Build Instructions

Follow the guidelines below for building **Electron itself**, for the purposes of creating custom Electron binaries. For bundling and distributing your app code with the prebuilt Electron binaries, see the application distribution guide.

Platform prerequisites

Check the build prerequisites for your platform before proceeding

- macOS
- Linux
- Windows

Build Tools

Electron's Build Tools automate much of the setup for compiling Electron from source with different configurations and build targets. If you wish to set up the environment manually, the instructions are listed below.

Electron uses GN for project generation and ninja for building. Project configurations can be found in the `.gn` and `.gni` files.

GN Files

The following `gn` files contain the main rules for building Electron:

- `BUILD.gn` defines how Electron itself is built and includes the default configurations for linking with Chromium.
- `build/args/{testing,release,all}.gn` contain the default build arguments for building Electron.

GN prerequisites

You'll need to install `depot_tools`, the toolset used for fetching Chromium and its dependencies.

Also, on Windows, you'll need to set the environment variable `DEPOT_TOOLS_WIN_TOOLCHAIN=0`. To do so, open `Control Panel` → `System and Security` → `System` → `Advanced system settings` and add a system variable `DEPOT_TOOLS_WIN_TOOLCHAIN` with value `0`. This tells `depot_tools` to use your locally installed version of Visual Studio (by default, `depot_tools` will try to download a Google-internal version that only Googlers have access to).

Setting up the git cache

If you plan on checking out Electron more than once (for example, to have multiple parallel directories checked out to different branches), using the git cache

will speed up subsequent calls to `gclient`. To do this, set a `GIT_CACHE_PATH` environment variable:

```
$ export GIT_CACHE_PATH="${HOME}/.git_cache"
$ mkdir -p "${GIT_CACHE_PATH}"
# This will use about 16G.
```

Getting the code

```
$ mkdir electron && cd electron
$ gclient config --name "src/electron" --unmanaged https://github.com/electron/electron
$ gclient sync --with_branch_heads --with_tags
# This will take a while, go get a coffee.
```

> Instead of `https://github.com/electron/electron`, you can use your own fork here (something like `https://github.com/<username>/electron`).

A note on pulling/pushing

If you intend to `git pull` or `git push` from the official `electron` repository in the future, you now need to update the respective folder's origin URLs.

```
$ cd src/electron
$ git remote remove origin
$ git remote add origin https://github.com/electron/electron
$ git checkout main
$ git branch --set-upstream-to=origin/main
$ cd -
```

:memo: `gclient` works by checking a file called `DEPS` inside the `src/electron` folder for dependencies (like Chromium or Node.js). Running `gclient sync -f` ensures that all dependencies required to build Electron match that file.

So, in order to pull, you'd run the following commands:

```
$ cd src/electron
$ git pull
$ gclient sync -f
```

Building

Set the environment variable for chromium build tools

On Linux & MacOS

```
$ cd src
$ export CHROMIUM_BUILDTOOLS_PATH=`pwd`/buildtools
```

On Windows:

```
$ cd src
$ set CHROMIUM_BUILDTOOLS_PATH=%cd%\buildtools
```

To generate Testing build config of Electron:

```
$ gn gen out/Testing --args="import(\"//electron/build/args/testing.gn\")"
```

To generate Release build config of Electron:

```
$ gn gen out/Release --args="import(\"//electron/build/args/release.gn\")"
```

Note: This will generate a `out/Testing` or `out/Release` build directory under `src/` with the testing or release build depending upon the configuration passed above. You can replace `Testing|Release` with another names, but it should be a subdirectory of `out`.

Also you shouldn't have to run `gn gen` again—if you want to change the build arguments, you can run `gn args out/Testing` to bring up an editor. To see the list of available build configuration options, run `gn args out/Testing --list`.

To build, run `ninja` with the `electron` target: Note: This will also take a while and probably heat up your lap.

