Building Kernels

This page details the process of building custom kernels for Android devices. These instructions guide you through the process of selecting the right sources, building the kernel, and embedding the results into a system image built from the Android Open Source Project (AOSP).

You can acquire more recent kernel sources by using Repo; build them without further configuration by running `build/build.sh` from the root of your source checkout.

Note: The root of the kernel source checkout contains `build/build.sh`. The Android tree contains only prebuilt kernel binaries. The kernel trees contain the kernel sources and all tools to build the kernels, including this script.

To build older kernels or kernels not listed on this page, refer to the instructions on how to build legacy kernels.

[Downloading sources and build tools]

For recent kernels, use `repo` to download the sources, toolchain, and build scripts. Some kernels (for example, the Pixel 3 kernels) require sources from multiple git repositories, while others (for example, the common kernels) require only a single source. Using the `repo` approach ensures a correct source directory setup.

Download the sources for the appropriate branch:

```
mkdir android-kernel && cd android-kernel
```

```
repo init -u https://android.googlesource.com/kernel/manifest -b BRANCH
```

```
repo sync
```

The following table lists the `BRANCH` names for kernels available through this method.

Device	Binary path in AOSP tree	Repo branches
Pixel 6 (oriole)\ Pixel 6 Pro (raven)	device/google/raviole-kernel	android-gs-raviole-5.10-android12L
Pixel 5a (barbet)	device/google/barbet-kernel	android-msm-barbet-4.19-android12L
Pixel 5 (redfin)\ Pixel 4a (5G) (bramble)	device/google/redbull-kernel	android-msm-redbull-4.19-android12L
Pixel 4a (sunfish)	device/google/sunfish-kernel	android-msm-sunfish-4.14-android12L
Pixel 4 (flame)\ Pixel 4 XL (coral)	device/google/coral-kernel	android-msm-coral-4.14-android12L

```
  Pixel 3a (sargo)\       device/google/bonito-kernel       android-msm-bonito-4.9-
android12L
  Pixel 3a XL (bonito)

  Pixel 3 (blueline)\     device/google/crosshatch-kernel   android-msm-crosshatch-
4.9-android12
  Pixel 3 XL (crosshatch)

  Pixel 2 (walleye)\      device/google/wahoo-kernel        android-msm-wahoo-4.4-
android10-qpr3
  Pixel 2 XL (taimen)

  Pixel (sailfish)\       device/google/marlin-kernel       android-msm-marlin-3.18-
pie-qpr2
  Pixel XL (marlin)

  Hikey960                device/linaro/hikey-kernel        hikey-linaro-android-4.14\
                                                            hikey-linaro-android-4.19\
                                                            common-android12-5.4

  Beagle x15              device/ti/beagle\_x15-kernel      omap-beagle-x15-android-
4.14\
                                                            omap-beagle-x15-android-
4.19

  Android Common Kernel   N/A                               common-android-4.4\
                                                            common-android-4.9\
                                                            common-android-4.14\
                                                            common-android-4.19\
                                                            common-android-4.19-
stable\
                                                            common-android11-5.4\
                                                            common-android12-5.4\
                                                            common-android12-5.10\
                                                            common-android-mainline
  ------------------------------------------------------------------------------------
----------
```

Note: You can switch among different branches within one Repo checkout. The common kernel manifests (and most others) define the kernel git repository to be cloned fully (not shallow), which enables fast switching among them. Switching to a different branch is similar to initializing a branch; the `-u` parameter is optional. For example, to switch to `common-android-mainline` from your existing Repo checkout, run: `$ repo init -b common-android-mainline && repo sync`.

[Building the kernel]

Then build the kernel with this:

```
build/build.sh
```

Note: Common kernels are generic, customizable kernels and therefore don't define a default configuration. See Customize the kernel build to find out how to specify the build configuration for common kernels. For example, to

build the GKI kernel for the aarch64 platform, run:
`$ BUILD_CONFIG=common/build.config.gki.aarch64 build/build.sh`

The kernel binary, modules, and corresponding image are located in the `out/BRANCH/dist` directory.

[Building the GKI modules]

Android 11 introduced GKI, which separates the kernel into a Google-maintained kernel image and vendor maintained-modules, which are built separately.

