Establishing a Build Environment

source.android.com /source/initializing.html

In this document

This section describes how to set up your local work environment to build the Android source files. You will need to use Linux or Mac OS. Building under Windows is not currently supported.

For an overview of the entire code-review and code-update process, see Life of a Patch.

Choosing a Branch

Some of the requirements for your build environment are determined by which version of the source code you plan to compile. See Build Numbers for a full listing of branches you may choose from. You may also choose to download and build the latest source code (called `master`), in which case you will simply omit the branch specification when you initialize the repository.

Once you have selected a branch, follow the appropriate instructions below to set up your build environment.

Setting up a Linux build environment

These instructions apply to all branches, including `master`.

The Android build is routinely tested in house on recent versions of Ubuntu LTS (14.04), but most distributions should have the required build tools available. Reports of successes or failures on other distributions are welcome.

For Gingerbread (2.3.x) and newer versions, including the `master` branch, a 64-bit environment is required. Older versions can be compiled on 32-bit systems.

Note: See the Requirements for the complete list of hardware and software requirements. Then follow the detailed instructions for Ubuntu and Mac OS below.

Installing the JDK

The `master` branch of Android in the Android Open Source Project (AOSP) requires Java 7. On Ubuntu, use OpenJDK.

Java 7: For the latest version of Android

```
$ sudo apt-get update
$ sudo apt-get install openjdk-7-
jdk
```

Optionally, update the default Java version by running:

```
$ sudo update-alternatives --config java
$ sudo update-alternatives --config
javac
```

If you encounter version errors for Java, set its path as described in the Wrong Java Version section.

To develop older versions of Android, download and install the corresponding version of the Java JDK:
Java 6: for Gingerbread through KitKat
Java 5: for Cupcake through Froyo

Note: The `lunch` command in the build step will ensure that the Sun JDK is used instead of any previously installed JDK.

Installing required packages (Ubuntu 14.04)

You will need a 64-bit version of Ubuntu. Ubuntu 14.04 is recommended.

```
$ sudo apt-get install bison g++-multilib git gperf libxml2-utils make
python-networkx zlib1g-dev:i386 zip
```

Installing required packages (Ubuntu 12.04)

You may use Ubuntu 12.04 to build older versions of Android. Version 12.04 is not supported on master or recent releases.

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
 zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
 libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
 libgl1-mesa-dev g++-multilib mingw32 tofrodos \
 python-markdown libxml2-utils xsltproc zlib1g-dev:i386
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-
gnu/libGL.so
```

Installing required packages (Ubuntu 10.04 -- 11.10)

Building on Ubuntu 10.04-11.10 is no longer supported, but may be useful for building older releases of AOSP.

```
$ sudo apt-get install git gnupg flex bison gperf build-essential
\
 zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs \
 x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev \
 libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown \
 libxml2-utils xsltproc
```

On Ubuntu 10.10:

```
$ sudo ln -s /usr/lib32/mesa/libGL.so.1
/usr/lib32/mesa/libGL.so
```

On Ubuntu 11.10:

```
$ sudo apt-get install libx11-
dev:i386
```

Configuring USB Access

Under GNU/Linux systems (and specifically under Ubuntu systems), regular users can't directly access USB devices by default. The system needs to be configured to allow such access.

The recommended approach is to create a file at `/etc/udev/rules.d/51-android.rules` (as the root user).

To do this, run the following command to download the 51-android.rules file attached to this site, modify it to include your username, and place it in the correct location:

```
$ wget -S -O - http://source.android.com/source/51-android.rules | sed
"s/<username>/$USER/" | sudo tee >/dev/null /etc/udev/rules.d/51-
android.rules; sudo udevadm control --reload-rules
```

Those new rules take effect the next time a device is plugged in. It might therefore be necessary to unplug the device and plug it back into the computer.

This is known to work on both Ubuntu Hardy Heron (8.04.x LTS) and Lucid Lynx (10.04.x LTS). Other versions of Ubuntu or other variants of GNU/Linux might require different configurations.

Using a separate output directory

By default, the output of each build is stored in the `out/` subdirectory of the matching source tree.

On some machines with multiple storage devices, builds are faster when storing the source files and the output on separate volumes. For additional performance, the output can be stored on a filesystem optimized for speed instead of crash robustness, since all files can be re-generated in case of filesystem corruption.

To set this up, export the `OUT_DIR_COMMON_BASE` variable to point to the location where your output directories will be stored.

