Pull Requests

- Setting up your local environment
  - Step 1: Fork
  - Step 2: Build
  - Step 3: Branch
- Making Changes
  - Step 4: Code
  - Step 5: Commit
    - Commit message guidelines
  - Step 6: Rebase
  - Step 7: Test
  - Step 8: Push
  - Step 9: Opening the Pull Request
  - Step 10: Discuss and Update
    - Approval and Request Changes Workflow
  - Step 11: Landing
  - Continuous Integration Testing

Setting up your local environment

Step 1: Fork

Fork the project on GitHub and clone your fork locally.

```
$ git clone git@github.com:username/electron.git
$ cd electron
$ git remote add upstream https://github.com/electron/electron.git
$ git fetch upstream
```

Step 2: Build

Build steps and dependencies differ slightly depending on your operating system. See these detailed guides on building Electron locally:

- Building on macOS
- Building on Linux
- Building on Windows

Once you've built the project locally, you're ready to start making changes!

Step 3: Branch

To keep your development environment organized, create local branches to hold your work. These should be branched directly off of the `main` branch.

```
$ git checkout -b my-branch -t upstream/main
```

Making Changes

Step 4: Code

Most pull requests opened against the `electron/electron` repository include changes to either the C/C++ code in the `shell/` folder, the JavaScript code in the `lib/` folder, the documentation in `docs/api/` or tests in the `spec/` folder.

Please be sure to run `npm run lint` from time to time on any code changes to ensure that they follow the project's code style.

See coding style for more information about best practice when modifying code in different parts of the project.

Step 5: Commit

It is recommended to keep your changes grouped logically within individual commits. Many contributors find it easier to review changes that are split across multiple commits. There is no limit to the number of commits in a pull request.

```
$ git add my/changed/files
$ git commit
```

Note that multiple commits get squashed when they are landed.

Commit message guidelines

A good commit message should describe what changed and why. The Electron project uses semantic commit messages to streamline the release process.

Before a pull request can be merged, it **must** have a pull request title with a semantic prefix.

Examples of commit messages with semantic prefixes:

- `fix: don't overwrite prevent_default if default wasn't prevented`
- `feat: add app.isPackaged() method`
- `docs: app.isDefaultProtocolClient is now available on Linux`

Common prefixes:

- fix: A bug fix
- feat: A new feature
- docs: Documentation changes
- test: Adding missing tests or correcting existing tests
- build: Changes that affect the build system
- ci: Changes to our CI configuration files and scripts
- perf: A code change that improves performance
- refactor: A code change that neither fixes a bug nor adds a feature
- style: Changes that do not affect the meaning of the code (linting)

Other things to keep in mind when writing a commit message:

1. The first line should:
   - contain a short description of the change (preferably 50 characters or less, and no more than 72 characters)
   - be entirely in lowercase with the exception of proper nouns, acronyms, and the words that refer to code, like function/variable names

2. Keep the second line blank.
3. Wrap all other lines at 72 columns.

Breaking Changes

A commit that has the text `BREAKING CHANGE:` at the beginning of its optional body or footer section introduces a breaking API change (correlating with Major in semantic versioning). A breaking change can be part of commits of any type. e.g., a `fix:`, `feat:` & `chore:` types would all be valid, in addition to any other type.

See conventionalcommits.org for more details.

Step 6: Rebase

Once you have committed your changes, it is a good idea to use `git rebase` (not `git merge`) to synchronize your work with the main repository.

```
$ git fetch upstream
$ git rebase upstream/main
```

This ensures that your working branch has the latest changes from `electron/electron` main.

Step 7: Test

Bug fixes and features should always come with tests. A testing guide has been provided to make the process easier. Looking at other tests to see how they should be structured can also help.

Before submitting your changes in a pull request, always run the full test suite. To run the tests:

```
$ npm run test
```

Make sure the linter does not report any issues and that all tests pass. Please do not submit patches that fail either check.

If you are updating tests and want to run a single spec to check it:

```
$ npm run test -match=menu
```

The above would only run spec modules matching `menu`, which is useful for anyone who's working on tests that would otherwise be at the very end of the testing cycle.

Step 8: Push

Once your commits are ready to go -- with passing tests and linting -- begin the process of opening a pull request by pushing your working branch to your fork on GitHub.

```
$ git push origin my-branch
```

Step 9: Opening the Pull Request

From within GitHub, opening a new pull request will present you with a template that should be filled out. It can be found here.

If you do not adequately complete this template, your PR may be delayed in being merged as maintainers seek more information or clarify ambiguities.

Step 10: Discuss and update

You will probably get feedback or requests for changes to your pull request. This is a big part of the submission process so don't be discouraged! Some contributors may sign off on the pull request right away. Others may have detailed comments or feedback. This is a necessary part of the process in order to evaluate whether the changes are correct and necessary.

To make changes to an existing pull request, make the changes to your local branch, add a new commit with those changes, and push those to your fork. GitHub will automatically update the pull request.

```
$ git add my/changed/files
$ git commit
$ git push origin my-branch
```

There are a number of more advanced mechanisms for managing commits using `git rebase` that can be used, but are beyond the scope of this guide.

Feel free to post a comment in the pull request to ping reviewers if you are awaiting an answer on something. If you encounter words or acronyms that seem unfamiliar, refer to this glossary.

Approval and Request Changes Workflow

All pull requests require approval from a Code Owner of the area you modified in order to land. Whenever a maintainer reviews a pull request they may request changes. These may be small, such as fixing a typo, or may involve substantive changes. Such requests are intended to be helpful, but at times may come across as abrupt or unhelpful, especially if they do not include concrete suggestions on *how* to change them.

Try not to be discouraged. If you feel that a review is unfair, say so or seek the input of another project contributor. Often such comments are the result of a reviewer having taken insufficient time to review and are not ill-intended. Such difficulties can often be resolved with a bit of patience. That said, reviewers should be expected to provide helpful feedback.

Step 11: Landing

In order to land, a pull request needs to be reviewed and approved by at least one Electron Code Owner and pass CI. After that, if there are no objections from other contributors, the pull request can be merged.

Congratulations and thanks for your contribution!

Continuous Integration Testing

Every pull request is tested on the Continuous Integration (CI) system to confirm that it works on Electron's supported platforms.

Ideally, the pull request will pass ("be green") on all of CI's platforms. This means that all tests pass and there are no linting errors. However, it is not uncommon for the CI infrastructure itself to fail on specific platforms or for so-called "flaky" tests to fail ("be red"). Each CI failure must be manually inspected to determine the cause.

CI starts automatically when you open a pull request, but only core maintainers can restart a CI run. If you believe CI is giving a false negative, ask a maintainer to restart the tests.