Use Case of the Commands

1. source oe-init-build-env

Use Case: Setting up the Yocto Build Environment

This command initializes the Yocto build environment, preparing the system for running bitbake commands.

It creates a build directory (default: build/).

It sets up the necessary environment variables for BitBake, Yocto recipes, and layers.

When to Use?

Before running bitbake, this must be executed in every new terminal session.

When switching between different build directories.

When working on a new Yocto project.

✅ Example Workflow:

cd poky

source oe-init-build-env

🔹 After this, you can run BitBake commands.

2. bitbake core-image-minimal

Use Case: Building a Minimal Embedded Linux Image

This command tells BitBake to compile and package a minimal Linux image.

It fetches, builds, and assembles the required software components.

The generated image can be flashed onto an embedded board (e.g., Raspberry Pi, BeagleBone, S32G2).

When to Use?

When you need a lightweight, bootable Linux OS for embedded systems.

When developing a custom embedded OS, starting from a minimal base.

When testing Yocto builds before adding extra software.

✅ Example Workflow:

bitbake core-image-minimal

🔹 After the build completes, the image is available in:

build/tmp/deploy/images/<machine>/core-image-minimal-<machine>.wic

🔹 Flash to an SD card or eMMC:

sudo dd if=core-image-minimal-<machine>.wic of=/dev/sdX bs=4M status=progress

🔹 Boot the embedded board using this image.

Understanding local.conf in Yocto

The local.conf file is a configuration file in Yocto that controls various build settings, target machine, image types, and performance optimizations.

📌 Where is local.conf Located?

After you run:

source oe-init-build-env

Yocto creates a build directory (build/), and inside it, you will find:

build/

├── conf/

│ ├── bblayers.conf

│ ├── local.conf 👈 \*\*This is the file we are talking about!\*\*

│ ├── site.conf (optional)

│ ├── auto.conf (optional)

└── ...

👉 File path: build/conf/local.conf

📌 What is the Purpose of local.conf?

Defines target hardware (machine type).

Configures build performance settings (parallel jobs, RAM usage).

Specifies output image format (e.g., .wic, .tar.gz).

Manages package management system (DEB, RPM, IPK).

Controls extra features like SSH, debugging tools, or logging.

Specifies mirror locations for fetching source files.

📌 Key Sections of local.conf

Here are the important variables inside local.conf:

1️⃣ Target Machine (Board Configuration)

MACHINE ??= "qemuarm"

Step-by-Step Guide to Mount and Unmount a Device in Linux

When working with external storage devices (USB drives, SD cards, eMMC partitions, etc.), you need to mount them before accessing files and unmount them before removing them safely.

🔹 Step 1: Identify the Device

Before mounting, you need to find the device name.

🔸 List all storage devices:

lsblk

🔸 Check partitions and device files:

fdisk -l

🔸 View detailed disk information:

blkid

🔹 Example Output:

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 500G 0 disk

├─sda1 8:1 0 100G 0 part /

├─sda2 8:2 0 400G 0 part /home

sdb 8:16 1 32G 0 disk

└─sdb1 8:17 1 32G 0 part

Here, sdb1 is the external device (USB/eMMC/SD card partition).

🔹 Step 2: Create a Mount Point

A mount point is a directory where the device will be attached.

🔸 Create a mount point (if not already created):

sudo mkdir -p /mnt/mydevice

This creates a folder /mnt/mydevice, where your device’s filesystem will be accessible.

🔹 Step 3: Mount the Device

🔸 Mount the device to /mnt/mydevice:

sudo mount /dev/sdb1 /mnt/mydevice

✅ Now, the device is accessible under /mnt/mydevice.

🔸 Verify that the device is mounted:

df -h

🔹 Example Output:

Filesystem Size Used Avail Use% Mounted on

/dev/sda1 100G 60G 40G 60% /

/dev/sdb1 32G 10G 22G 30% /mnt/mydevice

This confirms that /dev/sdb1 is successfully mounted.

