# Robotnix – Build Android (AOSP) using Nix

Daniel Fullmer

October 16, 2020

# Introduction

How to build Android (AOSP):
(according to `https://source.android.com/setup/start`)

# Introduction

How to build Android (AOSP):
(according to `https://source.android.com/setup/start`)

On Ubuntu 18.04:

```
$ sudo apt-get install git-core gnupg flex bison \
    build-essential zip curl  zlib1g-dev gcc-multilib \
    g++-multilib libc6-dev-i386 lib32ncurses5-dev \
    x11proto-core-dev libx11-dev lib32z1-dev \
    libgl1-mesa-dev libxml2-utils \
    xsltproc unzip fontconfig
```

# Introduction

Fetching AOSP source:

```
$ curl \
  https://storage.googleapis.com/git-repo-downloads/repo \
  > ~/bin/repo
$ chmod a+x ~/bin/repo
```

# Introduction

Fetching AOSP source:

```
$ curl \
  https://storage.googleapis.com/git-repo-downloads/repo \
  > ~/bin/repo
$ chmod a+x ~/bin/repo

$ mkdir aosp
$ cd aosp
$ repo init \
  -u https://android.googlesource.com/platform/manifest \
  -b android-11.0.0_r5
$ repo sync -j8
```

# Introduction

Fetching AOSP source:

```
$ curl \
  https://storage.googleapis.com/git-repo-downloads/repo \
  > ~/bin/repo
$ chmod a+x ~/bin/repo

$ mkdir aosp
$ cd aosp
$ repo init \
  -u https://android.googlesource.com/platform/manifest \
  -b android-11.0.0_r5
$ repo sync -j8
```

Make sure you know the tag (android-11.0.0_r5) of the latest release.

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

- Build products will be under out/target/product/crosshatch.

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

- Build products will be under out/target/product/crosshatch.
- Except, this uses old kernel and chromium webview prebuilts.

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

- Build products will be under out/target/product/crosshatch.
- Except, this uses old kernel and chromium webview prebuilts.
- This will also be missing proprietary vendor binaries.

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

- Build products will be under out/target/product/crosshatch.
- Except, this uses old kernel and chromium webview prebuilts.
- This will also be missing proprietary vendor binaries.
- This build isn't signed with secure keys.

# Introduction

Build the code: (for Pixel 3 XL)

```
$ source build/envsetup.sh
$ lunch aosp_crosshatch-user
$ make
```

- Build products will be under out/target/product/crosshatch.
- Except, this uses old kernel and chromium webview prebuilts.
- This will also be missing proprietary vendor binaries.
- This build isn't signed with secure keys.
- There is documentation on all these issues, but they are not readily accessible and require many more steps

# Quick Start

To build and flash a Pixel 3 XL image:

```
$ git clone https://github.com/danielfullmer/robotnix
$ cd robotnix
$ nix-build \
    --arg configuration \
    '{ device="crosshatch"; flavor="vanilla"; }' \
    -A img
$ fastboot update -w ./result
```

# Why Nix?

- Good at integrating build tools across the diverse projects (AOSP, Chromium, Linux kernel, etc)
- Reliability, reproducibility
- Sandboxed builds
  - Android 10 now does this partially with “nsjail” (no network access)
- All build inputs are hashed. (Avoids sneaking in impurities)

# Broad Goals

- Simplicity
- Customizability
- Reproducibility (bit-for-bit)
- Authenticity
- Security
- Privacy

# Module System

Uses a NixOS-style module system:

```
{
  device = "crosshatch";
  flavor = "vanilla";

  apps.updater = {
    enable = true;
    url = "https://example.com/android";
  };

  apps.fdroid.enable = true;
  microg.enable = true;
  apps.bromite.enable = true;
  webview.bromite.enable = true;
}
```

# Optional Modules

- Flavors: Vanilla, GrapheneOS, LineageOS
- Chromium / Webview
- Linux Kernel
- Over-the-Air (OTA) Updater
- Signed builds
- F-Droid + Privileged Extension (enables background app updates)
- MicroG
- Seedvault Backup
- Others...

# Flavors

- Vanilla AOSP (`https://android.googlesource.com/`)
  - Android 11 (released Sept 2020)
  - Pixel phones
- GrapheneOS
  - Privacy and security focused
  - Android 11
  - Pixel phones
- LineageOS
  - Android 10
  - $\geq$ 110 devices
  - “Experimental” support

# Chromium / Webview

```
{
  apps.chromium.enable = true;
  webview.chromium.enable = true;
}
```

"Android WebView is a system component powered by Chrome that allows Android apps to display web content"

Chromium Forks:

- Bromite (privacy-focused fork)
- Vanadium (security-focused fork. GrapheneOS)

# Linux Kernel

```
{
  kernel.src = fetchGit { ... };
  kernel.patches = [ ./example.patch ];
  kernel.compiler = "clang";
}
```

# Signed Builds

```
{
  signing.enable = true;
  keyStorePath = "/var/secrets/android";
}
```

Keys can be easily generated using “generateKeysScript”

Over-the-air (OTA) / sideloaded updates must be signed by the generated keys.

# Signed Builds

```
{
  signing.enable = true;
  keyStorePath = "/var/secrets/android";
}
```

Keys can be easily generated using “generateKeysScript”

Over-the-air (OTA) / sideloaded updates must be signed by the generated keys.

Pixel phones use Android Verified Boot (AVB) with a user-settable root of trust. Read-only system image. Any modifications to underlying system must have a valid signature. Helps prevent persistent exploitation.

# Over-the-air (OTA) Updater

Updater app from GrapheneOS:

```
{
  apps.updater.enable = true;
  apps.updater.url = "https://example.com/android/";
}
```

Updater URL needs to point to a directory containing OTA files and some update metadata:

```
$ nix-build ... -A otaDir -o /data/webroot/android
```

# F-Droid

"Free and Open Source Android App Repository"

```
{
  apps.fdroid.enable = true;
}
```

Privileged Extension:

- Install apps without needing "Unknown Sources" to be enabled (like Google Play).
- Install updates in the background without the user having to click "install".

# MicroG

"A free-as-in-freedom re-implementation of Google's proprietary Android user space apps and libraries"

```
{
  apps.microg.enable = true;
}
```

# Seedvault Backup

"A backup application using Android's internal backup API"

```
{
  apps.seedvault.enable = true;
}
```

Allows you to automatically backup application data to a USB drive / Nextcloud / etc.

# Source Files

```
{
  source.dirs."extra-dir".src = fetchGit { ... };

  source.dirs."frameworks/base".patches = [
    ./example.patch
  ];
}
```

Build environment bind mounts read-only source dirs directly from /nix/store/. (Avoids copying)

# Source Files

```
{
  source.dirs."extra-dir".src = fetchGit { ... };

  source.dirs."frameworks/base".patches = [
    ./example.patch
  ];
}
```

Build environment bind mounts read-only source dirs directly from /nix/store/. (Avoids copying)

The flavor option mostly sets up the default source.dirs

# Prebuilt Apps

Include prebuilt apps in the built image:

```
{
  apps.prebuilt.ExampleApp.apk = ./example.apk;
}
```

Naturally, this could refer to some .apk output file from any other derivation.

# Prebuilt Apps

Include prebuilt apps in the built image:

```
{
  apps.prebuilt.ExampleApp.apk = ./example.apk;
}
```

Naturally, this could refer to some .apk output file from any other derivation.

Other modules like Seedvault and MicroG are implemented using apps.prebuilt

# Emulator

Script to launch the emulator with an attached robotnix-built image:

```
$ nix-build \
    --arg configuration \
    { device="x86_64"; flavor="vanilla"; } \
    -A build.emulator
$ ./result
```

# Building SDK

Prebuilt Android SDK is published under an additional "SDK License"

Code used to build SDK is mostly Apache and GPL.

# Building SDK

Prebuilt Android SDK is published under an additional "SDK License"

Code used to build SDK is mostly Apache and GPL.

```
$ nix-build ./sdk
$ ls result
android-sdk_eng.10.0.0_r33_linux-x86.zip
repository.xml
repo-sys-img.xml
sdk-repo-linux-build-tools-eng.10.0.0_r33.zip
sdk-repo-linux-docs-eng.10.0.0_r33.zip
sdk-repo-linux-platforms-eng.10.0.0_r33.zip
sdk-repo-linux-platform-tools-eng.10.0.0_r33.zip
sdk-repo-linux-samples-eng.10.0.0_r33.zip
sdk-repo-linux-sources-eng.10.0.0_r33.zip
sdk-repo-linux-system-images-eng.10.0.0_r33.zip
```

# Reproducibility Status

Google does a good job with reproducibility overall.

A few small patches required.

In the past, verified to be bit-for-bit reproducible for Pixel 3 XL and Pixel 1 XL using the Vanilla flavor.

LineageOS also has some patches and has been tested as well.

# Reproducibility Status

Google does a good job with reproducibility overall.

A few small patches required.

