



# Top 10 GitHub best practices for developers



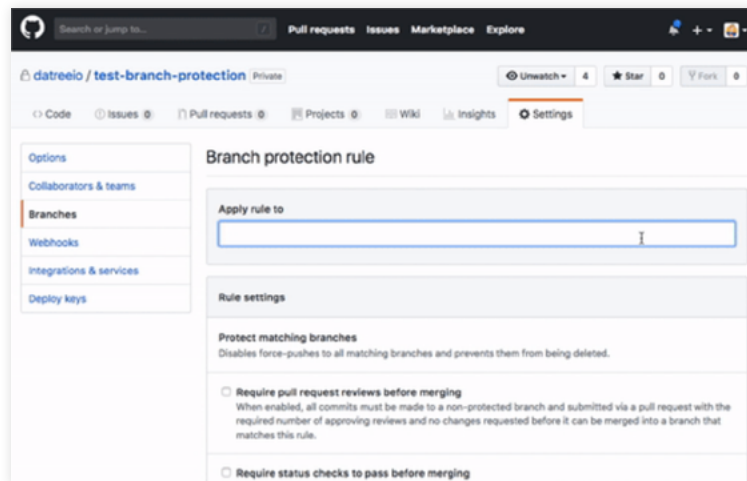
## How we created this GitHub best practices list

We interviewed hundreds of software developers, and performed code scanning on thousands of GitHub repositories using our own product to produce this list.

These best practices are still applicable even if you use something other than GitHub for version control or source control, because they're all about ensuring source code security and writing good code.

## # 9 – Don't just git commit directly to master

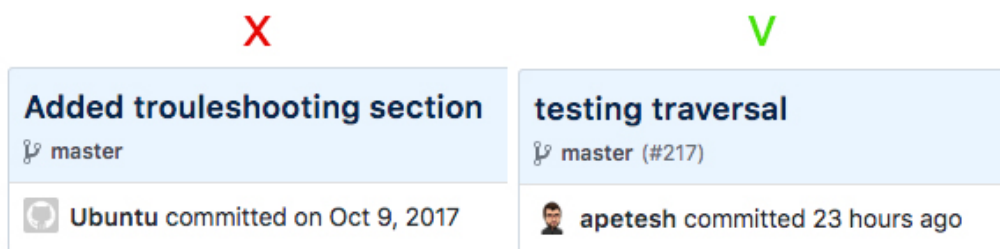
Regardless if you use Gitflow or any other git branching model, it is always a good idea to turn on git branch protection to prevent direct commits and ensure your main branch code is deployable at all times. All commits should be managed via pull requests.



## # 8 – Do git commit with the right email address

Sometimes you commit code using the wrong email address, and as a result GitHub shows that your commit has an unrecognized author. Having commits with unrecognized authors makes it more difficult to track who wrote which part of the code.

Ensure your Git client is configured with the correct email address and linked to your GitHub user. Check your pull requests during code reviews for unrecognized commits.