🔹 Step 4: Access Files on the Mounted Device

🔸 List the files:

ls /mnt/mydevice

🔸 Copy files from the device:

cp /mnt/mydevice/example.txt ~/Documents/

🔸 Write files to the device:

echo "Hello Yocto!" > /mnt/mydevice/newfile.txt

🔹 Step 5: Unmount the Device

Before removing the device, you must unmount it to prevent data corruption.

🔸 Unmount the device:

sudo umount /mnt/mydevice

✅ The device is now safely unmounted.

🔸 Verify the unmount:

lsblk

If sdb1 no longer shows a MOUNTPOINT, it is successfully unmounted.

🔹 Step 6: Safely Remove the Device

If using a USB drive, safely eject it:

sudo eject /dev/sdb

📌 Summary of Mounting & Unmounting

Step Command Purpose

1. Identify the device lsblk, fdisk -l Find the device name (/dev/sdb1)

2. Create a mount point mkdir -p /mnt/mydevice Prepare a folder for mounting

3. Mount the device mount /dev/sdb1 /mnt/mydevice Attach the device to the mount point

4. Access the device ls /mnt/mydevice View and manipulate files

5. Unmount the device umount /mnt/mydevice Safely detach the device

6. Eject (USB only) eject /dev/sdb Safely remove a USB

🔥 Bonus: Mounting with Specific Filesystem

If your device uses a specific filesystem (e.g., ext4, vfat, NTFS):

sudo mount -t ext4 /dev/sdb1 /mnt/mydevice

Replace ext4 with:

vfat (for FAT32 USBs)

ntfs (for Windows drives)

Would you like help automating this process with a script? 😊🚀

You said:

can balenaetcher help in doing mounting?

ChatGPT said:

ChatGPT

Can Balena Etcher Help in Mounting?

🚀 No, Balena Etcher does not perform mounting. It is used for flashing OS images to storage devices (USB drives, SD cards, eMMC, etc.), not for mounting file systems.

📌 What Does Balena Etcher Do?

Flashes .iso, .img, .wic, or .bin images to SD cards, USBs, or eMMC.

Used for creating bootable drives (e.g., flashing Yocto Linux to a board).

Erases & formats the target device before writing.

Does NOT mount the drive—instead, it writes a raw disk image.

📌 When Should You Use Balena Etcher?

✔️ Flashing a Yocto .wic image to an SD card:

balenaEtcher → Select Image → Select SD Card → Flash! ✅

✔️ Creating a bootable USB drive for an embedded board. ✔️ Flashing OS images for Raspberry Pi, NXP S32G, BeagleBone, etc.

📌 When Should You Manually Mount a Device?

You need to manually mount a device if you: ✔️ Want to access or modify files on an SD card or eMMC.

✔️ Need to transfer logs, firmware, or configurations between your PC and the embedded system.

✔️ Use a non-flashing method for copying files.

📌 How to Mount After Using Balena Etcher?

1️⃣ Insert the flashed SD card / USB drive.

2️⃣ Check available partitions:

lsblk

3️⃣ Mount the partition manually:

sudo mount /dev/sdb1 /mnt/mydevice

4️⃣ Access files inside /mnt/mydevice.

📌 Key Differences

Feature Balena Etcher Manual Mounting

Writes OS Images? ✅ Yes ❌ No

Formats Drive? ✅ Yes ❌ No

Mounts Drive? ❌ No ✅ Yes

Allows File Access? ❌ No ✅ Yes

Use Case Flashing OS images (e.g., Yocto) Accessing/modifying storage

🔥 Final Answer:

❌ Balena Etcher does not mount a device.

✅ It only writes OS images to storage devices.

✅ To access the drive’s files, you must manually mount it.

Defines the target hardware platform for the build.

Example values:

"s32g274a" (for NXP S32G boards)

"qemux86-64" (for QEMU emulator)

"raspberrypi4" (for Raspberry Pi 4)

✅ If you are building for a specific board, you must set this!

2️⃣ Parallel Build Optimization

BB\_NUMBER\_THREADS ?= "8"

PARALLEL\_MAKE ?= "-j8"

BB\_NUMBER\_THREADS → Number of parallel BitBake tasks.

PARALLEL\_MAKE → Number of parallel compilation threads.

✅ Set this based on your CPU cores (e.g., 8 for an 8-core CPU).

