Patches in Electron

Electron is built on two major upstream projects: Chromium and Node.js. Each of these projects has several of their own dependencies, too. We try our best to use these dependencies exactly as they are but sometimes we can't achieve our goals without patching those upstream dependencies to fit our use cases.

Patch justification

Every patch in Electron is a maintenance burden. When upstream code changes, patches can break—sometimes without even a patch conflict or a compilation error. It's an ongoing effort to keep our patch set up-to-date and effective. So we strive to keep our patch count at a minimum. To that end, every patch must describe its reason for existence in its commit message. That reason must be one of the following:

1. The patch is temporary, and is intended to be (or has been) committed upstream or otherwise eventually removed. Include a link to an upstream PR or code review if available, or a procedure for verifying whether the patch is still needed at a later date.
2. The patch allows the code to compile in the Electron environment, but cannot be upstreamed because it's Electron-specific (e.g. patching out references to Chrome's `Profile`). Include reasoning about why the change cannot be implemented without a patch (e.g. by subclassing or copying the code).
3. The patch makes Electron-specific changes in functionality which are fundamentally incompatible with upstream.

In general, all the upstream projects we work with are friendly folks and are often happy to accept refactorings that allow the code in question to be compatible with both Electron and the upstream project. (See e.g. this change in Chromium, which allowed us to remove a patch that did the same thing, or this change in Node, which was a no-op for Node but fixed a bug in Electron.) **We should aim to upstream changes whenever we can, and avoid indefinite-lifetime patches**.

Patch system

If you find yourself in the unfortunate position of having to make a change which can only be made through patching an upstream project, you'll need to know how to manage patches in Electron.

All patches to upstream projects in Electron are contained in the `patches/` directory. Each subdirectory of `patches/` contains several patch files, along with a `.patches` file which lists the order in which the patches should be applied. Think of these files as making up a series of git commits that are applied on top of the upstream project after we check it out.

```
patches
├── config.json   <-- this describes which patchset directory is applied to what
project
├── chromium
│   ├── .patches
│   ├── accelerator.patch
│   ├── add_contentgpuclient_precreatemessageloop_callback.patch
│   ⋮
├── node
│   ├── .patches
│   ├── add_openssl_is_boringssl_guard_to_oaep_hash_check.patch
│   ├── build_add_gn_build_files.patch
```

```
|   ⋮
⋮
```

To help manage these patch sets, we provide two tools: `git-import-patches` and `git-export-patches`. `git-import-patches` imports a set of patch files into a git repository by applying each patch in the correct order and creating a commit for each one. `git-export-patches` does the reverse; it exports a series of git commits in a repository into a set of files in a directory and an accompanying `.patches` file.

> *Side note: the reason we use a `.patches` file to maintain the order of applied patches, rather than prepending a number like `001-` to each file, is because it reduces conflicts related to patch ordering. It prevents the situation where two PRs both add a patch at the end of the series with the same numbering and end up both getting merged resulting in a duplicate identifier, and it also reduces churn when a patch is added or deleted in the middle of the series.*

Usage

Adding a new patch

```
$ cd src/third_party/electron_node
$ vim some/code/file.cc
$ git commit
$ ../../electron/script/git-export-patches -o ../../electron/patches/node
```

> ***NOTE**: `git-export-patches` ignores any uncommitted files, so you must create a commit if you want your changes to be exported. The subject line of the commit message will be used to derive the patch file name, and the body of the commit message should include the reason for the patch's existence.*

Re-exporting patches will sometimes cause shasums in unrelated patches to change. This is generally harmless and can be ignored (but go ahead and add those changes to your PR, it'll stop them from showing up for other people).

Editing an existing patch

```
$ cd src/v8
$ vim some/code/file.cc
$ git log
# Find the commit sha of the patch you want to edit.
$ git commit --fixup [COMMIT_SHA]
$ git rebase --autosquash -i [COMMIT_SHA]^
$ ../electron/script/git-export-patches -o ../electron/patches/v8
```

Removing a patch

```
$ vim src/electron/patches/node/.patches
# Delete the line with the name of the patch you want to remove
$ cd src/third_party/electron_node
$ git reset --hard refs/patches/upstream-head
$ ../../electron/script/git-import-patches ../../electron/patches/node
$ ../../electron/script/git-export-patches -o ../../electron/patches/node
```

Note that `git-import-patches` will mark the commit that was `HEAD` when it was run as `refs/patches/upstream-head`. This lets you keep track of which commits are from Electron patches (those that come after `refs/patches/upstream-head`) and which commits are in upstream (those before `refs/patches/upstream-head`).

Resolving conflicts

When updating an upstream dependency, patches may fail to apply cleanly. Often, the conflict can be resolved automatically by git with a 3-way merge. You can instruct `git-import-patches` to use the 3-way merge algorithm by passing the `-3` argument:

```
$ cd src/third_party/electron_node
# If the patch application failed midway through, you can reset it with:
$ git am --abort
# And then retry with 3-way merge:
$ ../../electron/script/git-import-patches -3 ../../electron/patches/node
```

If `git-import-patches -3` encounters a merge conflict that it can't resolve automatically, it will pause and allow you to resolve the conflict manually. Once you have resolved the conflict, `git add` the resolved files and continue to apply the rest of the patches by running `git am --continue`.