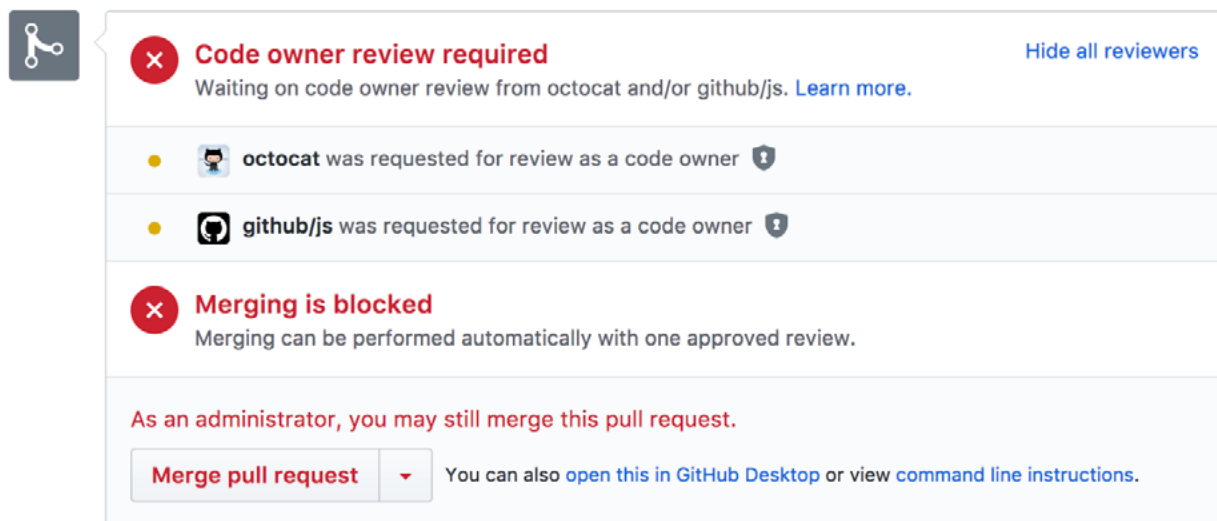


## # 7 – Define code owners for faster code reviews

When you're dealing with dozens, hundreds, or more repositories and engineers, it's nearly impossible to know who owns which parts of the codebase and need to review your changes.

Even in smaller teams you'd still have code owners – for example, front-end code changes should be reviewed by the Front-End Engineer.

Use **Code Owners** feature to define which teams and people are automatically selected as reviewers for the repository.



The screenshot shows a GitHub pull request interface with a red 'X' icon and the text 'Code owner review required'. Below this, it says 'Waiting on code owner review from octocat and/or github/js. [Learn more.](#)' and a 'Hide all reviewers' link. Two entries follow: 'octocat was requested for review as a code owner' and 'github/js was requested for review as a code owner', each with a yellow dot and a shield icon. Below these is another red 'X' icon and the text 'Merging is blocked', followed by 'Merging can be performed automatically with one approved review.' At the bottom, it says 'As an administrator, you may still merge this pull request.' and provides a 'Merge pull request' button with a dropdown arrow. To the right of the button, it says 'You can also [open this in GitHub Desktop](#) or view [command line instructions](#).'

## # 6 – Don't let secrets leak into source code

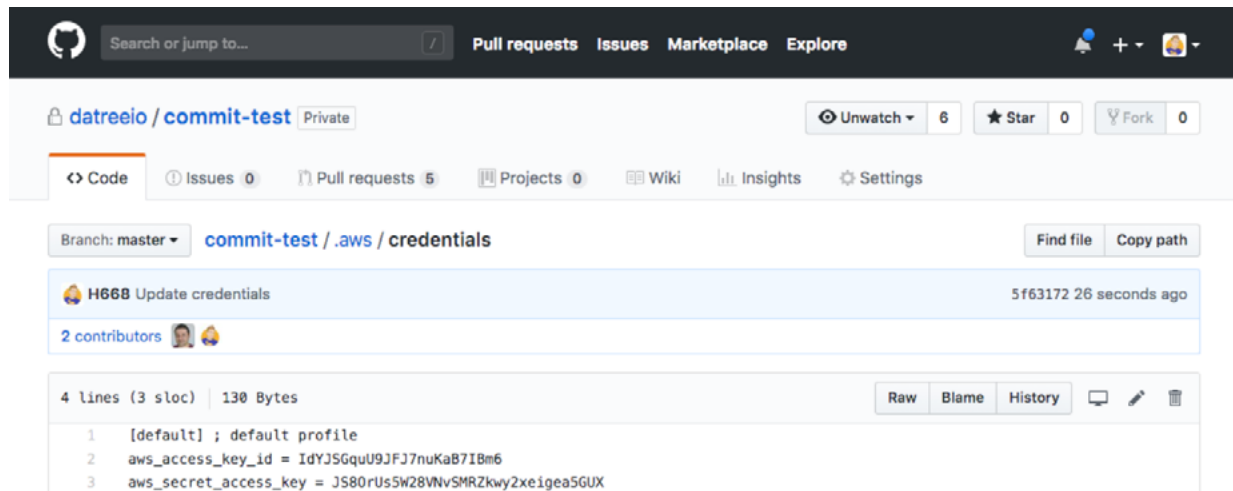
Secrets, or secret keys or secret credentials, include things like account passwords, API keys, private tokens, and SSH keys. You should not check them into your source code.

