Using clang-tidy on C++ Code

`clang-tidy` is a tool to automatically check C/C++/Objective-C code for style violations, programming errors, and best practices.

Electron's `clang-tidy` integration is provided as a linter script which can be run with `npm run lint:clang-tidy`. While `clang-tidy` checks your on-disk files, you need to have built Electron so that it knows which compiler flags were used. There is one required option for the script `--output-dir`, which tells the script which build directory to pull the compilation information from. A typical usage would be: `npm run lint:clang-tidy --out-dir ../out/Testing`

With no filenames provided, all C/C++/Objective-C files will be checked. You can provide a list of files to be checked by passing the filenames after the options: `npm run lint:clang-tidy --out-dir ../out/Testing shell/browser/api/electron_api_app.cc`

While `clang-tidy` has a long list of possible checks, in Electron only a few are enabled by default. At the moment Electron doesn't have a `.clang-tidy` config, so `clang-tidy` will find the one from Chromium at `src/.clang-tidy` and use the checks which Chromium has enabled. You can change which checks are run by using the `--checks=` option. This is passed straight through to `clang-tidy`, so see its documentation for full details. Wildcards can be used, and checks can be disabled by prefixing a `-`. By default any checks listed are added to those in `.clang-tidy`, so if you'd like to limit the checks to specific ones you should first exclude all checks then add back what you want, like `--checks=-*,performance*`.

Running `clang-tidy` is rather slow - internally it compiles each file and then runs the checks so it will always be some factor slower than compilation. While you can use parallel runs to speed it up using the `--jobs|-j` option, `clang-tidy` also uses a lot of memory during its checks, so it can easily run into out-of-memory errors. As such the default number of jobs is one.