This example shows a kernel image configuration:

```
BUILD_CONFIG=common/build.config.gki.x86_64 build/build.sh
```

This example shows a module configuration (Cuttlefish and Emulator):

```
BUILD_CONFIG=common-modules/virtual-device/build.config.cuttlefish.x86_64
build/build.sh
```

In Android 12 Cuttlefish and Goldfish converge, so they share the same kernel: `virtual_device`. To build that kernel's modules, use this build configuration:

```
BUILD_CONFIG=common-modules/virtual-device/build.config.virtual_device.x86_64
build/build.sh
```

[Running the kernel]

There are multiple ways to run a custom-built kernel. The following are known ways suitable for various development scenarios.

[Embedding into the Android image build]

Copy `Image.lz4-dtb` to the respective kernel binary location within the AOSP tree and rebuild the boot image.

Alternatively, define the `TARGET_PREBUILT_KERNEL` variable while using `make bootimage` (or any other `make` command line that builds a boot image). This variable is supported by all devices as it's set up via `device/common/populate-new-device.sh`. For example:

```
export TARGET_PREBUILT_KERNEL=DIST_DIR/Image.lz4-dtb
```

[Flashing and booting kernels with fastboot]

Most recent devices have a bootloader extension to streamline the process of generating and booting a boot image.

To boot the kernel without flashing:

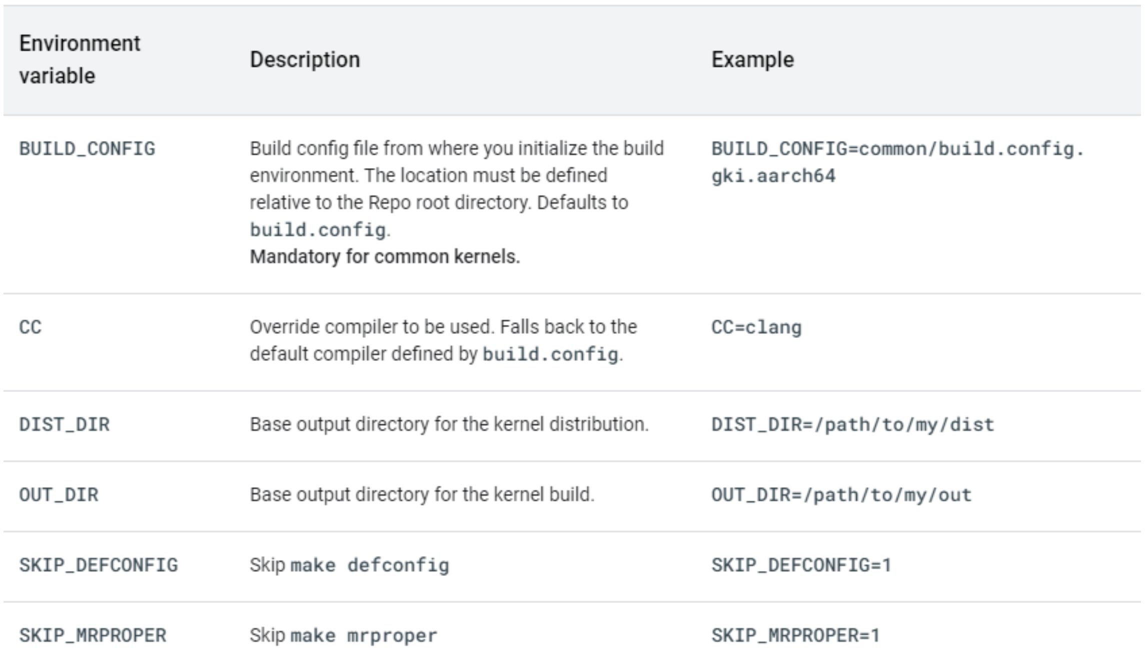
```
adb reboot bootloader
fastboot boot Image.lz4-dtb
```

Using this method, the kernel isn't actually flashed, and won't persist across a reboot.

Note: Kernel names differ by device. To locate the correct filename for your kernel, refer to `device/VENDOR/NAME-kernel` in the AOSP tree.

Customizing the kernel build

The build process and outcome can be influenced by environment variables. Most of them are optional and each kernel branch should come with a proper default configuration. The most frequently used ones are listed here. For a complete (and up-to-date) list, refer to `build/build.sh`.