Instead, we recommend you inject secrets as environment variables externally from a secure store. You can use tools like Hashicorp [Vault](#) or [AWS Secrets Manager](#) to do this.

There are also tools for scanning secrets in repos and prevent them from getting into repos.

- [Git-secrets](#) can help you to identify passwords in your code.
- [Git hooks](#) can be used to build a pre-commit hook and check every pull request for secrets.
- Datree has a built-in [policy rule](#) for this.

Read this [tutorial](#) or watch this [video](#) for a more detailed explanation on why you should manage secrets this way and how to do it right.



The screenshot shows a GitHub repository named 'datreeio / commit-test' with a private setting. The file path is '.aws / credentials'. A commit titled 'H668 Update credentials' is shown, made 26 seconds ago. The commit contains 4 lines of code (3 sloc) and 130 bytes. The code is as follows:

```
1 [default] ; default profile
2 aws_access_key_id = IdYJSGquU9JFJ7nuKaB7IBm6
3 aws_secret_access_key = JS80rUs5W28VNvSMRZkwy2xe1gea5GUX
```

## # 5 – Don't commit dependencies into source code

Pushing dependencies into your remote origin will increase repository size. Remove any projects dependencies included in your repositories and let your package manager download them in each build. if you are afraid of “dependencies availability” you should consider using a binary repository manager solution like [Jfrog](#) or [Nexus Repository](#). Or check out GitHub's [Git-Sizer](#).

## # 4 – Don't commit configuration files into source code

We strongly recommend against committing your local config files to version control. Usually, those are private configuration files you don't want to push to remote because they are holding secrets, personal preferences, history or general information that should stay only in your local environment.

## # 3 – Create a meaningful git ignore file

A .gitignore file is a must in each repository to ignore predefined files and directories. It will help you to prevent secret keys, dependencies and many other possible discrepancies in your code. You can choose a relevant template from [Gitignore.io](#) to get started quickly.



```
# Created by https://www.gitignore.io/api/node

### Node ###
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*

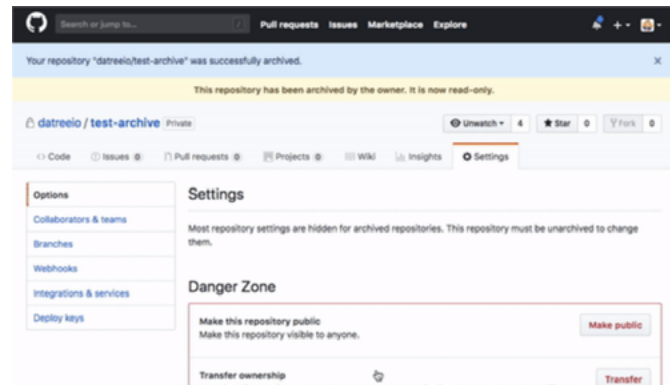
# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSCover
lib-cov
```