3️⃣ Output Image Type

IMAGE\_FSTYPES = "wic.gz ext4 tar.gz"

Defines the output formats for the built image.

Example formats:

"wic.gz" → Bootable SD card image

"ext4" → Filesystem image

"tar.gz" → Root filesystem archive

✅ Change this based on your deployment needs.

4️⃣ Package Management System

PACKAGE\_CLASSES ?= "package\_rpm"

Defines the package format used inside the image.

Options:

"package\_ipk" (Lightweight, used in OpenEmbedded)

"package\_deb" (Used in Debian-based systems)

"package\_rpm" (Used in Fedora/Red Hat systems)

✅ Choose the package format based on your target system.

5️⃣ Source Code Download Location

DL\_DIR ?= "${TOPDIR}/downloads"

SSTATE\_DIR ?= "${TOPDIR}/sstate-cache"

DL\_DIR → Directory where Yocto stores downloaded sources.

SSTATE\_DIR → Directory for cached builds (used for faster recompilation).

✅ Reusing sstate-cache speeds up incremental builds!

6️⃣ Extra Debugging and Development Tools

EXTRA\_IMAGE\_FEATURES = "debug-tweaks ssh-server-dropbear"

Adds extra tools to the built image:

"debug-tweaks" → Enables root login without a password.

"ssh-server-dropbear" → Installs a lightweight SSH server.

✅ Useful for debugging an embedded system over SSH.

7️⃣ Root Filesystem Size and Compression

IMAGE\_ROOTFS\_SIZE = "512000"

IMAGE\_ROOTFS\_EXTRA\_SPACE = "102400"

IMAGE\_ROOTFS\_SIZE → Sets the root filesystem size (in KB).

IMAGE\_ROOTFS\_EXTRA\_SPACE → Adds extra space to avoid running out of storage.

✅ Increase this if your image is too large!

8️⃣ Enabling or Disabling Package Build

DISTRO\_FEATURES\_remove = "x11 wayland"

Removes X11 (GUI system) and Wayland to reduce image size.

✅ Use this if you only need a command-line environment.

📌 Example local.conf File

Here is a simplified local.conf setup for an NXP S32G board:

MACHINE ??= "s32g274a"

BB\_NUMBER\_THREADS ?= "8"

PARALLEL\_MAKE ?= "-j8"

IMAGE\_FSTYPES = "wic.gz ext4 tar.gz"

PACKAGE\_CLASSES ?= "package\_ipk"

DL\_DIR ?= "${TOPDIR}/downloads"

SSTATE\_DIR ?= "${TOPDIR}/sstate-cache"

EXTRA\_IMAGE\_FEATURES = "debug-tweaks ssh-server-dropbear"

IMAGE\_ROOTFS\_SIZE = "1024000"

IMAGE\_ROOTFS\_EXTRA\_SPACE = "204800"

DISTRO\_FEATURES\_remove = "x11 wayland"

📌 Summary

Variable Purpose

MACHINE Selects the target board (e.g., "s32g274a", "raspberrypi4")

BB\_NUMBER\_THREADS, PARALLEL\_MAKE Controls build parallelization (set based on CPU cores)

IMAGE\_FSTYPES Defines the output image format (e.g., wic.gz, ext4, tar.gz)

PACKAGE\_CLASSES Selects package management system (ipk, deb, rpm)

DL\_DIR, SSTATE\_DIR Defines source and cache directories for faster builds

EXTRA\_IMAGE\_FEATURES Enables SSH, debugging, extra utilities

IMAGE\_ROOTFS\_SIZE Controls root filesystem size

DISTRO\_FEATURES\_remove Strips unwanted GUI features (e.g., "x11 wayland")

🚀 What Can You Customize in local.conf?

Change MACHINE to match your board.

Adjust IMAGE\_FSTYPES to generate .wic, .ext4, or other formats.

Enable ssh-server-openssh instead of dropbear for secure remote login.

Increase IMAGE\_ROOTFS\_SIZE if your image runs out of space.

Set BB\_NUMBER\_THREADS to improve build speed.

what is mounting and unmounting of a device in very simple terms

ChatGPT said:

ChatGPT

🖥️ What is Mounting and Unmounting? (Simple Explanation)

✅ Mounting → Making a storage device accessible on your computer.

