Checking out and building Chromium for Mac

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-chrome instead.

Contents

System requirements

- A 64-bit Intel Mac running 10.15.4+. (Building on Arm Macs is not yet supported.)
- Xcode 12.2+. This version of Xcode comes with ...
- The macOS 11.0 SDK. Run

  ```
  $ ls `xcode-select -p`/Platforms/MacOSX.platform/Developer/SDKs
  ```

  to check whether you have it. Building with a newer SDK usually works too (please fix it if it doesn't), but the releases currently use Xcode 12.2 and the macOS 11.0 SDK.

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 `~/.bash_profile` or `~/.zshrc`). Assuming you cloned `depot_tools` to `/path/to/depot_tools` (note: you **must** use the absolute path or Python will not be able to find infra tools):

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

Get the code

Ensure that unicode filenames aren't mangled by HFS:

```
$ git config --global core.precomposeUnicode true
```

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
```

Run the `fetch` tool from `depot_tools` to check out the code and its dependencies.

```
$ fetch chromium
```

If you don't need the full repo history, you can save time by using `fetch --no-history chromium`. You can call `git fetch --unshallow` to retrieve the full history later.

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

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
```

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:

```
$ gn gen 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.
- Building Chromium for arm Macs requires additional setup.

Faster builds

Full rebuilds are about the same speed in Debug and Release, but linking is a lot faster in Release builds.

Put

```
is_debug = false
```

in your `args.gn` to do a release build.

Put

```
is_component_build = true
```

in your `args.gn` to build many small dylibs instead of a single large executable. This makes incremental builds much faster, at the cost of producing a binary that opens less quickly. Component builds work in both debug and release.

Put

```
symbol_level = 0
```

in your args.gn to disable debug symbols altogether. This makes both full rebuilds and linking faster (at the cost of not getting symbolized backtraces in gdb).

CCache

You might also want to install ccache to speed up the build.

Build Chromium

Build Chromium (the “chrome” target) with Ninja using the command:

```
$ autoninja -C out/Default chrome
```

(`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 `autoninja -C out/Default chrome/test:unit_tests`).

Run Chromium

Once it is built, you can simply run the browser:

```
$ out/Default/Chromium.app/Contents/MacOS/Chromium
```

Avoiding the "incoming network connections" dialog

Every time you start a new developer build of Chrome you get a system dialog asking "Do you want the application Chromium.app to accept incoming network connections?" - to avoid this, run with this command-line flag:

--disable-features="MediaRouter"

Running test targets

You can run the tests in the same way. You can also limit which tests are run using the `--gtest_filter` arg, e.g.:

```
$ out/Default/unit_tests --gtest_filter="PushClientTest.*"
```

You can find out more about GoogleTest at its GitHub page.

Debugging

Good debugging tips can be found here. If you would like to debug in a graphical environment, rather than using `lldb` at the command line, that is possible without building in Xcode (see Debugging in Xcode).

Tips for printing variables from `lldb` prompt (both in Xcode or in terminal):

- If `uptr` is a `std::unique_ptr` , the address it wraps is accessible as `uptr.__ptr_.__value_` .
- To pretty-print `base::string16` , ensure you have a `~/.lldbinit` file and add the following

line into it (substitute {SRC} for your actual path to the root of Chromium's sources):

```
command script import {SRC}/tools/lldb/lldb_chrome.py
```

Update 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.

Tips, tricks, and troubleshooting

Using Xcode-Ninja Hybrid

While using Xcode is unsupported, GN supports a hybrid approach of using Ninja for building, but Xcode for editing and driving compilation. Xcode is still slow, but it runs fairly well even **with indexing enabled**. Most people build in the Terminal and write code with a text editor, though.

With hybrid builds, compilation is still handled by Ninja, and can be run from the command line (e.g. `autoninja -C out/gn chrome`) or by choosing the `chrome` target in the hybrid project and choosing Build.

To use Xcode-Ninja Hybrid pass `--ide=xcode` to `gn gen`:

```
$ gn gen out/gn --ide=xcode
```

Open it:

```
$ open out/gn/all.xcodeproj
```

You may run into a problem where http://YES is opened as a new tab every time you launch Chrome. To fix this, open the scheme editor for the Run scheme, choose the Options tab, and uncheck “Allow debugging when using document Versions Browser”. When this option is checked, Xcode adds `--NSDocumentRevisionsDebugMode YES` to the launch arguments, and the `YES` gets interpreted as a URL to open.

If you have problems building, join us in `#chromium` on `irc.freenode.net` and ask there. Be sure

that the waterfall is green and the tree is open before checking out. This will increase your chances of success.

Improving performance of `git status`

Increase the vnode cache size

`git status` is used frequently to determine the status of your checkout. Due to the large number of files in Chromium's checkout, `git status` performance can be quite variable. Increasing the system's vnode cache appears to help. By default, this command:

```
$ sysctl -a | egrep kern\..*vnodes
```

Outputs `kern.maxvnodes: 263168` (263168 is 257 * 1024). To increase this setting:

```
$ sudo sysctl kern.maxvnodes=$((512*1024))
```

Higher values may be appropriate if you routinely move between different Chromium checkouts. This setting will reset on reboot, the startup setting can be set in `/etc/sysctl.conf`:

```
$ echo kern.maxvnodes=$((512*1024)) | sudo tee -a /etc/sysctl.conf
```

Or edit the file directly.

Configure git to use an untracked cache

If `git --version` reports 2.8 or higher, try running

```
$ git update-index --test-untracked-cache
```

If the output ends with `OK`, then the following may also improve performance of `git status`:

```
$ git config core.untrackedCache true
```

If `git --version` reports 2.6 or higher, but below 2.8, you can instead run

```
$ git update-index --untracked-cache
```

Xcode license agreement

If you're getting the error

> Agreeing to the Xcode/iOS license requires admin privileges, please re-run as root via sudo.

the Xcode license hasn't been accepted yet which (contrary to the message) any user can do by running:

```
$ xcodebuild -license
```

Only accepting for all users of the machine requires root: