Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Test-Driven Development and Environments with JavaScript (/intermediate-javascript/test-driven-development-and-environments-with-javascript) / Git Best Practices and Adding a .gitignore File

Text

The very first step in creating any project that uses multiple JS dependencies that are managed by node package manager (npm) is to create a `.gitignore` file, and we'll do just that in this lesson. We'll also take the opportunity to review a few additional Git best practices. You are expected to incorporate all of these best practices into your projects for the remainder of the Intermediate JavaScript course.

Adding a `.gitignore` File

We briefly discussed `.gitignore` files in Git Configurations (https://www.learnhowtoprogram.com/introduction-to-programming/getting-started-with-intro-to-programming/git-configurations). That lesson discussed setting up a `.gitignore_global` file which we can use to ensure that git ignores files we don't want to commit to GitHub.

Just like with `.gitignore_global`, `.gitignore` files aren't specific to project written in JavaScript. In general, they should be included in all projects being pushed to GitHub, and we'll continue to use `.gitignore` files for the remained of the program.

A `.gitignore` file lists all of the files that are *local* to a project that Git should not push to GitHub. A `.gitignore` always goes in the top level of the project directory, which is also called the project's 'root'. The **root** of a project is the folder that is the parent for all the project files and subfolders.

In order for a `.gitignore` file to work correctly, it *must* be committed *before* we commit (by accident) any code we don't want in our Git history.

Let's create a `.gitignore` file now and list a few files and folders in it. When file(s) or folder(s) are listed in the `.gitignore`, then that directs Git to ignore them when you make commits.

.gitignore

```
node_modules/
.DS_Store // only include this if you are on a Mac
dist/
```

We add the `node_modules/` and `dist/` folders to `.gitignore` because these are both automatically generated by npm and webpack, respectively. Since we've only just been introduced to these folders in the last lesson, we'll revisit why we add these folders to the `.gitignore` when we learn how they are generated.

As previously mentioned, Git will only ignore the files and folders inside of the `.gitignore` if we commit `.gitignore` to our project's Git history ***before*** we add those folders/files inside of the `.gitignore` to our project.

Let's go ahead and make a commit. Set up the repository on your own if you haven't ready. Now we'll make our commit:

```
$ git add .gitignore
$ git commit -m "add .gitignore file to project"
$ git push origin main
```

Note that we didn't do `$ git add .` . That's because our project directory has other files already. And if our project already had a `node_modules` and `dist` folder when we went to commit out `.gitignore` , then we'd be adding those files to GitHub along with our `.gitignore` file. This will cause Git to not be able to ignore those files.

This is like putting up a sign that says "Wet Paint" on a bench *after* someone has already sat on the bench by accident. Then we'd have a mess to clean up. GitHub absolutely needs to know what files should be ignored *before* they (usually accidentally) get pushed.

That's why we need to be diligent about setting up the `.gitignore` first thing, and also committing `.gitignore` without any other file with `$ git add .gitignore` .

Once our `.gitignore` file has been pushed to GitHub, we can make further commits without worrying about accidentally pushing files that are listed in our `.gitignore` .

From now on, you should always include a `.gitignore` file in the top level directory of your projects, not just in this JavaScript course but in future courses as well. This goes for C#, Ruby, and React projects, too. Your first commit should generally include your `.gitignore` file. Once again, make sure to avoid pushing any files that you want to ignore when you make that first commit — because GitHub won't know they should be ignored yet.

Removing a Committed File that Should Be Ignored

If you do accidentally add a file that should be ignored to your repository, you'll need to remove it. You can do so by running `$ git rm -r --cached [FILE-NAME]`, where `[FILE-NAME]` is the name of the file that shouldn't be tracked. Git will no longer track the file.

However, this won't entirely fix the problem if you are working with sensitive data. We won't be working with that kind of data in this section, but we will do so once we start working with APIs in the next section. Sensitive data often includes a private key that should never be shared with others. (It's similar to a password.) Just running `$ git rm -r --cached [FILE-NAME]` won't fix the issue. While GitHub will stop tracking the file, it won't remove the file from your Git history. That's a big problem — a hacker could still get that information.

If you find yourself needing to remove a file from a repository's history, see Removing Files from a Repository's History (https://help.github.com/en/github/managing-large-files/removing-files-from-a-repositorys-history) on GitHub's documentation.

In future lessons, we will specify which files and directories shouldn't be tracked. These files should be added to the local `.gitignore` file for all future projects.

While you could add these files to your `.gitignore_global` file, it's still important to have the local `.gitignore` file. If other developers push to the project later, or if you push to it from a different machine, you'll automatically use the local `.gitignore` file.

Git Best Practices

Let's take another look at our commit above. Instead of using `$ git add .`, we specified the name of the file we wanted to commit. In the real world, using `$ git add .` is often a bad idea. It's generally better to add files one at a time and push code that's both ready to be committed (in other words, working, not broken code) that's relevant to the commit you're working on.

Let's demonstrate why with an example. In the snippet below, we run `$ git status` to look at an imaginary file.

```
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committe
d)
  (use "git checkout -- <file>..." to discard changes in wo
rking directory)

    modified:   ready-to-commit.js
  modified:   really-yucky-code.js
  modified:   most-embarrassing-elementary-school-secrets.m
d

no changes added to commit (use "git add" and/or "git commi
t -a")
```

Here we have three files:

- The first, `ready-to-commit.js`, is ready to be committed and pushed.
- The second, `really-yucky-code.js`, isn't ready yet and shouldn't be committed.
- The third should *never* be pushed to GitHub. You'd never live it down!

Obviously, `$ git add .` isn't the answer here. Instead, we should just run `$ git add ready-to-commit.js`. And if we want to commit multiple files, we could do so by passing in multiple file names like this: `$ git add file1 file2 file3`.

Checking Changes with `$ git diff`

We could also verify all the changes we've made in `ready-to-commit.js` since our last commit by using the `$ git diff` command. If we run `$ git diff ready-to-commit.js`, we'll see a list of these changes. It's generally a good idea to do a quick `git diff` before committing files. That way, we can verify we're committing the right code.

If we want to be thorough and follow best practices, we should do the following when we commit *instead* of using `$ git add .`:

- First, get the `$ git status`.
- Use `$ git diff [FILE-NAME]` to verify that the changes look correct.
- Use `$ git add [FILE-NAME]` to select a file to commit. Multiple files can be committed at once *if* they are related to the same feature — but should be added one by one by doing `$ git add file1 file2 etc`.
- Run `$ git commit` and add a great commit message.

While it's often convenient to run `$ git add .`, especially for learning projects, we recommend practicing good habits with your git workflow now.

Previous (/intermediate-javascript/test-driven-development-and-environments-with-javascript/future-project-structure)
Next (/intermediate-javascript/test-driven-development-and-environments-with-javascript/creating-a-package-json-with-npm)

Lesson 5 of 49
Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)