✅ Unmounting → Safely disconnecting the device so no data is lost.

📌 Think of It Like Plugging a USB Drive into Your Computer

Mounting is like plugging in a USB drive and seeing it appear in File Explorer (Windows) or Finder (Mac).

Unmounting is like safely ejecting the USB before removing it.

📌 Example in Linux

1️⃣ Mount (Make the device usable)

sudo mount /dev/sdb1 /mnt/myusb

🔹 Now, you can open /mnt/myusb and access files.

2️⃣ Unmount (Safely remove the device)

sudo umount /mnt/myusb

🔹 Now, it's safe to unplug the device.

📌 Why is Unmounting Important?

❌ If you remove a USB/eMMC/SD card without unmounting, data can be corrupted or lost.

✅ Unmounting ensures that all file operations are completed before removal.

🚀 Final Summary

Mounting = Attaching a device to access its files.

Unmounting = Safely removing it to avoid data loss.

Would you like an automatic script for mounting and unmounting? 😊

You said:

but say when i connect a usb, i anyway see that its available, why the need to mount it

ChatGPT said:

ChatGPT

🖥️ Why Do We Need to Mount a Device If It Already Appears?

When you plug in a USB drive, SD card, or external hard disk, your operating system automatically mounts it for you. But in some cases, manual mounting is necessary. Let’s break it down.

📌 Why Does a USB Drive Automatically Appear?

Modern operating systems (Windows, macOS, and Linux with GUI) have an auto-mounting feature.

When you insert a USB drive:

The OS detects the device.

It assigns a mount point automatically.

It displays it in File Explorer (Windows) or Finder (Mac).

✅ This means you don’t have to manually mount it in most cases.

📌 When Do You Need to Manually Mount a Device?

🚀 1. When Using a Headless (No GUI) Linux System

If you're working with an embedded board (e.g., NXP S32G, Raspberry Pi), there is no GUI auto-mount.

You must manually mount the USB or eMMC:

sudo mount /dev/sdb1 /mnt/myusb

🚀 2. When the Filesystem Is Not Recognized

If the USB drive uses an unsupported filesystem (e.g., ext4, NTFS on some systems), it won’t auto-mount.

You must specify the filesystem manually:

sudo mount -t ext4 /dev/sdb1 /mnt/myusb

🚀 3. When Using Multiple Partitions on a Single Device

Some USB devices or SD cards have multiple partitions, and the OS may not mount all of them.

You must manually find and mount the partition you need.

🚀 4. When Using a Raw Disk Image (.img or .wic)

If you flash a Yocto image using Balena Etcher, it won’t automatically appear.

You need to manually mount the root filesystem:

sudo mount /dev/sdb2 /mnt/rootfs

🚀 5. When You Need Custom Mounting Options

If you want to mount as read-only:

sudo mount -o ro /dev/sdb1 /mnt/readonly

If you want to enable write permissions for all users:

sudo mount -o uid=1000,gid=1000 /dev/sdb1 /mnt/myusb

📌 Why Is Unmounting Still Necessary?

Even if a USB automatically mounts, you still need to unmount it before removing it:

sudo umount /mnt/myusb

✅ Prevents data loss

✅ Ensures all files are saved properly

✅ Avoids corruption in eMMC/SD cards in embedded systems

🚀 Final Answer

Yes, your USB auto-mounts in modern OSes because they handle it in the background.

But in some cases (embedded systems, unsupported filesystems, multiple partitions), you still need to manually mount it.

Unmounting is always needed to prevent corruption.

📌 What is dmesg Used For?

The dmesg (display message) command is used to view system logs related to the Linux kernel. It primarily helps in diagnosing hardware and system issues.

🔹 What Does dmesg Show?