In the past, verified to be bit-for-bit reproducible for Pixel 3 XL and Pixel 1 XL using the Vanilla flavor.

LineageOS also has some patches and has been tested as well.

Automatically publish a "reproducibility report" like `https://r13y.com/`

Remainder of talk is ideas for future work on Robotnix

# Publishing Build Products

Robotnix currently has a single large derivation which executes the full android build process.

# Publishing Build Products

Robotnix currently has a single large derivation which executes the full android build process.

Signing the build can be done in another small derivation that depends on this.

# Publishing Build Products

Robotnix currently has a single large derivation which executes the full android build process.

Signing the build can be done in another small derivation that depends on this.

Publish to Amazon S3 / Cachix

Obvious downsides:

- 3.4GB output per build (target files)
- Each configuration is a different build.
- Combinatorial explosion with increasing number of options.
- Publish a smaller subset? (1-2 per device?)

# m-of-n verification

If the target files are bit-for-bit reproducible, we could have multiple independent builders each publishing their unsigned target files (but signed with their own nix binary cache keys).

nix copy-sigs

nix verify –sigs-needed <m>

After verifying the target files, a user could then sign with their own keys and generate their own img / ota files.

# Incremental / Shared Builds

Most of the build takes places in one very large derivation.

# Incremental / Shared Builds

Most of the build takes places in one very large derivation.

Changes to the robotnix configuration often require a full rebuild.

# Incremental / Shared Builds

Most of the build takes places in one very large derivation.

Changes to the robotnix configuration often require a full rebuild.

ccache is available, but insufficient

# Incremental / Shared Builds

Most of the build takes places in one very large derivation.

Changes to the robotnix configuration often require a full rebuild.

ccache is available, but insufficient

Small changes to the robotnix configuration could be built much more quickly by sharing some common intermediate build products.

# Incremental / Shared Builds

Most of the build takes places in one very large derivation.

Changes to the robotnix configuration often require a full rebuild.

ccache is available, but insufficient

Small changes to the robotnix configuration could be built much more quickly by sharing some common intermediate build products.

Will describe one attempt at solving this: blueprint2nix / soongnix

# Android Build System

- Makefiles (ckati)
- Blueprint / Soong
- Generates a combined '.ninja' file

# Blueprint Files

Example Android.bp file:

```
cc_binary {
    name: "ext2simg",
    host_supported: true,
    defaults: ["e2fsprogs-defaults"],

    srcs: ["ext2simg.c"],
    shared_libs: [
        "libext2fs",
        "libext2_com_err",
        "libsparse",
        "libz",
    ],
}
```

# Blueprint Files

Example Android.bp file:

```
cc_binary {
    name = "ext2simg";
    host_supported = true;
    defaults = ["e2fsprogs-defaults"];

    srcs = ["ext2simg.c"];
    shared_libs = [
        "libext2fs",
        "libext2_com_err",
        "libsparse",
        "libz"
    ];
}
```

# Blueprint Files

Example Android.bp file:

```
ext2simg = cc_binary {
    name = "ext2simg";
    host_supported = true;
    defaults = ["e2fsprogs-defaults"];

    srcs = ["ext2simg.c"];
    shared_libs = [
        "libext2fs",
        "libext2_com_err",
        "libsparse",
        "libz"
    ];
};
```

# Blueprint Files

```
{ cc_binary }:
let
ext2simg = cc_binary {
    name = "ext2simg";
    host_supported = true;
    defaults = ["e2fsprogs-defaults"];

    srcs = ["ext2simg.c"];
    shared_libs = [
        "libext2fs",
        "libext2_com_err",
        "libsparse",
        "libz"
    ];
};
in { inherit ext2simg }
```

# Blueprint2nix

`https://github.com/danielfullmer/blueprint2nix`

Uses modified version of the bpfmt Blueprint formatter to output '.nix' files.

# Blueprint2nix

`https://github.com/danielfullmer/blueprint2nix`

Uses modified version of the bpfmt Blueprint formatter to output '.nix' files.

Still need to implement things like cc_binary.

