

A = “foo”

B = “ot”

| | |
|---|---|
| ■ A:append  = “bar” | -> “foobar” |
| ■ B:prepend = “wo” | -> “woot” |
| ■ A:remove = “oob” | -> “far” |
| ■ B:remove = “oo” | -> “wt” |

## Recipe Override Operators

- Bitbake uses OVERRIDES to control what variables are overridden after bitbake parses recipes and configuration files

- OVERRIDES = "architecture:os:machine"
- TEST = "default"
- TEST:os = "osspecific"
- TEST:nooverride = "othercondvalue"

## Unsetting Variables

- **It is possible to completely remove a variable or a variable flag from bitbake's internal data dictionary by using the "unset" keyword**

- **unset DATE** (**unset variable**)
- **unset do_fetch[noexec]** (**unset task attribute/flag**)

## Bitbake Variables/Metadata

- **These are set automatically by bitbake**
  - **TOPDIR** – The build directory
  - **LAYERDIR** – Current layer directory
  - **FILE** – Path and filename of file being processed
- **Policy variables control the build**
  - **BUILD_ARCH** – Host machine architecture
  - **TARGET_ARCH** – Target architecture
  - And many others...

## Build Time Metadata

- PN – Pakage name ("myrecipe")
- PV – Package version (1.0)
- PR – Package Release (r0)
- P = "${PN}-${PV}"
- PF = "${PN}-${PV}-${PR}"
- FILE_DIRNAME – Directory for FILE
- FILESPATH = "${FILE_DIRNAME}/${PF}:\
  ${FILE_DIRNAME}/${P}:\
  ${FILE_DIRNAME}/${PN}:\
  ${FILE_DIRNAME}/files:${FILE_DIRNAME}"

## Build Time Metadata

- TOPDIR – The build directory
- TMPDIR = “${TOPDIR}/tmp”
- WORKDIR = ${TMPDIR}/work/${PF}”
- S = “${WORKDIR}/${P}”
- B = “${S}”
- D = “${WORKDIR}/${image}” (Destination dir)
- DEPLOY_DIR = “${TMPDIR}/deploy”
- DEPLOY_DIR_IMAGE = “${DEPLOY_DIR}/images”

# Dependency Metadata

- **Build time package variables**
  - **DEPENDS** – Build time package dependencies
  - **PROVIDES** = “${P} ${PF} ${PN}”
- **Runtime package variables**
  - **RDEPENDS** – Runtime package dependencies
  - **RRECOMMENDS** – Runtime recommended packages
  - **RSUGGESTS** – Runtime suggested packages
  - **RPROVIDES** – Runtime provides
  - **RCONFLICTS** – Runtime package conflicts
  - **RREPLACES** – Runtime package replaces

## Common Metadata

- **Variables you commonly set**
  - **SUMMARY** – Short description of package/recipe
  - **HOMEPAGE** – Upstream web page
  - **LICENSE** – Licenses of included source code
  - **LIC_FILES_CHKSUM** – Checksums of license files  at time of packaging (checked for change by build)
  - **SRC_URI** – URI of source code, patches and extra  files to be used to build packages. Uses different  fetchers based on the URI
  - **FILES** – Files to be included in binary packages

## Examining Recipes: bc

- **Look at 'bc' recipe:**
  - Found in `poky/meta/recipes-extended/bc/bc_1.07.1.bb`
  - Uses `LIC_FILES_CHKSUM` and `SRC_URI` checksums
  - Note the `DEPENDS` build dependency declaration indicating that this package depends on `flex` to build

# Examining Recipes: bc.bb

```
SUMMARY = "Arbitrary precision calculator language"
HOMEPAGE = "http://www.gnu.org/software/bc/bc.html"
DESCRIPTION = "bc is an arbitrary precision numeric processing language.
Syntax is similar to C, but differs in many substantial areas. It supports
interactive execution of statements."

LICENSE = "GPLv3+"
LIC_FILES_CHKSUM = "file://COPYING;md5=d32239bcb673463ab874e80d47fae504 \
                    file://COPYING.LIB;md5=6a6a8e020838b23406c81b19c1d46df6 \
  file://bc/bcdefs.h;endline=17;md5=4295c06df9e833519a342f7b5d43db06 \
  file://dc/dc.h;endline=18;md5=36b8c600b63ee8c3aeade2764f6b2a4b \
  file://lib/number.c;endline=20;md5=cf43068cc88f837731dc53240456cfaf"

SECTION = "base"
DEPENDS = "flex-native"

SRC_URI = "${GNU_MIRROR}/${BPN}/${BP}.tar.gz \
           file://no-gen-libmath.patch \
           file://libmath.h \
           file://0001-dc-fix-exit-code-of-q-command.patch"
SRC_URI[md5sum] = "cda93857418655ea43590736fc3ca9fc"
SRC_URI[sha256sum] =
"62adfca89b0a1c0164c2cdca59ca210c1d44c3ffc46daf9931cf4942664cb02a"

inherit autotools texinfo update-alternatives
…
```

## Building upon bbclass

- **Use inheritance for common design patterns**
- **Provide a class file (.bbclass) which is then inherited by other recipes (.bb files)**

- `inherit autotools`
  - Bitbake will include the `autotools.bbclass` file
  - Found in a ‘`classes`’ directory via the `BBPATH` variable

## Examining Recipes: flac

- **Look at 'flac' recipe**
  - Found in `poky/meta/recipes-multimedia/flac/flac_1.3.3.bb`
- **Inherits from both `autotools` and `gettext`**
  - Customizes `autoconf` configure options (`EXTRA_OECONF`) based on "`TUNE`" features
- **Breaks up output into multiple binary packages**
  - See `PACKAGES` var
  - This recipe produces additional packages with those names, while the `FILES_*` vars specify which files go into these additional packages

# Examining Recipes: flac.bb

```
SUMMARY = "Free Lossless Audio Codec"
DESCRIPTION = "FLAC stands for Free Lossless Audio Codec, a lossless audio
compression format."
HOMEPAGE = "https://xiph.org/flac/"
BUGTRACKER = "http://sourceforge.net/p/flac/bugs/"
SECTION = "libs"
LICENSE = "GFDL-1.2 & GPLv2+ & LGPLv2.1+ & BSD"
LIC_FILES_CHKSUM = …
DEPENDS = "libogg”
SRC_URI = "http://downloads.xiph.org/releases/flac/${BP}.tar.xz"

SRC_URI[md5sum] = "26703ed2858c1fc9ffc05136d13daa69"
SRC_URI[sha256sum] =
"213e82bd716c9de6db2f98bcadbc4c24c7e2efe8c75939a1a84e28539c4e1748"

CVE_PRODUCT = "libflac flac"

inherit autotools gettext
EXTRA_OECONF = "--disable-oggtest \
                --with-ogg-libraries=${STAGING_LIBDIR} \
                --with-ogg-includes=${STAGING_INCDIR} \
                --disable-xmms-plugin \
                --without-libiconv-prefix \
                ac_cv_prog_NASM="" \
                ”
```

(con't next page)

# Examining Recipes: flac.bb

(con't from previous page)

```
EXTRA_OECONF += "${@bb.utils.contains("TUNE_FEATURES", "altivec", " --enable-
altivec", " --disable-altivec", d)}"
EXTRA_OECONF += "${@bb.utils.contains("TUNE_FEATURES", "vsx", " --enable-
vsx", " --disable-vsx", d)}"
EXTRA_OECONF += "${@bb.utils.contains("TUNE_FEATURES", "core2", " --enable-
sse", "", d)}"
EXTRA_OECONF += "${@bb.utils.contains("TUNE_FEATURES", "corei7", " --enable-
sse", "", d)}"

PACKAGES += "libflac libflac++ liboggflac liboggflac++"
FILES_${PN} = "${bindir}/*"
FILES_libflac = "${libdir}/libFLAC.so.*"
FILES_libflac++ = "${libdir}/libFLAC++.so.*"
FILES_liboggflac = "${libdir}/libOggFLAC.so.*"
FILES_liboggflac++ = "${libdir}/libOggFLAC++.so.*"
```

## Grouping Local Metadata

- **Sometimes sharing metadata between recipes is easier via an include file**

- `include file.inc`
  - Will include .inc file if found via `BBPATH`
  - Can also specify an absolute path
  - If not found, will continue without an error

- `require file.inc`
  - Same as an include
  - Fails with an error if not found

## Examining Recipes: dhcp

- **Look at 'dhcp' recipe(s):**
  - Found in `poky/meta/recipes-connectivity/dhcp/dhcp_4.4.2.bb`
- **Splits recipe into common .inc file to share common metadata between multiple recipes**

