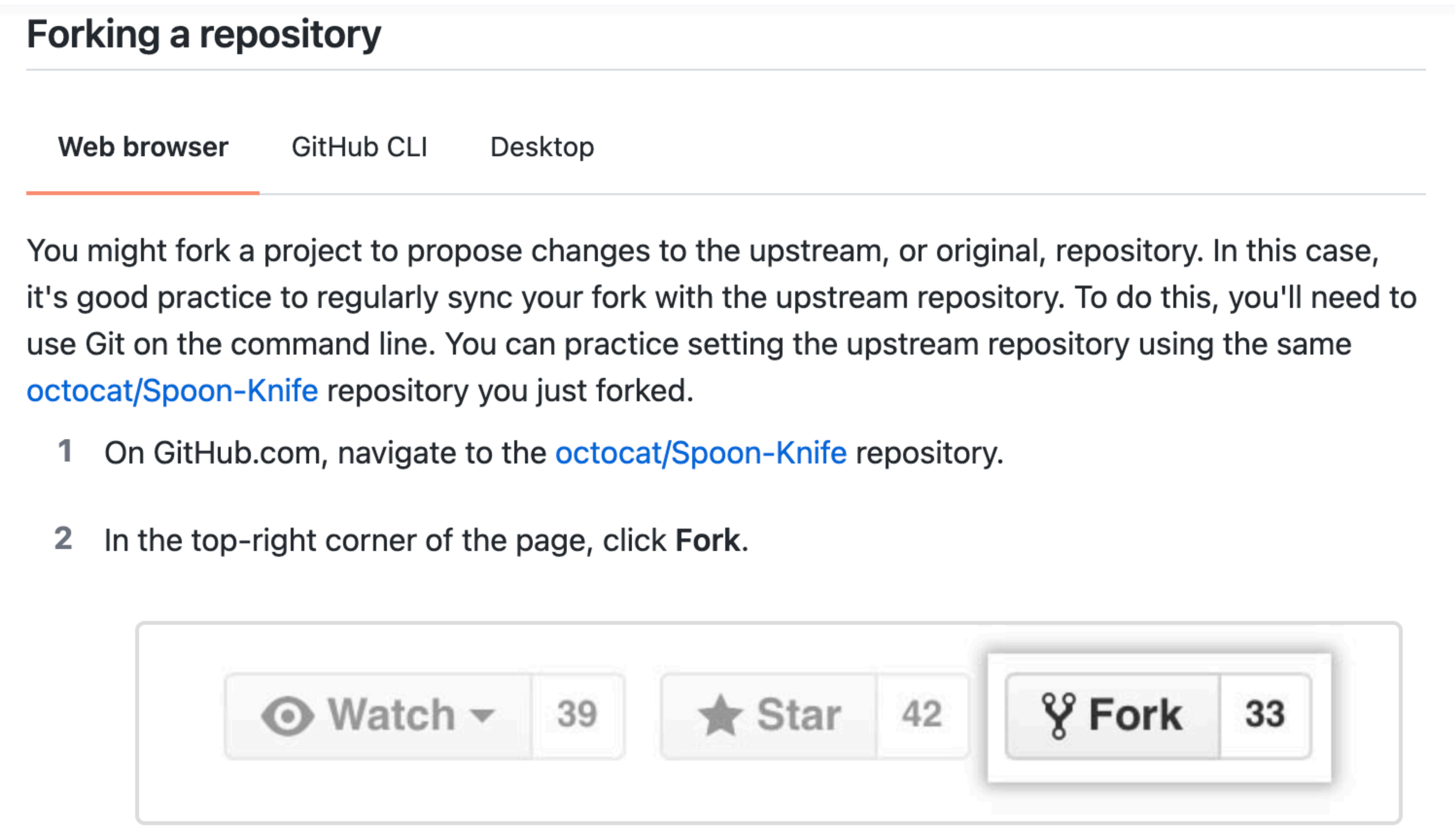
Git part 2

Roman Fedytskyi JSP React competency lead

Git Fork



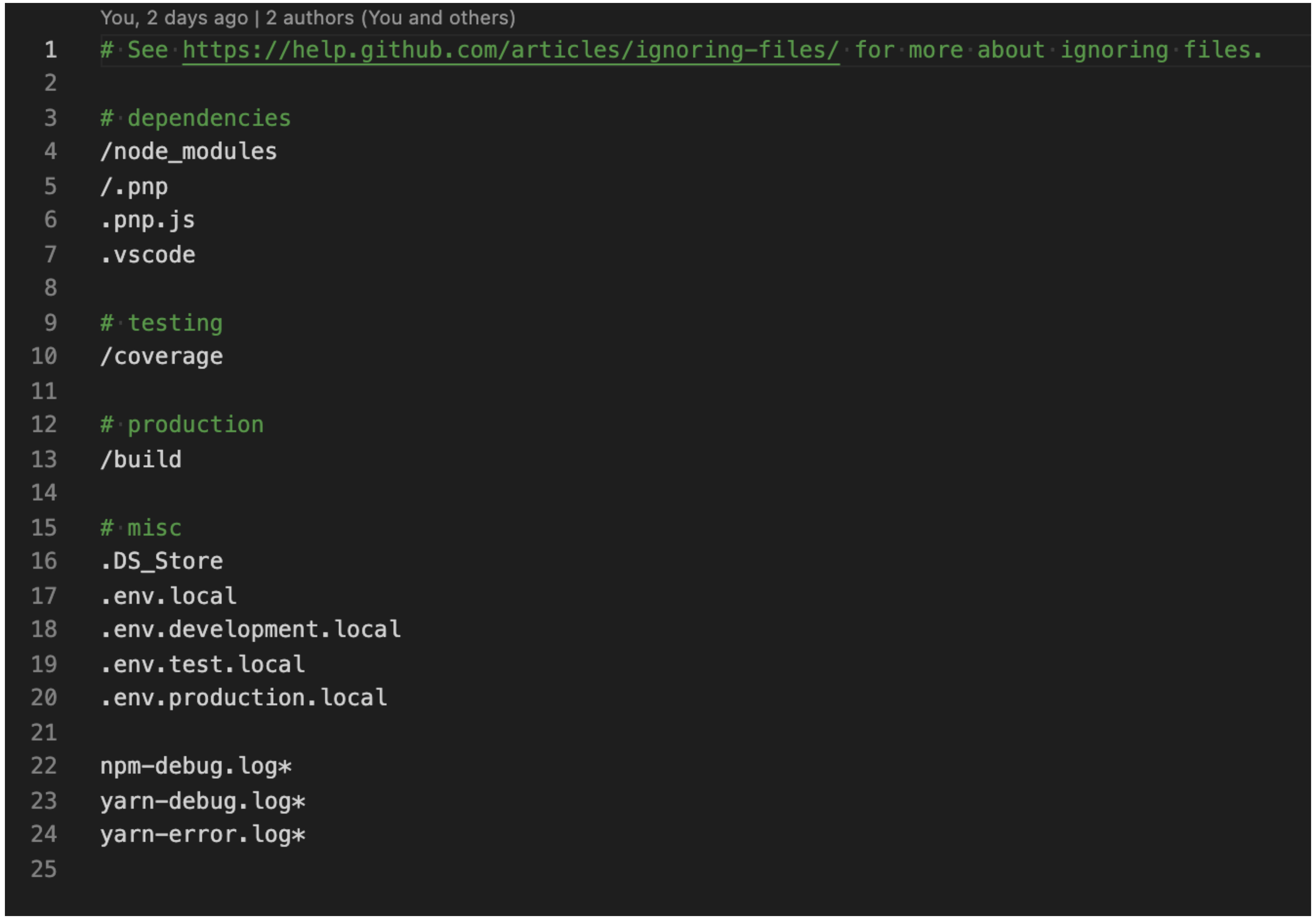
Forking a repository

You might fork a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line. You can practice setting the upstream repository using the same octocat/Spoon-Knife repository you just forked.

1. On GitHub.com, navigate to the octocat/Spoon-Knife repository.
2. In the top-right corner of the page, click **Fork**.

If you want to contribute to existing projects where we do not have permission to make changes by pushing changes, you can create your own "fork" of the project. This means that GitHub will create your own copy of the project, this copy will be in your namespace and you can easily make changes by pushing the changes.

