Image

[What is an image 2](#_Toc146965846)

[Creating custom images 2](#_Toc146965847)

[Two ways 2](#_Toc146965848)

[Package group 2](#_Toc146965849)

[Creating an image from scratch 3](#_Toc146965850)

[Reusing an existing image 3](#_Toc146965851)

[Customizing Images Using Custom IMAGE\_FEATURES and EXTRA\_IMAGE\_FEATURES 4](#_Toc146965852)

[What's the difference between IMAGE\_FEATURES and EXTRA\_IMAGE\_FEATURES 4](#_Toc146965853)

[How it works? 4](#_Toc146965854)

[Example of IMAGE\_FEATURES 4](#_Toc146965855)

[debug-tweaks 5](#_Toc146965856)

[Read-Only Root Filesystem 5](#_Toc146965857)

[Why do we need read-only rootfs 5](#_Toc146965858)

[How to do it? 5](#_Toc146965859)

[Boot Splash screen 5](#_Toc146965860)

[Some other Features 5](#_Toc146965861)

[IMAGE\_LINGUAS 6](#_Toc146965862)

[IMAGE\_FSTYPES 6](#_Toc146965863)

[Types supported 6](#_Toc146965864)

[Creating your own image type 6](#_Toc146965865)

[IMAGE\_NAME 7](#_Toc146965866)

[IMAGE\_MANIFEST 7](#_Toc146965867)

[Challenge 7](#_Toc146965868)

# What is an image

* Image is a top level recipe. (It inherits an image.bbclass)
* Building an image creates an entire Linux distribution from source
* Compiler, tools, libraries
* BSP: Bootloader, Kernel
* Root filesystem:
  + Base OS
  + services
  + Applications
  + etc

# Creating custom images

* You often need to create your own Image recipe in order to add new packages or functionality

# Two ways

* creating an image from scratch
* extend an existing recipe (preferable)

# Package group

* A package group is a set of packages that can be included on any image.
* A package group can contain a set of packages.
* Using the packagegroup name in IMAGE\_INSTALL variable install all the packages defined by the package group into the root file system of your target image.
* There are many package groups. There are present in subdirectories named "packagegroups"
  + $ find . -name 'packagegroups'
* They are recipe files(.bb) and starts with packagegroup-
* For example,
  + packagegroup-core-boot: Provides the minimum set of packages necessary to create a bootable image with console.

# Creating an image from scratch

* The simplest way is to inherit the core-image bbclass, as it provides a set of image features that can be used very easily
* inherit core-image
* Which tells us that the definition of what actually gets installed is defined in the core-image.bbclass.
* Image recipes set IMAGE\_INSTALL to specify the packages to install into an image through image.bbclass.
* Create an images directory
  + $ mkdir -p recipes-examples/images
* Create the image recipe
  + $ vi recipes-examples/images/lwl-image.bb
  + SUMMARY = "A small boot image for LWL learners"
  + LICENSE = "MIT"
* inherit core-image
  + # Core files for basic console boot
  + IMAGE\_INSTALL = "packagegroup-core-boot"
  + IMAGE\_ROOTFS\_SIZE ?= "8192"
  + #Add our needed applications
  + IMAGE\_INSTALL += "usbutils"

# Reusing an existing image

* When an image mostly fits our needs and we need to do minor adjustments on it,it is very convenient to reuse its code
* This makes code maintenance easier and highlights the functional differences
* For example, if we want to include an application (lsusb)
* Create another recipe:
  + $ vim recipes-examples/images/lwl-image-reuse.bb
  + require recipes-core/images/core-image-minimal.bb
  + IMAGE\_INSTALL\_append = " usbutils"

# Customizing Images Using Custom IMAGE\_FEATURES and EXTRA\_IMAGE\_FEATURES

* Another method for customizing your image is to enable or disable high-level image features by using the IMAGE\_FEATURES and EXTRA\_IMAGE\_FEATURES variables
* IMAGE\_FEATURES/EXTRA\_IMAGE\_FEATURES is made to enable special features for your image, such as empty password for root, debug image, special packages, x11, splash, ssh-server

# What's the difference between IMAGE\_FEATURES and EXTRA\_IMAGE\_FEATURES

* Best practice is to
  + Use IMAGE\_FEATURES from a recipe
  + Use EXTRA\_IMAGE\_FEATURES from local.conf

# How it works?

* To understand how these features work, the best reference is meta/classes/core-image.bbclass
* This class lists out the available IMAGE\_FEATURES of which most map to package groups while some, such as debug-tweaks and read-only-rootfs, resolve as general configuration settings
* In summary, the file looks at the contents of the IMAGE\_FEATURES variable and then maps or configures the feature accordingly
* Based on this information, the build system automatically adds the appropriate packages or configurations to the IMAGE\_INSTALL variable.

# Example of IMAGE\_FEATURES

* To illustrate how you can use these variables to modify your image, consider an example that selects the SSH server.
* The Yocto Project ships with two SSH servers you can use with your images: Dropbear and OpenSSH.
* OpenSSH is a well-known standard SSH server implementation
* Dropbear is a minimal SSH server appropriate for resource-constrained environments
* By default, the core-image-sato image is configured to use Dropbear. The core-image-full-cmdline and core-image-lsb images both include OpenSSH.
* The core-image-minimal image does not contain an SSH server.

# debug-tweaks

* In the default state, local.conf file has EXTRA\_IMAGE\_FEATURES set to "debug-tweaks"
* debug-tweaks features enable password-less login for the root user
* Advantage: makes logging in for debugging or inspection easy during development
* Disadvantage: anyone can easily log in during production.
* So, you need to remove the 'debug-tweaks' feature from production image

# Read-Only Root Filesystem

## Why do we need read-only rootfs

* + Reduce wear on flash memory
  + Eliminate system file corruption

## How to do it?

* To create the read-only root filesystem, simply add the "read-only-rootfs" feature to your image.
* IMAGE\_FEATURES = "read-only-rootfs" in your recipe
  + or
* EXTRA\_IMAGE\_FEATURES += "read-only-rootfs" in local.conf

# Boot Splash screen

* IMAGE\_FEATURES += "splash"
  + or
* EXTRA\_IMAGE\_FEATURES += "splash"

# Some other Features

* tools-debug: Installs debugging tools such as strace and gdb.
* tools-sdk: Installs a full SDK that runs on the device.

# IMAGE\_LINGUAS

* Specifies the list of locales to install into the image during the root filesystem construction process
* IMAGE\_LINGUAS = "zh-cn"
* Inside qemu image
  + $ locale -a

# IMAGE\_FSTYPES

* The IMAGE\_FSTYPES variable determines the root filesystem image type
* If more than one format is specified, one image per format will be generated
* Image formats instructions are delivered in Poky: meta/classes/image\_types.bbclass
  + $ bitbake -e <image\_name> | grep ^IMAGE\_FSTYPES=

## Types supported

Btrfs *container cpio cpio.gz cpio.lz4 cpio.lzma cpio.xz cramfs*

*elf ext2 ext2.bz2 ext2.gz ext2.lzma ext3 ext3.gz ext4 ext4.gz*

*f2fs ddimg iso jffs2 jffs2.sum multiubi squashfs squashfs-lz4*

*squashfs-zo squashfs-xz tar tar.bz2 tar.gz tar.lz4 tar.xz ubi ubifs*

*wic ic.bz2 wic.gz wic.lzma*

# Creating your own image type

* If you have a particular layout on your storage (for example bootloader location on an SD card), you may want to create your own image type
* This is done through a class that inherits from image\_types
* It has to define a function named IMAGE\_CMD\_<type>
* Example: sdcard\_image-rpi.bbclass in meta-raspberrypi

# IMAGE\_NAME

* The name of the output image files minus the extension
* This variable is derived using the IMAGE\_BASENAME, MACHINE, and DATETIME variables
* IMAGE\_NAME = "${IMAGE\_BASENAME}-${MACHINE}-${DATETIME}"

# IMAGE\_MANIFEST

* The manifest file for the image
* This file lists all the installed packages that make up the image.
* The file contains package information on a line-per-package basis as follows:
  + packagename packagearch version
* The image class defines the manifest file as follows:
  + IMAGE\_MANIFEST = "${DEPLOY\_DIR\_IMAGE}/${IMAGE\_NAME}.rootfs.manifest"

# Challenge

Try other IMAGE\_FEATURES