For the testing configuration:

```
$ ninja -C out/Testing electron
```

For the release configuration:

```
$ ninja -C out/Release electron
```

This will build all of what was previously 'libchromiumcontent' (i.e. the `content/` directory of `chromium` and its dependencies, incl. WebKit and V8), so it will take a while.

The built executable will be under `./out/Testing`:

```
$ ./out/Testing/Electron.app/Contents/MacOS/Electron
# or, on Windows
$ ./out/Testing/electron.exe
# or, on Linux
$ ./out/Testing/electron
```

Packaging

On linux, first strip the debugging and symbol information:

```
$ electron/script/strip-binaries.py -d out/Release
```

To package the electron build as a distributable zip file:

```
$ ninja -C out/Release electron:electron_dist_zip
```

Cross-compiling

To compile for a platform that isn't the same as the one you're building on, set the `target_cpu` and `target_os` GN arguments. For example, to compile an x86 target from an x64 host, specify `target_cpu = "x86"` in `gn args`.

```
$ gn gen out/Testing-x86 --args='... target_cpu = "x86"'
```

Not all combinations of source and target CPU/OS are supported by Chromium.

Host	Target	Status
Windows x64	Windows arm64	Experimental
Windows x64	Windows x86	Automatically tested
Linux x64	Linux x86	Automatically tested

If you test other combinations and find them to work, please update this document :)

See the GN reference for allowable values of `target_os` and `target_cpu`.

Windows on Arm (experimental) To cross-compile for Windows on Arm, follow Chromium's guide to get the necessary dependencies, SDK and libraries, then build with `ELECTRON_BUILDING_WOA=1` in your environment before running `gclient sync`.

```
set ELECTRON_BUILDING_WOA=1
gclient sync -f --with_branch_heads --with_tags
```

Or (if using PowerShell):

```
$env:ELECTRON_BUILDING_WOA=1
gclient sync -f --with_branch_heads --with_tags
```

Next, run `gn gen` as above with `target_cpu="arm64"`.

Tests

To run the tests, you'll first need to build the test modules against the same version of Node.js that was built as part of the build process. To generate build headers for the modules to compile against, run the following under `src/` directory.

```
$ ninja -C out/Testing third_party/electron_node:headers
```

You can now run the tests.

If you're debugging something, it can be helpful to pass some extra flags to the Electron binary:

```
$ npm run test -- \
  --enable-logging -g 'BrowserWindow module'
```

Sharing the git cache between multiple machines

It is possible to share the gclient git cache with other machines by exporting it as SMB share on linux, but only one process/machine can be using the cache at a time. The locks created by git-cache script will try to prevent this, but it may not work perfectly in a network.

On Windows, SMBv2 has a directory cache that will cause problems with the git cache script, so it is necessary to disable it by setting the registry key

`HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Lanmanworkstation\Parameters\DirectoryC`

to 0. More information: https://stackoverflow.com/a/9935126

This can be set quickly in powershell (ran as administrator):

```
New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Services\Lanmanworkstation\Parameters
```

Troubleshooting

gclient sync complains about rebase

If `gclient sync` is interrupted the git tree may be left in a bad state, leading to a cryptic message when running `gclient sync` in the future:

```
2> Conflict while rebasing this branch.
2> Fix the conflict and run gclient again.
2> See man git-rebase for details.
```

If there are no git conflicts or rebases in `src/electron`, you may need to abort a `git am` in `src`:

```
$ cd ../
$ git am --abort
$ cd electron
$ gclient sync -f
```

I'm being asked for a username/password for chromium-internal.googlesource.com

If you see a prompt for `Username for 'https://chrome-internal.googlesource.com':` when running `gclient sync` on Windows, it's probably because the `DEPOT_TOOLS_WIN_TOOLCHAIN` environment variable is not set to 0. Open `Control Panel` → `System and Security` → `System` → `Advanced system settings` and add a system variable `DEPOT_TOOLS_WIN_TOOLCHAIN` with value `0`. This tells `depot_tools` to use your locally installed version of Visual Studio (by default, `depot_tools` will try to download a Google-internal version that only Googlers have access to).