Checking out and building Chromium for Android

There are instructions for other platforms linked from the get the code page.

Instructions for Google Employees

Are you a Google employee? See go/building-android-chrome instead.

System requirements

- A 64-bit Intel machine running Linux with at least 8GB of RAM. More than 16GB is highly recommended.
- At least 100GB of free disk space.
- You must have Git and Python installed already.

Most development is done on Ubuntu. Other distros may or may not work; see the Linux instructions for some suggestions.

Building the Android client on Windows or Mac is not supported and doesn't work.

Install `depot_tools`

Clone the `depot_tools` repository:

```
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
```

Add `depot_tools` to the end of your PATH (you will probably want to put this in your `~/.bashrc` or `~/.zshrc`). Assuming you cloned `depot_tools` to `/path/to/depot_tools`:

```
export PATH="$PATH:/path/to/depot_tools"
```

Get the code

Create a `chromium` directory for the checkout and change to it (you can call this whatever you like and put it wherever you like, as long as the full path has no spaces):

```
mkdir ~/chromium && cd ~/chromium
fetch --nohooks android
```

If you don't want the full repo history, you can save a lot of time by adding the `--no-history` flag to `fetch`.

Expect the command to take 30 minutes on even a fast connection, and many hours on slower ones.

If you've already installed the build dependencies on the machine (from another checkout, for example), you can omit the `--nohooks` flag and `fetch` will automatically execute `gclient runhooks` at the end.

When `fetch` completes, it will have created a hidden `.gclient` file and a directory called `src` in the working directory. The remaining instructions assume you have switched to the `src` directory:

```
cd src
```

Converting an existing Linux checkout

If you have an existing Linux checkout, you can add Android support by appending `target_os = ['android']` to your `.gclient` file (in the directory above `src`):

```
echo "target_os = [ 'android' ]" >> ../.gclient
```

Then run `gclient sync` to pull the new Android dependencies:

```
gclient sync
```

(This is the only difference between `fetch android` and `fetch chromium`.)

Install additional build dependencies

Once you have checked out the code, run

```
build/install-build-deps-android.sh
```

to get all of the dependencies you need to build on Linux, *plus* all of the Android-specific dependencies (you need some of the regular Linux dependencies because an Android build includes a bunch of the Linux tools and utilities).

Run the hooks

Once you've run `install-build-deps` at least once, you can now run the Chromium-specific hooks, which will download additional binaries and other things you might need:

```
gclient runhooks
```

Optional: You can also install API keys if you want your build to talk to some Google services, but this is not necessary for most development and testing purposes.

Setting up the build

Chromium uses Ninja as its main build tool along with a tool called GN to generate `.ninja` files. You can create any number of *build directories* with different configurations. To create a build directory which builds Chrome for Android, run:

```
gn gen --args='target_os="android"' out/Default
```

- You only have to run this once for each new build directory, Ninja will update the build files as needed.
- You can replace `Default` with another name, but it should be a subdirectory of `out`.
- For other build arguments, including release settings, see GN build configuration. The default will be a debug component build matching the current host operating system and CPU.
- For more info on GN, run `gn help` on the command line or read the quick start guide.

Also be aware that some scripts (e.g. `tombstones.py`, `adb_gdb.py`) require you to set `CHROMIUM_OUTPUT_DIR=out/Default`.

Build Chromium

Build Chromium with Ninja using the command:

```
autoninja -C out/Default chrome_public_apk
```

`autoninja` is a wrapper that automatically provides optimal values for the arguments passed to `ninja`.

You can get a list of all of the other build targets from GN by running `gn ls out/Default` from the command line. To compile one, pass the GN label to Ninja with no preceding “//” (so, for `//chrome/test:unit_tests` use `ninja -C out/Default chrome/test:unit_tests`).

Multiple Chrome APK Targets

The Google Play Store allows apps to send customized `.apk` files depending on the version of Android running on a device. Chrome uses this feature to target 3 different versions using 3 different ninja targets:

1. `chrome_public_apk` (ChromePublic.apk)
    - `minSdkVersion=16` (Jelly Bean).
    - Stores libchrome.so compressed within the APK.
    - Uses Crazy Linker.
    - Shipped only for Android < 21, but still works fine on Android >= 21.
2. `chrome_modern_public_apk` (ChromeModernPublic.apk)
    - `minSdkVersion=21` (Lollipop).
    - Uses Crazy Linker.
    - Stores libchrome.so uncompressed within the APK.
        - This APK is bigger, but the installation size is smaller since there is no need to extract the .so file.
3. `monochrome_public_apk` (MonochromePublic.apk)
    - `minSdkVersion=24` (Nougat).
    - Contains both WebView and Chrome within the same APK.
        - This APK is even bigger, but much smaller than SystemWebView.apk + ChromePublic.apk.
    - Stores libchrome.so uncompressed within the APK.
    - Does not use Crazy Linker (WebView requires system linker).
        - But system linker supports crazy linker features now anyways.