## Examining Recipes: dhcp.bb

```
require dhcp.inc

SRC_URI += "file://0001-define-macro-_PATH_DHCPD_CONF-and-
_PATH_DHCLIENT_CON.patch \
            file://0002-dhclient-dbus.patch \
            file://0003-link-with-lcrypto.patch \
            file://0004-Fix-out-of-tree-builds.patch \
            file://0005-dhcp-client-fix-invoke-dhclient-script-failed-on-
Rea.patch \
            file://0007-Add-configure-argument-to-make-the-libxml2-
dependenc.patch \
            file://0009-remove-dhclient-script-bash-dependency.patch \
            file://0012-dhcp-correct-the-intention-for-xml2-lib-search.patch
\
            file://0013-fixup_use_libbind.patch \
            file://0001-workaround-busybox-limitation-in-linux-dhclient-
script.patch \
            file://CVE-2021-25217.patch \
            file://CVE-2022-2928.patch \
            file://CVE-2022-2929.patch \
"

SRC_URI[md5sum] = "2afdaf8498dc1edaf3012efdd589b3e1"
SRC_URI[sha256sum] =
"1a7ccd64a16e5e68f7b5e0f527fd07240a2892ea53fe245620f4f5f607004521"
…
```

Some useful tools to help guide you when something goes wrong

## TROUBLESHOOTING

## Bitbake Environment

- **Each recipe has its own environment which contains all the variables and methods required to build that recipe**
- **You've seen some of the variables already**
  - `DESCRIPTION`, `SRC_URI`, `LICENSE`, `S`, `LIC_FILES_CHKSUM`, `do_compile()`, `do_install()`
- **Example**
  - `S = "${WORKDIR}"`
  - What does this mean?

## Examine a Recipe's Environment

- **To view a recipe's envrionment**

```
$ bitbake -e myrecipe
```

- **Where is the source code for this recipe?**

```
$ bitbake -e virtual/kernel | grep "^S="
S="${HOME}/yocto/build/tmp/work-
shared/qemuarm/kernel-source"
```

- **What file was used in building this  recipe?**

```
$ bitbake -e netbase | grep "^FILE="
FILE="${HOME}/yocto/poky/meta/recipes-
core/netbase/netbase_5.3.bb"
```

## Examine a Recipe's Environment

- **What is this recipe's full version string?**

```
$ bitbake -e netbase | grep "^PF="
PF="netbase-1_5.3-r0"
```

- **Where is this recipe's BUILD directory?**

```
$ bitbake -e virtual/kernel | grep "^B="
B="${HOME}/yocto/build/tmp/work/qemuarm-poky-linux-\
gnueabi/linux-yocto/3.19.2+gitAUTOINC+9e70b482d3\
_473e2f3788-r0/linux-qemuarm-standard-build"
```

- **What packages were produced by this recipe?**

```
$ bitbake -e virtual/kernel | grep "^PACKAGES="
PACKAGES="kernel kernel-base kernel-vmlinux kernel-
image kernel-dev kernel-modules kernel-devicetree"
```

## BitBake Log Files

- **Every build produces lots of log output for diagnostics  and error chasing**
  - Look in `build/tmp/log/cooker/<machine>`

```
$ cat tmp/log/cooker/qemuarm/console-latest.log | grep 'NOTE:.*task.*Started'

NOTE: recipe hello-1.0.0-r0: task do_fetch: Started
NOTE: recipe hello-1.0.0-r0: task do_unpack: Started
NOTE: recipe hello-1.0.0-r0: task do_patch: Started
NOTE: recipe hello-1.0.0-r0: task do_configure: Started
NOTE: recipe hello-1.0.0-r0: task do_populate_lic: Started
NOTE: recipe hello-1.0.0-r0: task do_compile: Started
NOTE: recipe hello-1.0.0-r0: task do_install: Started
NOTE: recipe hello-1.0.0-r0: task do_populate_sysroot: Started
NOTE: recipe hello-1.0.0-r0: task do_package: Started
NOTE: recipe hello-1.0.0-r0: task do_packagedata: Started
NOTE: recipe hello-1.0.0-r0: task do_package_write_rpm: Started
NOTE: recipe hello-1.0.0-r0: task do_package_qa: Started
NOTE: recipe hello-1.0.0-r0: task do_rootfs: Started
…
```

## BitBake Per-Recipe Log Files

- **Every recipe produces lots of log output for  diagnostics and debugging**
- **Use the environment to find the log files for a  given recipe:**

```
$ bitbake -e hello | grep “^T=”
T="${HOME}yocto/build/tmp/work/armv5e-poky-linux-
gnueabi/hello/1.0.0-r0/temp"
```

- **Each task that runs for a recipe produces “log”  and “run” files in `${WORKDIR}/temp`**
  - These files contain the  output of the respective  tasks for each recipe

```
$ find . -type l -name 'log.*’
```

  - These contain the  commands executed which  produce the build results

```
$ find . -type l -name 'run.*'
```