.gitignore



You, 2 days ago | 2 authors (You and others)

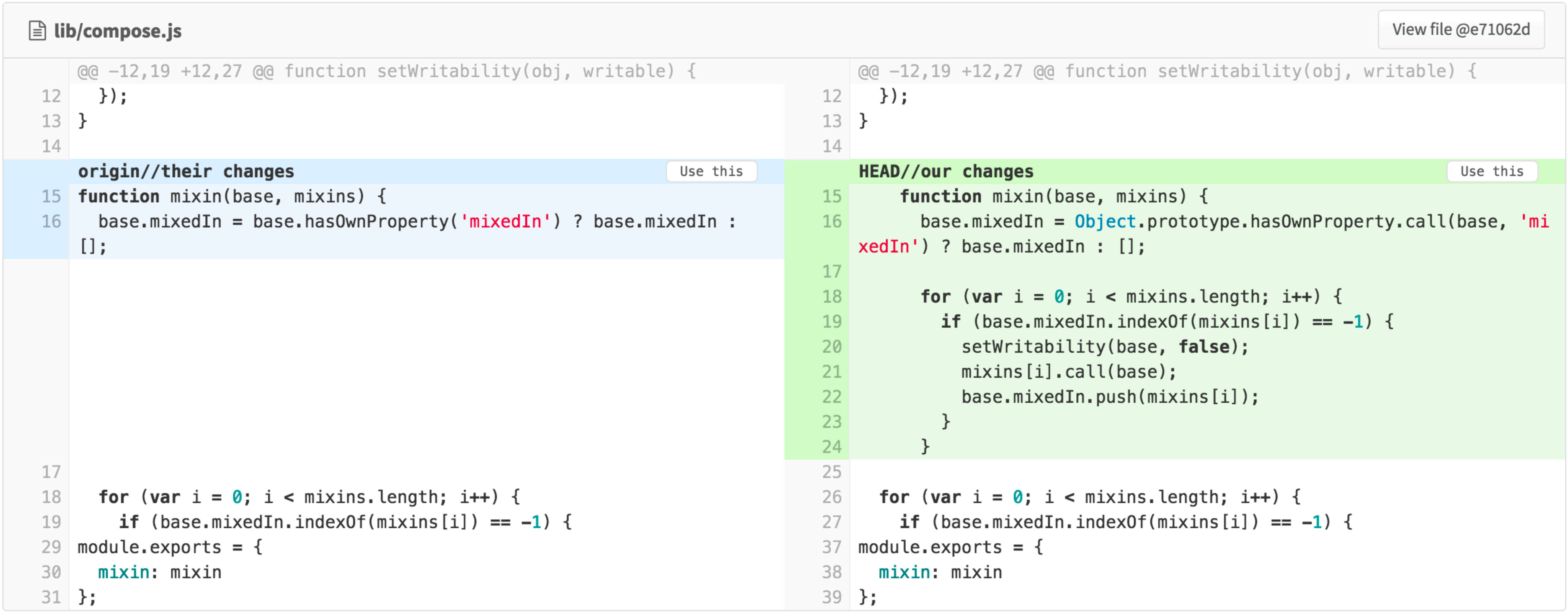
```
# See https://help.github.com/articles/ignoring-files/ for more about ignoring files.

# dependencies
/node_modules
/.pnp
.pnp.js
.vscode

# testing
/coverage

# production
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*

```

In almost any coding project, you'll have some files that you ***don't*** want to be included in version control: build files, cache files, etc.
git diff (show difference what was changed)

Resolve conflicts

To see the beginning of the merge conflict in your file, search the file for the conflict marker `<<<<<<<`. When you open the file in your text editor, you'll see the changes from the HEAD or base branch after the line `<<<<<<< HEAD`. Next, you'll see `=======`, which divides your changes from the changes in the other branch, followed by `>>>>>>> BRANCH-NAME`. In this example, one person wrote "open an issue" in the base or HEAD branch and another person wrote "ask your question in IRC" in the compare branch or `branch-a`.

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```



```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

origin//their changes
function mixin(base, mixins) {
  base.mixedIn = base.hasOwnProperty('mixedIn') ? base.mixedIn : [];

  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

HEAD//our changes
    function mixin(base, mixins) {
      base.mixedIn = Object.prototype.hasOwnProperty.call(base, 'mixedIn') ? base.mixedIn : [];

      for (var i = 0; i < mixins.length; i++) {
        if (base.mixedIn.indexOf(mixins[i]) == -1) {
          setWritability(base, false);
          mixins[i].call(base);
          base.mixedIn.push(mixins[i]);
        }
      }

  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

git rebase vs git merge

Git Rebase

- Streamlines a potentially complex history.
- Avoids merge commit “noise” in busy repos with busy branches.
- Cleans intermediate commits by making them a single commit, which can be
- helpful for DevOps teams.

Git Merge

- Simple and familiar.
- Preserves complete history and chronological order.
- Maintains the context of the branch.