Environment variable	Description	Example
`BUILD_CONFIG`	Build config file from where you initialize the build environment. The location must be defined relative to the Repo root directory. Defaults to `build.config`. **Mandatory for common kernels.**	`BUILD_CONFIG=common/build.config.gki.aarch64`
`CC`	Override compiler to be used. Falls back to the default compiler defined by `build.config`.	`CC=clang`
`DIST_DIR`	Base output directory for the kernel distribution.	`DIST_DIR=/path/to/my/dist`
`OUT_DIR`	Base output directory for the kernel build.	`OUT_DIR=/path/to/my/out`
`SKIP_DEFCONFIG`	Skip `make defconfig`	`SKIP_DEFCONFIG=1`
`SKIP_MRPROPER`	Skip `make mrproper`	`SKIP_MRPROPER=1`

[Custom kernel config for local builds]

If you need to switch a kernel configuration option regularly, for example, when working on a feature, or if you need an option to be set for development purposes, you can achieve that flexibility by maintaining a local modification or copy of the build config.

Set the variable `POST_DEFCONFIG_CMDS` to a statement that is evaluated right after the usual `make defconfig` step is done. As the `build.config` files are sourced into the build environment, functions defined in `build.config` can be called as part of the post-defconfig commands.

A common example is disabling link time optimization (LTO) for crosshatch kernels during development. While LTO is beneficial for released kernels, the overhead at build time can be significant. The following snippet added to the local `build.config` disables LTO persistently when using `build/build.sh`.

```
POST_DEFCONFIG_CMDS="check_defconfig && update_debug_config"
function update_debug_config() {
    $/.config \
         -d LTO \
         -d LTO_CLANG \
         -d CFI \
         -d CFI_PERMISSIVE \
         -d CFI_CLANG
    (cd $ && \
     make O=$ olddefconfig)
}
```

[Identifying kernel versions]

You can identify the correct version to build from two sources: the AOSP tree and the system image.

[Kernel version from AOSP tree]

The AOSP tree contains prebuilt kernel versions. The git log reveals the correct version as part of the commit message:

```
cd $AOSP/device/VENDOR/NAME
git log --max-count=1
```

If the kernel version isn't listed in the git log, obtain it from the system image, as described below.

[Kernel version from system image]

To determine the kernel version used in a system image, run the following command against the kernel file:

```
file kernel
```

For `Image.lz4-dtb` files, run:

```
grep -a 'Linux version' Image.lz4-dtb
```

[Building a Boot Image]

It's possible to build a boot image using the kernel build environment. To do this you need a ramdisk binary, which you can obtain by downloading a GKI boot image and unpacking it. Any GKI boot image from the associated Android release will work.

```
tools/mkbootimg/unpack_bootimg.py --boot_img=boot-5.4-gz.img
mv tools/mkbootimg/out/ramdisk gki-ramdisk.lz4
```

The target folder is the top-level directory of the kernel tree (the current working directory).

If you're developing with AOSP master, you can instead download the `ramdisk-recovery.img` build artifact from an aosp_arm64 build on ci.android.com and use that as your ramdisk binary.

When you have a ramdisk binary and have copied it to `gki-ramdisk.lz4` in the root directory of the kernel build, you can generate a boot image by executing:

```
BUILD_BOOT_IMG=1 SKIP_VENDOR_BOOT=1 KERNEL_BINARY=Image
GKI_RAMDISK_PREBUILT_BINARY=gki-ramdisk.lz4
BUILD_CONFIG=common/build.config.gki.aarch64 build/build.sh
```

If you're working with x86-based architecture, replace `Image` with `bzImage`, and `aarch64` with `x86_64`:

```
BUILD_BOOT_IMG=1 SKIP_VENDOR_BOOT=1 KERNEL_BINARY=bzImage
GKI_RAMDISK_PREBUILT_BINARY=gki-ramdisk.lz4
BUILD_CONFIG=common/build.config.gki.x86_64 build/build.sh
```

That file is located in the artifact directory `$KERNEL_ROOT/out/$KERNEL_VERSION/dist`.

The boot image is located at `out/<kernel branch>/dist/boot.img`.