Advanced Installation Instructions

To install prebuilt Electron binaries, use `npm`. The preferred method is to install Electron as a development dependency in your app:

```
npm install electron --save-dev
```

See the Electron versioning doc for info on how to manage Electron versions in your apps.

Running Electron ad-hoc

If you're in a pinch and would prefer to not use `npm install` in your local project, you can also run Electron ad-hoc using the `npx` command runner bundled with `npm`:

```
npx electron .
```

The above command will run the current working directory with Electron. Note that any dependencies in your app will not be installed.

Customization

If you want to change the architecture that is downloaded (e.g., `ia32` on an `x64` machine), you can use the `--arch` flag with npm install or set the `npm_config_arch` environment variable:

```
npm install --arch=ia32 electron
```

In addition to changing the architecture, you can also specify the platform (e.g., `win32`, `linux`, etc.) using the `--platform` flag:

```
npm install --platform=win32 electron
```

Proxies

If you need to use an HTTP proxy, you need to set the `ELECTRON_GET_USE_PROXY` variable to any value, plus additional environment variables depending on your host system's Node version:

- Node 10 and above
- Before Node 10

Custom Mirrors and Caches

During installation, the `electron` module will call out to `@electron/get` to download prebuilt binaries of Electron for your platform. It will do so by contacting GitHub's release download page (`https://github.com/electron/electron/releases/tag/v$VERSION`, where `$VERSION` is the exact version of Electron).

If you are unable to access GitHub or you need to provide a custom build, you can do so by either providing a mirror or an existing cache directory.

Mirror

You can use environment variables to override the base URL, the path at which to look for Electron binaries, and the binary filename. The URL used by `@electron/get` is composed as follows:

```
url = ELECTRON_MIRROR + ELECTRON_CUSTOM_DIR + '/' + ELECTRON_CUSTOM_FILENAME
```

For instance, to use the China CDN mirror:

```
ELECTRON_MIRROR="https://npmmirror.com/mirrors/electron/"
```

By default, `ELECTRON_CUSTOM_DIR` is set to `v$VERSION`. To change the format, use the `{{ version }}` placeholder. For example, `version-{{ version }}` resolves to `version-5.0.0`, `{{ version }}` resolves to `5.0.0`, and `v{{ version }}` is equivalent to the default. As a more concrete example, to use the China non-CDN mirror:

```
ELECTRON_MIRROR="https://npmmirror.com/mirrors/electron/"
ELECTRON_CUSTOM_DIR="{{ version }}"
```

The above configuration will download from URLs such as `https://npmmirror.com/mirrors/electron/8.0.0/electron-v8.0.0-linux-x64.zip`.

If your mirror serves artifacts with different checksums to the official Electron release you may have to set `electron_use_remote_checksums=1` to force Electron to use the remote `SHASUMS256.txt` file to verify the checksum instead of the embedded checksums.

Cache

Alternatively, you can override the local cache. `@electron/get` will cache downloaded binaries in a local directory to not stress your network. You can use that cache folder to provide custom builds of Electron or to avoid making contact with the network at all.

- Linux: `$XDG_CACHE_HOME` or `~/.cache/electron/`
- macOS: `~/Library/Caches/electron/`
- Windows: `$LOCALAPPDATA/electron/Cache` or `~/AppData/Local/electron/Cache/`

On environments that have been using older versions of Electron, you might find the cache also in `~/.electron`.

You can also override the local cache location by providing a `electron_config_cache` environment variable.

The cache contains the version's official zip file as well as a checksum, and is stored as `[checksum]/[filename]`. A typical cache might look like this:

```
├── a91b089b5dc5b1279966511344b805ec84869b6cd60af44f800b363bba25b915
│   └── electron-v15.3.1-darwin-x64.zip
```

Skip binary download

Under the hood, Electron's JavaScript API binds to a binary that contains its implementations. Because this binary is crucial to the function of any Electron app, it is downloaded by default in the `postinstall` step every time you

install `electron` from the npm registry.

However, if you want to install your project's dependencies but don't need to use Electron functionality, you can set the `ELECTRON_SKIP_BINARY_DOWNLOAD` environment variable to prevent the binary from being downloaded. For instance, this feature can be useful in continuous integration environments when running unit tests that mock out the `electron` module.

```
ELECTRON_SKIP_BINARY_DOWNLOAD=1 npm install
```

Troubleshooting

When running `npm install electron`, some users occasionally encounter installation errors.

In almost all cases, these errors are the result of network problems and not actual issues with the `electron` npm package. Errors like `ELIFECYCLE`, `EAI_AGAIN`, `ECONNRESET`, and `ETIMEDOUT` are all indications of such network problems. The best resolution is to try switching networks, or wait a bit and try installing again.

You can also attempt to download Electron directly from electron/electron/releases if installing via `npm` is failing.

If installation fails with an `EACCESS` error you may need to fix your npm permissions.

If the above error persists, the unsafe-perm flag may need to be set to true:

```
sudo npm install electron --unsafe-perm=true
```

On slower networks, it may be advisable to use the `--verbose` flag in order to show download progress:

```
npm install --verbose electron
```

If you need to force a re-download of the asset and the SHASUM file set the `force_no_cache` environment variable to `true`.