```
export OUT_DIR_COMMON_BASE=<path-to-your-out-
directory>
```

The output directory for each separate source tree will be named after the directory holding the source tree.

For instance, if you have source trees as `/source/master1` and `/source/master2` and `OUT_DIR_COMMON_BASE` is set to `/output`, the output directories will be `/output/master1` and `/output/master2`.

It's important in that case to not have multiple source trees stored in directories that have the same name, as those would end up sharing an output directory, with unpredictable results.

This is only supported on Jelly Bean (4.1) and newer, including the `master` branch.

Setting up a Mac OS build environment

In a default installation, Mac OS runs on a case-preserving but case-insensitive filesystem. This type of filesystem is not supported by git and will cause some git commands (such as `git status`) to behave abnormally. Because of this, we recommend that you always work with the AOSP source files on a case-sensitive filesystem. This can be done fairly easily using a disk image, discussed below.

Once the proper filesystem is available, building the `master` branch in a modern Mac OS environment is very straightforward. Earlier branches, including ICS, require some additional tools and SDKs.

Creating a case-sensitive disk image

You can create a case-sensitive filesystem within your existing Mac OS environment using a disk image. To create the image, launch Disk Utility and select "New Image". A size of 25GB is the minimum to complete the build; larger numbers are more future-proof. Using sparse images saves space while allowing to grow later as the need arises. Be sure to select "case sensitive, journaled" as the volume format.

You can also create it from a shell with the following command:

```
# hdiutil create -type SPARSE -fs 'Case-sensitive Journaled HFS+' -size 40g
~/android.dmg
```

This will create a `.dmg` (or possibly a `.dmg.sparsefile`) file which, once mounted, acts as a drive with the required formatting for Android development.

If you need a larger volume later, you can also resize the sparse image with the following command:

```
# hdiutil resize -size <new-size-you-want>g
~/android.dmg.sparseimage
```

For a disk image named `android.dmg` stored in your home directory, you can add helper functions to your `~/.bash_profile`:

- To mount the image when you execute `mountAndroid`:

  ```
  # mount the android file image
  function mountAndroid { hdiutil attach ~/android.dmg -mountpoint
  /Volumes/android; }
  ```

  Note: If your system created a `.dmg.sparsefile` file, replace `~/android.dmg` with `~/android.dmg.sparsefile`.

- To unmount it when you execute `umountAndroid`:

  ```
  # unmount the android file image
  function umountAndroid() { hdiutil detach /Volumes/android;
  }
  ```

Once you've mounted the `android` volume, you'll do all your work there. You can eject it (unmount it) just like you would with an external drive.

Installing the JDK

The `master` and `5.0.x` branches of Android in the Android Open Source Project (AOSP) require Java 7. On Mac OS, use jdk-7u71-macosx-x64.dmg.

To develop for versions of Android Gingerbread through KitKat, download and install the Java 6 version of the Java JDK.

Master branch

To build the latest source in a Mac OS environment, you will need an Intel/x86 machine running Mac OS X v10.8 (Mountain Lion) or later, along with Xcode 4.5.2 or later including the Command Line Tools.

Branch 5.0.x and earlier branches

To build 5.0.x and earlier source in a Mac OS environment, you will need an Intel/x86 machine running Mac OS X v10.8 (Mountain Lion), along with Xcode 4.5.2 and Command Line Tools.

Branch 4.4.x and earlier branches

To build 4.2.x and earlier source in a Mac OS environment, you will need an Intel/x86 machine running Mac OS X v10.6 (Snow Leopard) or Mac OS X v10.7 (Lion), along with Xcode 4.2 (Apple's Developer Tools). Although Lion does not come with a JDK, it should install automatically when you attempt to build the source.

The remaining sections for Mac OS apply only to those who wish to build earlier branches.

Branch 4.0.x and all earlier branches

To build android-4.0.x and earlier branches in a Mac OS environment, you need an Intel/x86 machine running Mac OS X v10.5 (Leopard) or Mac OS X v10.6 (Snow Leopard). You will need the Mac OS X v10.5 SDK.

Installing required packages

- Install Xcode from the Apple developer site. We recommend version 3.1.4 or newer (e.g., gcc 4.2). Version 4.x could cause difficulties. If you are not already registered as an Apple developer, you will have to create an Apple ID in order to download.

- Install MacPorts from macports.org.

  Note: Make sure that `/opt/local/bin` appears in your path **before** `/usr/bin`. If not, please add the following to your `~/.bash_profile` file:

  ```
  export
  PATH=/opt/local/bin:$PATH
  ```

  Note: If you do not have a `.bash_profile` file in your home directory, create one.

- Get make, git, and GPG packages from MacPorts:

  ```
  $ POSIXLY_CORRECT=1 sudo port install gmake libsdl git
  gnupg
  ```

  If using Mac OS X v10.4, also install bison:

  ```
  $ POSIXLY_CORRECT=1 sudo port install
  bison
  ```

Reverting from make 3.82

For versions of Android before ICS, there is a bug in gmake 3.82 that prevents android from building.

You can install version 3.81 using MacPorts by taking the following steps:

- Edit `/opt/local/etc/macports/sources.conf` and add a line that says

  ```
  file:///Users/Shared/dports
  ```

  above the rsync line. Then create this directory:

  ```
  $ mkdir
  /Users/Shared/dports
  ```

- In the new `dports` directory, run

  ```
  $ svn co --revision 50980
  http://svn.macports.org/repository/macports/trunk/dports/devel/gmake/
  devel/gmake/
  ```

- Create a port index for your new local repository:

  ```
  $ portindex /Users/Shared/dports
  ```

- Finally, install the old version of gmake with

  ```
  $ sudo port install gmake
  @3.81
  ```

Setting a file descriptor limit

On Mac OS, the default limit on the number of simultaneous file descriptors open is too low and a highly parallel build process may exceed this limit.

To increase the cap, add the following lines to your `~/.bash_profile`:

```
# set the number of open files to be
1024
ulimit -S -n 1024
```

Optimizing a build environment (optional)

Setting up ccache

You can optionally tell the build to use the ccache compilation tool. Ccache acts as a compiler cache that can be used to speed up rebuilds. This works very well if you use `make clean` often, or if you frequently switch between different build products.

Put the following in your `.bashrc` (or equivalent):

```
export
USE_CCACHE=1
```

By default the cache will be stored in `~/.ccache`. If your home directory is on NFS or some other non-local filesystem, you will want to specify the directory in your `.bashrc` file as well:

```
export CCACHE_DIR=<path-to-your-cache-directory>
```

The suggested cache size is 50-100GB. You will need to run the following command once you have downloaded the source code:

```
prebuilts/misc/linux-x86/ccache/ccache -M
50G
```

On Mac OS, you should replace `linux-x86` with `darwin-x86`:

```
prebuilts/misc/darwin-x86/ccache/ccache -M
50G
```

When building Ice Cream Sandwich (4.0.x) or older, ccache is in a different location:

```
prebuilt/linux-x86/ccache/ccache -M
50G
```

This setting is stored in the CCACHE_DIR and is persistent.

Next: Download the source

Your build environment is good to go! Proceed to downloading the source.