# Soongnix

`https://github.com/danielfullmer/soongnix`

Nix partial reimplementation of some soong "modules" (e.g. cc_binary, cc_library, python_binary_host, etc)

# Soongnix

`https://github.com/danielfullmer/soongnix`

Nix partial reimplementation of some soong "modules" (e.g. cc_binary, cc_library, python_binary_host, etc)

Can build adb and fastboot using the nixpkgs native clang toolchain. (also on 'aarch64-linux')

# Soongnix

`https://github.com/danielfullmer/soongnix`

Nix partial reimplementation of some soong "modules" (e.g. cc_binary, cc_library, python_binary_host, etc)

Can build adb and fastboot using the nixpkgs native clang toolchain. (also on 'aarch64-linux')

Some benefits from using Nix:

- laziness!
- derivations only depend on the source files they explicitly reference.
- only need to download a small part of the source tree

# Soongnix

`https://github.com/danielfullmer/soongnix`

Nix partial reimplementation of some soong "modules" (e.g. cc_binary, cc_library, python_binary_host, etc)

Can build adb and fastboot using the nixpkgs native clang toolchain. (also on 'aarch64-linux')

Some benefits from using Nix:

- laziness!
- derivations only depend on the source files they explicitly reference.
- only need to download a small part of the source tree

Proof of concept! Likely not maintainable for all of Android. (Maybe just host binaries?)

# Soongnix

`https://github.com/danielfullmer/soongnix`

Nix partial reimplementation of some soong "modules" (e.g. cc_binary, cc_library, python_binary_host, etc)

Can build adb and fastboot using the nixpkgs native clang toolchain. (also on 'aarch64-linux')

Some benefits from using Nix:

- laziness!
- derivations only depend on the source files they explicitly reference.
- only need to download a small part of the source tree

Proof of concept! Likely not maintainable for all of Android. (Maybe just host binaries?)

Could point the way to a potential future "bazel2nix"?

Jobset robotnix:soongnix

Evaluations | Evaluation errors | Jobs | Configuration | Links | Channels

**Last checked:** 2020-09-22 12:10:34, *with errors!*

**Last evaluation:** 2020-08-17 20:09:32

| | | | Jobs | | | |
|---|---|---|---|---|---|---|
| **#** | **Date** | **Input changes** | ✓ | ✗ | ? | Δ |
| 251 | 2020-08-17 | soongnix → 20200818030810 | 413 | 614 | | +9 |
| 250 | 2020-08-17 | soongnix → 20200818025951 | 404 | 623 | | +88 |
| 249 | 2020-08-17 | nixpkgs → 16fc531, soongnix → 20200818025505 | 316 | 549 | | -220 |
| 247 | 2020-08-17 | soongnix → 20200817070923 | 536 | 706 | | +10 |
| 246 | 2020-08-16 | soongnix → 20200817065700 | 526 | 716 | | +2 |
| 245 | 2020-08-16 | soongnix → 20200817061017 | 524 | 718 | | +2 |

| Job | 2020-08-17 | 2020-08-17 | 2020-08-17 | 2020-08-17 | 2020-08-16 | 2020-08-16 | 2020-08-16 | 2020-08-16 | 2020-08-16 |
|---|---|---|---|---|---|---|---|---|---|
| BlobCache_test | ✗ | ✗ | ✗ | | | | | | |
| FileCheck | ✗ | ✗ | ✗ | | | | | | |
| JniInvocation_test | ✗ | ✗ | ✗ | | | | | | |
| JniSafeRegisterNativeMethods_test | ✗ | ✗ | ✗ | | | | | | |
| LLVMHello | ✗ | ✗ | ✗ | | | | | | |
| Magick++ | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Magick++_platform | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MagickCore | | | | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| MagickWand | | | | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Magick_coders | | | | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Magick_filters | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

# Further Robotnix goals

- Continuous Integration

# Further Robotnix goals

- Continuous Integration
- NixOS module that integrates with robotnix configuration

```
{
    robotnix = {
        enable = true;
        autoUpgrade = true;
        baseUrl = "https://example.com/android";
        configurations = [
            ...
        ];
    };
}
```

# Final Notes

- I've been using this for over since July 2019 ($>$ 1 year) on my daily driver (Pixel 3 XL).
- Haven't lost any data, or had to wipe/reflash. Only used OTA updates.
- Working on making Robotnix useful for other people, not just me.
- Documentation, etc.

# Final Notes

- I've been using this for over since July 2019 ($>$ 1 year) on my daily driver (Pixel 3 XL).
- Haven't lost any data, or had to wipe/reflash. Only used OTA updates.
- Working on making Robotnix useful for other people, not just me.
- Documentation, etc.


# Conclusion

Try out Robotnix:
`https://github.com/danielfullmer/robotnix`

# Conclusion

Try out Robotnix:
`https://github.com/danielfullmer/robotnix`

Also check out some related projects:

- NixDroid (ajs124 and dasJ)
- mobile-nixos (samueldr)
- RattlesnakeOS (builds in AWS)
- HashbangOS / aosp-build