Build Instructions (Linux)

Follow the guidelines below for building **Electron itself** on Linux, 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.

Prerequisites

- At least 25GB disk space and 8GB RAM.
- Python 2.7.x. Some distributions like CentOS 6.x still use Python 2.6.x so you may need to check your Python version with `python -V`.

  Please also ensure that your system and Python version support at least TLS 1.2. For a quick test, run the following script:

  ```
  $ npx @electron/check-python-tls
  ```

  If the script returns that your configuration is using an outdated security protocol, use your system's package manager to update Python to the latest version in the 2.7.x branch. Alternatively, visit https://www.python.org/downloads/ for detailed instructions.
- Node.js. There are various ways to install Node. You can download source code from nodejs.org and compile it. Doing so permits installing Node on your own home directory as a standard user. Or try repositories such as NodeSource.
- clang 3.4 or later.
- Development headers of GTK 3 and libnotify.

On Ubuntu >= 20.04, install the following libraries:

```
$ sudo apt-get install build-essential clang libdbus-1-dev libgtk-3-dev \
                       libnotify-dev libasound2-dev libcap-dev \
                       libcups2-dev libxtst-dev \
                       libxss1 libnss3-dev gcc-multilib g++-multilib curl \
                       gperf bison python3-dbusmock openjdk-8-jre
```

On Ubuntu < 20.04, install the following libraries:

```
$ sudo apt-get install build-essential clang libdbus-1-dev libgtk-3-dev \
                       libnotify-dev libgnome-keyring-dev \
                       libasound2-dev libcap-dev libcups2-dev libxtst-dev \
                       libxss1 libnss3-dev gcc-multilib g++-multilib curl \
                       gperf bison python-dbusmock openjdk-8-jre
```

On RHEL / CentOS, install the following libraries:

```
$ sudo yum install clang dbus-devel gtk3-devel libnotify-devel \
                   libgnome-keyring-devel xorg-x11-server-utils libcap-devel \
```

```
                  cups-devel libXtst-devel alsa-lib-devel libXrandr-devel \
                  nss-devel python-dbusmock openjdk-8-jre
```

On Fedora, install the following libraries:

```
$ sudo dnf install clang dbus-devel gtk3-devel libnotify-devel \
                  libgnome-keyring-devel xorg-x11-server-utils libcap-devel \
                  cups-devel libXtst-devel alsa-lib-devel libXrandr-devel \
                  nss-devel python-dbusmock openjdk-8-jre
```

On Arch Linux / Manjaro, install the following libraries:

```
$ sudo pacman -Syu base-devel clang libdbus gtk2 libnotify \
                  libgnome-keyring alsa-lib libcap libcups libxtst \
                  libxss nss gcc-multilib curl gperf bison \
                  python2 python-dbusmock jdk8-openjdk
```

Other distributions may offer similar packages for installation via package managers such as pacman. Or one can compile from source code.

Cross compilation

If you want to build for an `arm` target you should also install the following dependencies:

```
$ sudo apt-get install libc6-dev-armhf-cross linux-libc-dev-armhf-cross \
                       g++-arm-linux-gnueabihf
```

Similarly for `arm64`, install the following:

```
$ sudo apt-get install libc6-dev-arm64-cross linux-libc-dev-arm64-cross \
                       g++-aarch64-linux-gnu
```

And to cross-compile for `arm` or `ia32` targets, you should pass the `target_cpu` parameter to `gn gen`:

```
$ gn gen out/Testing --args='import(...) target_cpu="arm"'
```

Building

See Build Instructions: GN

Troubleshooting

Error While Loading Shared Libraries: libtinfo.so.5

Prebuilt `clang` will try to link to `libtinfo.so.5`. Depending on the host architecture, symlink to appropriate `libncurses`:

```
$ sudo ln -s /usr/lib/libncurses.so.5 /usr/lib/libtinfo.so.5
```

Advanced topics

The default building configuration is targeted for major desktop Linux distributions. To build for a specific distribution or device, the following information may help you.

Using system `clang` instead of downloaded `clang` binaries

By default Electron is built with prebuilt `clang` binaries provided by the Chromium project. If for some reason you want to build with the `clang` installed in your system, you can specify the `clang_base_path` argument in the GN args.

For example if you installed `clang` under `/usr/local/bin/clang`:

```
$ gn gen out/Testing --args='import("//electron/build/args/testing.gn")
clang_base_path = "/usr/local/bin"'
```

Using compilers other than `clang`

Building Electron with compilers other than `clang` is not supported.