Note: These instructions use `chrome_public_apk`, but either of the other two targets can be substituted.

Note: These targets are actually the open-source equivalents to the closed-source targets that get shipped to the Play Store.

Note: For more in-depth differences, see android_native_libraries.md.

Updating your checkout

To update an existing checkout, you can run

```
$ git rebase-update
$ gclient sync
```

The first command updates the primary Chromium source repository and rebases any of your local branches on top of tip-of-tree (aka the Git branch `origin/master`). If you don't want to use this script, you can also just use `git pull` or other common Git commands to update the repo.

The second command syncs dependencies to the appropriate versions and re-runs hooks as needed.

Installing and Running Chromium on a device

Plug in your Android device

Make sure your Android device is plugged in via USB, and USB Debugging is enabled.

To enable USB Debugging:

- Navigate to Settings > About Phone > Build number
- Click ‘Build number’ 7 times
- Now navigate back to Settings > Developer Options
- Enable ‘USB Debugging’ and follow the prompts

You may also be prompted to allow access to your PC once your device is plugged in.

You can check if the device is connected by running:

```
third_party/android_tools/sdk/platform-tools/adb devices
```

Which prints a list of connected devices. If not connected, try unplugging and reattaching your device.

Enable apps from unknown sources

Allow Android to run APKs that haven't been signed through the Play Store:

- Enable ‘Unknown sources’ under Settings > Security

In case that setting isn't present, it may be possible to configure it via `adb shell` instead:

```
third_party/android_tools/sdk/platform-tools/adb shell settings put global verifier_verify_adb_installs 0
```

Build the full browser

```
autoninja -C out/Default chrome_public_apk
```

And deploy it to your Android device:

```
out/Default/bin/chrome_public_apk install
```

The app will appear on the device as “Chromium”.

Build Content shell

Wraps the content module (but not the /chrome embedder). See https://www.chromium.org/developers/content-module for details on the content module and content shell.

```
autoninja -C out/Default content_shell_apk
out/Default/bin/content_shell_apk install
```

this will build and install an Android apk under `out/Default/apks/ContentShell.apk`.

Build WebView

Android WebView is a system framework component. Since Android KitKat, it is implemented using Chromium code (based off the content module).

If you want to build the complete Android WebView framework component and test the effect of your chromium changes in Android apps using WebView, you should follow the Android AOSP + chromium WebView instructions

Running

For Content shell:

```
out/Default/bin/content_shell_apk launch [--args='--foo --bar'] http://example.com
```

For Chrome public:

```
out/Default/bin/chrome_public_apk launch [--args='--foo --bar'] http://example.com
```

Logging and debugging

Logging is often the easiest way to understand code flow. In C++ you can print log statements using the LOG macro. In Java, refer to android_logging.md.

You can see these log via `adb logcat`, or:

```
out/Default/bin/chrome_public_apk logcat
```

To debug C++ code, use one of the following commands:

```
out/Default/bin/content_shell_apk gdb
out/Default/bin/chrome_public_apk gdb
```

See Android Debugging Instructions for more on debugging, including how to debug Java code.

Testing

For information on running tests, see Android Test Instructions.

Faster Edit/Deploy

"Incremental install" uses reflection and side-loading to speed up the edit & deploy cycle (normally < 10 seconds). The initial launch of the apk will be a little slower since updated dex files are installed manually.

- Make sure to set`is_component_build = true`in your GN args
- All apk targets have *`_incremental` targets defined (e.g. `chrome_public_apk_incremental`) except for Webview and Monochrome

Here's an example:

```
autoninja -C out/Default chrome_public_apk_incremental
out/Default/bin/chrome_public_apk install --incremental --verbose
```

For gunit tests (note that run_*_incremental automatically add `--fast-local-dev` when calling `test_runner.py`):

```
autoninja -C out/Default base_unittests_incremental
out/Default/bin/run_base_unittests_incremental
```

For instrumentation tests:

```
autoninja -C out/Default chrome_public_test_apk_incremental
out/Default/bin/run_chrome_public_test_apk_incremental
```

To uninstall:

```
out/Default/bin/chrome_public_apk uninstall
```

To avoid typing `_incremental` when building targets, you can use the GN arg:

```
incremental_apk_by_default = true
```

This will make `chrome_public_apk` build in incremental mode.

Installing and Running Chromium on an Emulator

Running on an emulator is the same as on a device. Refer to android_emulator.md for setting up emulators.

Tips, tricks, and troubleshooting

Rebuilding libchrome.so for a particular release

These instructions are only necessary for Chrome 51 and earlier.

In the case where you want to modify the native code for an existing release of Chrome for Android (v25+) you can do the following steps. Note that in order to get your changes into the official release, you'll need to send your change for a codereview using the regular process for committing code to chromium.

1. Open Chrome on your Android device and visit chrome://version
2. Copy down the id listed next to “Build ID:”
3. Go to http://storage.googleapis.com/chrome-browser-components/BUILD_ID_FROM_STEP_2/index.html
4. Download the listed files and follow the steps in the README.