# ?????????? Contributing to isomorphic-git ??????????

Oh wow! Thanks for opening up the contributing file! :grin: :tada:

You are very welcome here and any contribution is appreciated. :+1:

# Tips

The code is written in "plain" JavaScript and as a rule of thumb shouldn't require transpilation. (The glaring exception being browser's lack of support for bare imports.)

## New feature checklists :sparkles:?

I'm honestly documenting these steps just so I don't forget them myself.

To add a parameter to an existing command `X`:

- [ ] add parameter to the function in `src/api/X.js` (and `src/commands/X.js` if necessary)

- [ ] document the parameter in the JSDoc comment above the function

- [ ] add a test case in `\_\_tests\_\_/test-X.js` if possible

- [ ] if this is your first time contributing, run `npm run add-contributor` and follow the prompts to add yourself to the README

- [ ] squash merge the PR with commit message "feat(X): Added 'bar' parameter"

To create a new command:

- [ ] add as a new file in `src/api` (and `src/commands` if necessary)

- [ ] add command to `src/index.js`

- [ ] update `\_\_tests\_\_/\_\_snapshots\_\_/test-exports.js.snap`

- [ ] create a test in `src/\_\_tests\_\_`

- [ ] document the command with a JSDoc comment

- [ ] add page to the Docs Sidebar `website/sidebars.json`

- [ ] if this is your first time contributing, run `npm run add-contributor` and follow the prompts to add yourself to the README

- [ ] squash merge the PR with commit message "feat: Added 'X' command"

# Overview

I have written this library as a series of layers that build upon one another and should tree-shake very well:

## commands

Each command is available as its own file, so you are able to import individual commands

if you only need a few in order to optimize your bundle size.

## managers

Managers are a level above models. They take care of implementation performance details like

- batching reads to and from the file system

- in-process concurrency locks

- lockfiles

- caching files and invalidating cached results

- reusing objects

- object memory pools

## everything else

These are the lowest level building blocks. They tend to be small, pure functions.

### models

Models generally have very few or no dependencies except for `'buffer'`.

This makes them portable to many different environments so they can be a useful lowest common denominator.

### utils

Utils are basically miscellaneous functions.

### storage

This folder contains code for reading and writing to the git "object store".

I'm hoping I can abstract it into a plugin interface at some point so that the plugin system can provide

alternative object stores that integrate seamlessly.

### wire

This folder contains the parsers and serializers for the Git wire protocol.

For a given thing, like an upload-pack command, there can be up to 4 different functions.

Client:

write[\*]Request: (input: Object) -> stream

parse[\*]Response: (input: stream) -> Object

Server:

parse[\*]Request: (input: stream) -> Object

write[\*]Response: (input: Object) -> stream