Kernel boot logs (hardware initialization)

USB device detection (when you plug in a USB)

SD card or eMMC logs (when mounted or fails to mount)

Driver errors (network, storage, graphics issues)

System crashes and debugging info

🔹 Basic Usage

1️⃣ Show latest kernel logs:

dmesg

2️⃣ Follow real-time logs (like tail -f):

dmesg -w

3️⃣ Filter logs by keyword (e.g., USB detection):

dmesg | grep usb

4️⃣ Show logs with timestamps:

dmesg -T

🔹 Use Cases

✅ Check if a USB drive or SD card was detected

dmesg | grep sd

✅ Debug hardware issues (e.g., Ethernet, CAN, SPI, GPIO)

dmesg | grep eth

✅ Analyze boot issues

dmesg | less

🚀 Final Summary

dmesg displays system logs from the kernel.

Helps debug hardware, drivers, and boot issues.

Useful for checking USB, SD card, and storage mounts.

📌 uname Command in Linux

The uname command is used to display system information, including the kernel version, OS, and architecture.

🔹 uname -r → Show Kernel Version

uname -r

✅ Example Output:

5.15.0-84-generic

✔️ Use Case:

Helps check the Linux kernel version running on your system.

Useful when debugging driver issues or checking compatibility for kernel modules.

🔹 uname -a → Show All System Information

uname -a

✅ Example Output:

Linux mydevice 5.15.0-84-generic #93-Ubuntu SMP Mon Sep 11 15:23:13 UTC 2023 x86\_64 x86\_64 x86\_64 GNU/Linux

✔️ Breakdown of Output:

Field Meaning

Linux Operating system type

mydevice Hostname of the system

5.15.0-84-generic Kernel version

#93-Ubuntu SMP Mon Sep 11 15:23:13 UTC 2023 Kernel build details

x86\_64 System architecture (64-bit)

GNU/Linux OS name

✔️ Use Case:

Useful for checking kernel version, system type, and architecture.

Helps when compiling kernel modules or debugging system issues.

What is picocom Used For?

picocom is a lightweight serial terminal emulator used to communicate with serial devices (UART, RS-232, USB-to-serial adapters, embedded boards, etc.).

🔹 Why Use picocom?

✔️ Connect to serial consoles of embedded boards (e.g., NXP S32G, Raspberry Pi, BeagleBone).

✔️ Monitor boot logs over a serial interface.

✔️ Send commands to a device via UART.

✔️ Debug firmware/kernel logs from embedded systems.

🔹 Basic picocom Usage

1️⃣ Connect to a Serial Port

picocom -b 115200 /dev/ttyUSB0

🔹 Explanation:

-b 115200 → Sets the baud rate to 115200 bps (common for embedded boards).

/dev/ttyUSB0 → Serial device (Check with ls /dev/tty\*).

2️⃣ Exit picocom Properly

Press CTRL + A, then CTRL + X to exit.

🔹 Common Use Cases

✅ Check serial connection to an embedded board

picocom -b 115200 /dev/ttyS0

✅ Monitor U-Boot/kernel logs over UART

picocom -b 115200 /dev/ttyUSB1

✅ Send AT commands to a modem (e.g., GSM, LTE)

picocom -b 9600 /dev/ttyUSB2

🚀 Final Summary

Command Purpose

picocom -b 115200 /dev/ttyUSB0 Open a serial connection at 115200 baud

CTRL + A, CTRL + X Exit picocom safely

picocom -b 9600 /dev/ttyS0 Connect to a serial modem

Things learnt today

* Dmesg is used to load the system info, like cpuinfo, or kernel info, which new device was mounted, u can do dmesg | grep “usb” to check for usb logs
* So mounting means able to use an external device on your system, like in windows when you manually connect a usb device like a pen drive you see the pop up, the ur pc, this is because your system automatically does it for u, but in certain setups like mebedded os, or linux systems as well, u have to manually mount it, for that we use sudo mount command
* Like how we flash our binaires using trace32 we have tools like Balena Etcher
* Local.conf file we can use to set which device we are using, like do we need the linux os on s32g2 chip or beaglebone or raspberry pie, u specify the machine name there
* We can provide which file system u want to use, the root file system size that should be there, few more things which I don’t remember now
* So for building yocto, there are very few steps, first clone poky, take kirkstone or any feature branch, then run source oe-init-buildenv, this will set the build env, and u will be able to use bitbake, then run bitbake <recipe-file-name> and one more cool thing, u don’t have to give the path of the bb file, bitbake will find it for you