## # 2 – Archive dead repositories

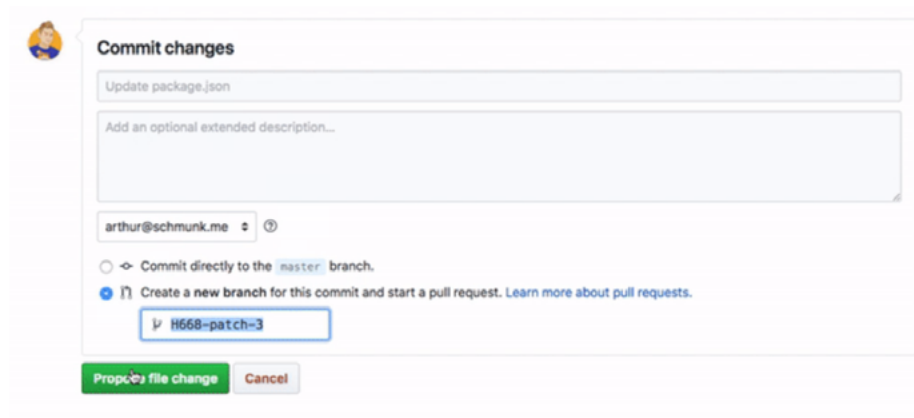
Over time, for various reasons, we find ourselves with unmaintained repositories. Sometimes developers create repos for an ad hoc use case, a POC, or some other reason. Sometimes they inherit repos with old and irrelevant code.

In any case, these repos were left intact. No one is doing any development work in those repos anymore, so you want to clean them up and avoid the risk of other people using them. The best practice is to archive them, i.e. make them “read-only” to everyone.



## # 1 – Lock package version

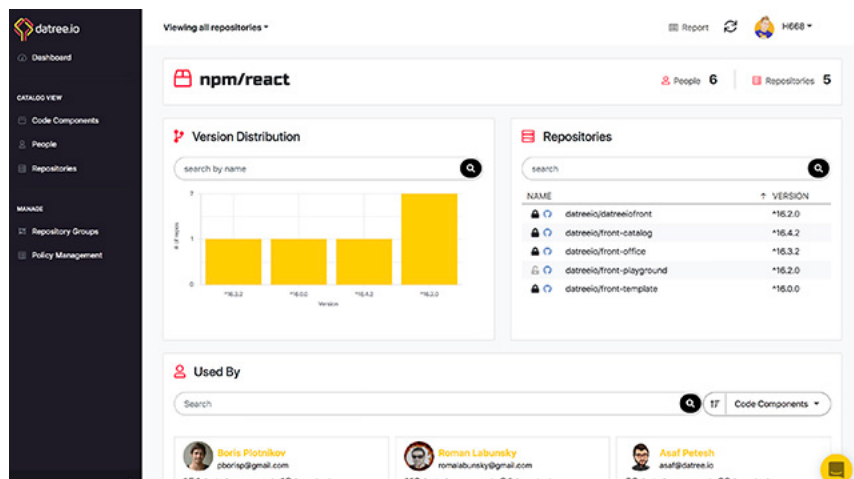
Your manifest file contains information about all packages and dependencies in your project and their versions. The best practice is to specify a version or version range for every package and dependency listed in the manifest. Otherwise, you can't be sure which version will get installed during the next build, and consequently your code may break.



## # 0 – Align packages versioning

Even when everyone on your team are using the same packages, reusing code and tests across different projects can still be difficult if the packages are of different versions.

If you have a package that is used in multiple projects, try at a minimum to use the same major version of the package.



## What's next?

All that's left for you to do is check off each of the aforementioned best practices, on each of your repositories, one by one...

Or, save your sanity and connect with Datree's GitHub app to scan your repositories and generate your free status report to assess if your repositories align with the listed best practices.

[\*\*< Get My Free Status Report />\*\*](#)



## About Datree

Datree is a Git-based policy enforcement solution. Datree ensures every code committed to source control follows your security policies and coding standards. It does that by performing automatic policy compliance checks on every pull request.

With Datree, you can centrally set and manage policies across all your Git repositories at once, including new future one, and easily track & report on policy compliance adherence.

[Schedule a demo](#) for more information.