Coding Style

These are the style guidelines for coding in Electron.

You can run `npm run lint` to show any style issues detected by `cpplint` and `eslint`.

General Code

- End files with a newline.
- Place requires in the following order:
  - Built in Node Modules (such as `path`)
  - Built in Electron Modules (such as `ipc`, `app`)
  - Local Modules (using relative paths)
- Place class properties in the following order:
  - Class methods and properties (methods starting with a `@`)
  - Instance methods and properties
- Avoid platform-dependent code:
  - Use `path.join()` to concatenate filenames.
  - Use `os.tmpdir()` rather than `/tmp` when you need to reference the temporary directory.
- Using a plain `return` when returning explicitly at the end of a function.
  - Not `return null`, `return undefined`, `null` or `undefined`

C++ and Python

For C++ and Python, we follow Chromium's Coding Style. There is also a script `script/cpplint.py` to check whether all files conform.

The Python version we are using now is Python 2.7.

The C++ code uses a lot of Chromium's abstractions and types, so it's recommended to get acquainted with them. A good place to start is Chromium's Important Abstractions and Data Structures document. The document mentions some special types, scoped types (that automatically release their memory when going out of scope), logging mechanisms etc.

Documentation

- Write remark markdown style.

You can run `npm run lint-docs` to ensure that your documentation changes are formatted correctly.

JavaScript

- Write standard JavaScript style.
- File names should be concatenated with `-` instead of `_`, e.g. `file-name.js` rather than `file_name.js`, because in github/atom module names are usually in the `module-name` form. This rule only applies to `.js` files.
- Use newer ES6/ES2015 syntax where appropriate
  - `const` for requires and other constants. If the value is a primitive, use uppercase naming (eg `const NUMBER_OF_RETRIES = 5`).
  - `let` for defining variables

    - Arrow functions instead of `function () { }`
    - Template literals instead of string concatenation using `+`

Naming Things

Electron APIs uses the same capitalization scheme as Node.js:

- When the module itself is a class like `BrowserWindow`, use `PascalCase`.
- When the module is a set of APIs, like `globalShortcut`, use `camelCase`.
- When the API is a property of object, and it is complex enough to be in a separate chapter like `win.webContents`, use `mixedCase`.
- For other non-module APIs, use natural titles, like `<webview> Tag` or `Process Object`.

When creating a new API, it is preferred to use getters and setters instead of jQuery's one-function style. For example, `.getText()` and `.setText(text)` are preferred to `.text([text])`. There is a discussion on this.