Intro to Client-Side Git Hooks (npm specific)

Dario Piotrowicz

April 22, 2021

Git Hooks

What Git Hooks Are

From the official git docs:

Definition

Hooks are programs you can place in a hooks directory to trigger actions at certain points in git's execution. Hooks that don't have the executable bit set are ignored.

Hooks are divided in:

- **Client-Side**
  Hooks executed on the committer's computer
- **Server-Side**
  Hooks executed on the server when receiving pushes

We'll just focus on the client-side here

Client-Side Git Hooks

The most important and useful client-side hooks are:

- **pre-commit**
  Invoked whem making a commit (before the editor opens for the commit message), it can modify the changes and prevent the commit by exiting with a non-zero value
- **pre-push**
  Invoked when pushing to remote, can be used to perform checks (by the way, the remote destination is provided as parameter to the hook) and prevent the push by exiting with a non-zero value
- **commit-msg**
  Invoked when committing or merging, it receives the name of the file that holds the proposed commit log message and can modify it, can also prevent the commit/merge by exiting with a non-zero value
- **prepare-commit-msg**
  Invoked when commiting, it's purpose is to edit the default commit message that is proposed to the committer, just like the others can abort the commit by exiting with a non-zero value

You can find the list of all the git hooks supported in the official git docs

Tools

Husky

One of the most popular ways to implement git hooks is by using husky

But:

- **Husky v.4 adds overhead and has dependencies**
  Based on it's implementation there are extra checks and overheads than necessary
- **Husky v.5 has a Parity License**
  Later the license should change to MIT, but currently uses the Parity License preventing its usage in non open source projects
- **Is Unnecessary**
  Both v.4 and v.5 are quite unnecessary, everything they do can be done with practically the same effort natively

So...

Native Git Hooks

Writing git hooks in the native way is very easy and doesn't have drawbacks

For that all you need to do is writing a script file which name corresponds to the hook that you're implementing and add it to the project's hook directory

Let's just see how it is done with an example, let's implement the **pre-commit** hook

Native Git Hooks - pre-commit 1

We can just create this file:

```
#!/bin/sh
echo "Hello World of Hooks";
```

Make it executable, place it in the **.git/hooks** directory of our project and that is actually it!

If we now try to commit some code we will just be presented with the *Hello World of Hooks* line (and everything will keep working as normal)

Is this enough? can you spot a problem with this approach?

Native Git Hooks - pre-commit 2

There's a significant problem here, the **.git** directory is external from the git flow and doesn't get committed, so basically this hook will only work on your personal machine and no one pulling down will notice anything different

The solution is simply to move the hook in a directory which will git will keep track of like for example a **git-hooks** directory

But now we do need to tell git to consider this one the new directory containing our project's hooks and we can do it with:

```
git config core.hooksPath ./git-hooks
```

Native Git Hooks - pre-commit 2

This is ok and the hooks will be part of the repository, but in order for the committer to use them they will have to run the config command

So let's add a new script to out *package.json* so that this will be done automatically after every installation:

```
"scripts": {
  ...,
  "postinstall": "git config core.hooksPath ./git-hooks",
  ...
}
```

Now everything is fine and as soon as a committer pulls down the code and runs ‘npm install‘ the hooks will be set for them

Examples

Using the Pre-commit hook with Prettier

A popular use of the pre-commit hook is to format your code using prettier when commits are made

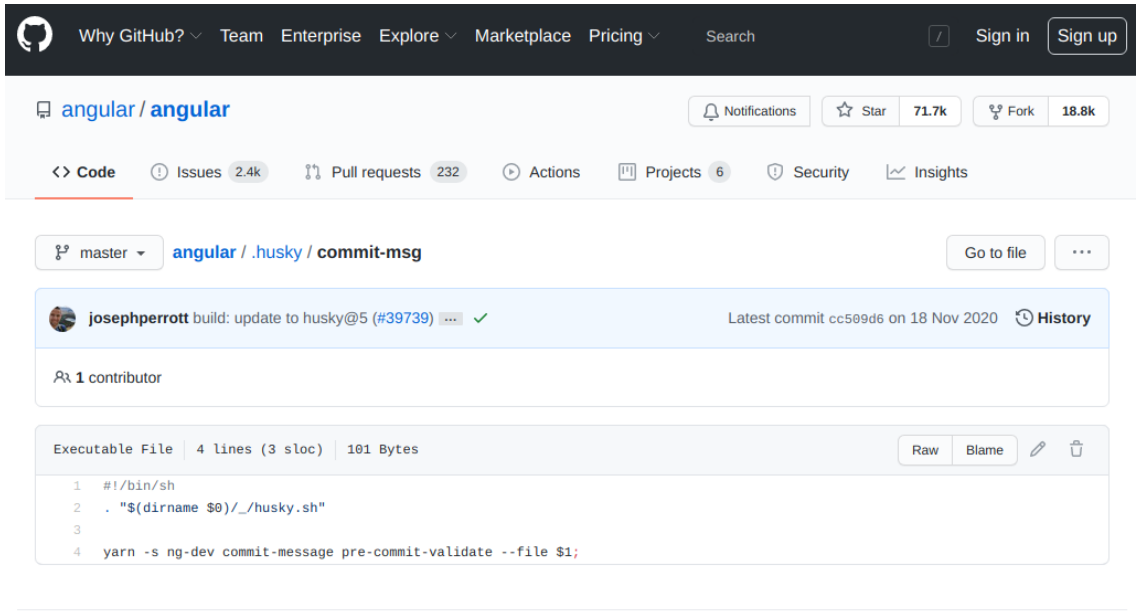
Prettier also does specifies this in its official docs

Tools such as pretty-quick and lint-staged are very popular and allow you to format and lint only staged files with close to no effort

Angular's Commit Message Format

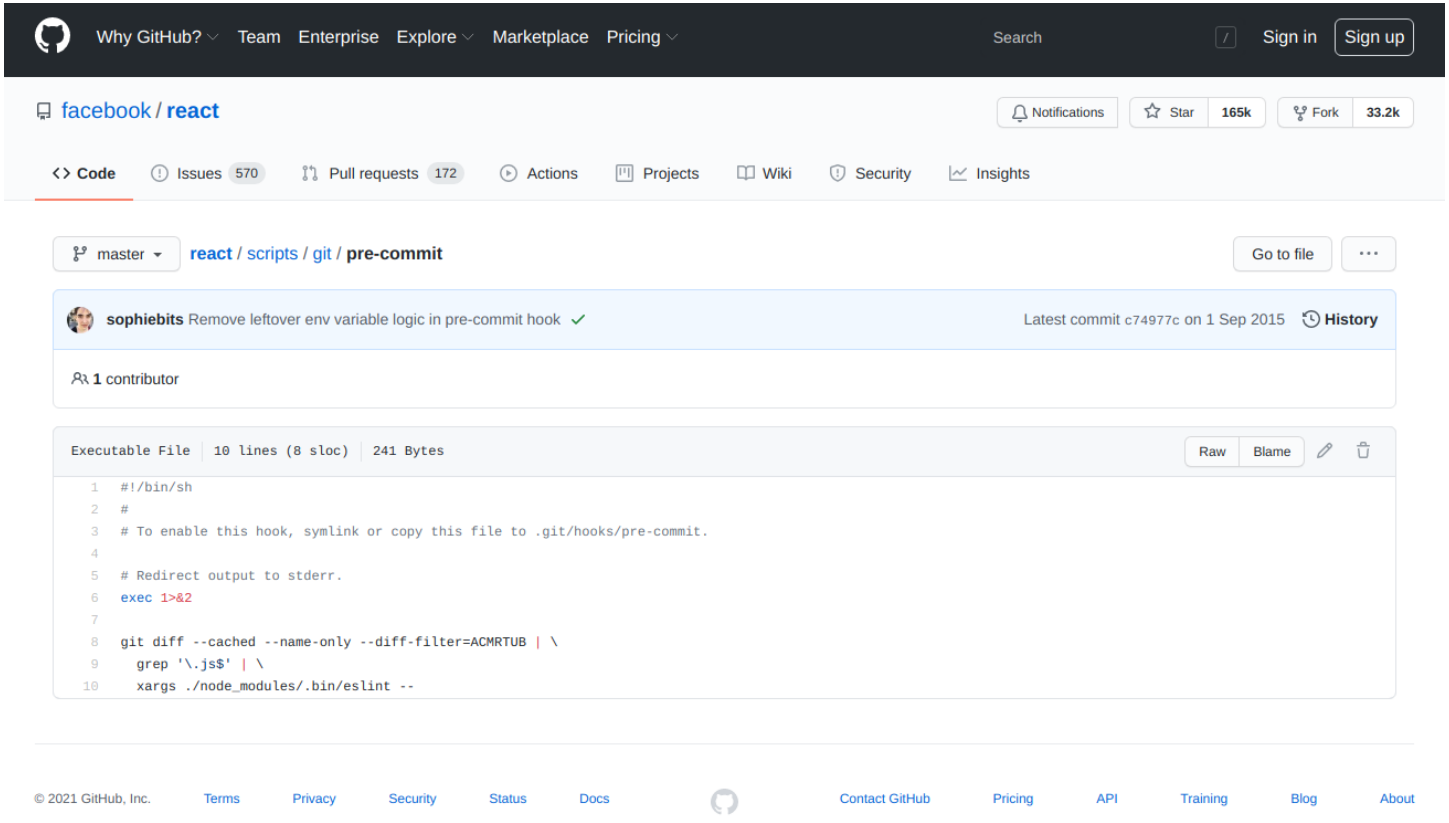
The Angular repo requires a specific commit message format

In order to "enforce" it they have a (husky) **commit-msg** set up as you can see here:



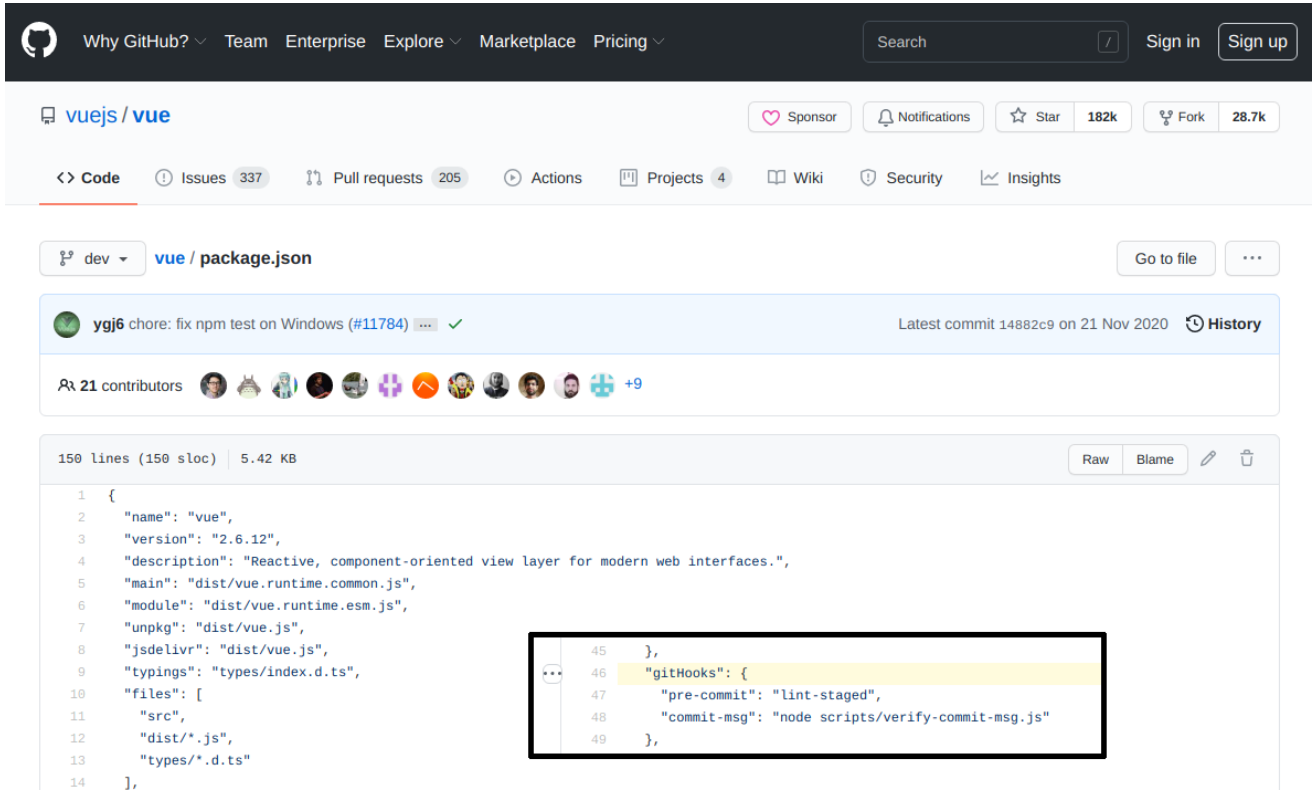
React's pre-commit linting hook

React has a **pre-commit** hook to eslint the staged *js* files



Vue's package.json gitHooks

The Vue repo implements both a **pre-commit** and **commit-msg** in its package.json using a fork of husky v4 (called yorkie):



Npm Examples

I've also created a repository with some git hooks implemented with the npm strategy presented earlier

All you need to know is in the repo's README

If you want you can clone the repository and check out the examples, or use them as reference:

https://github.com/dario-piotrowicz/npm-client-side-git-hooks-examples