datree.io

An Expanded Guide to GitHub Best Practices

Do you use GitHub? Implementing these best practices could save you time, improve developer productivity, and reduce security risks.

Last updated: Dec 2019

Written by:

Eyar Zilberman
Elliott Bonneville

Introduction

How we created this guide

We interviewed hundreds of software developers to understand their development workflows and how they work with GitHub. Using our own product, we also scanned thousands of GitHub repositories for our customers.

This list of GitHub best practices is derived from the insights we gleamed from those experiences.

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

Who this guide is for

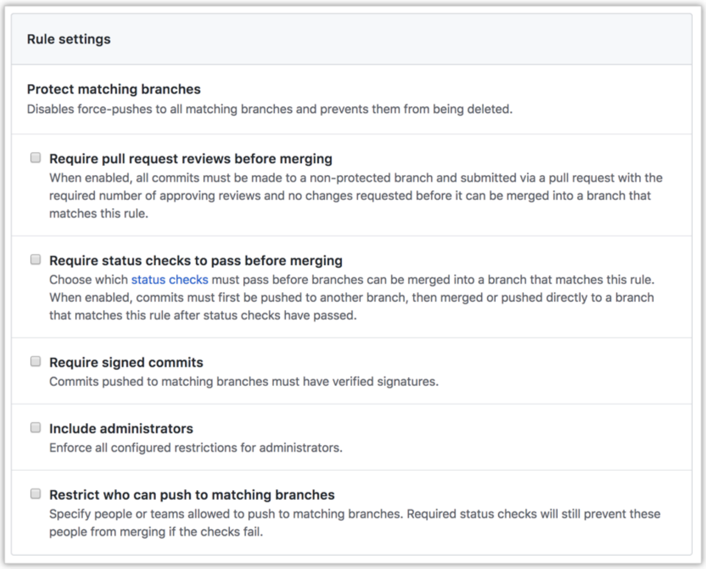
This guide is for anyone in the Engineering organization looking to improve developer workflow and productivity, as well as code quality and security.

Those responsible for putting together team-wide standard practices and policies would benefit greatly from this guide, but even if you're a developer not "in charge" of driving such standards, you will find these practices applicable and useful to remember.

GitHub Best Practices

0 Don’t git push straight to master

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



Rule settings

Protect matching branches
Disables force-pushes to all matching branches and prevents them from being deleted.

☐ **Require pull request reviews before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

☐ **Require status checks to pass before merging**
Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require signed commits**
Commits pushed to matching branches must have verified signatures.

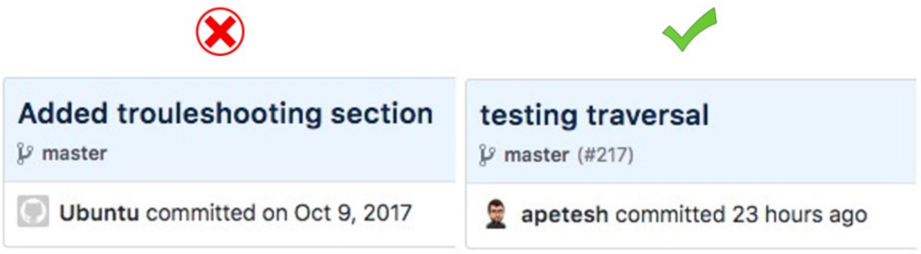
☐ **Include administrators**
Enforce all configured restrictions for administrators.

☐ **Restrict who can push to matching branches**
Specify people or teams allowed to push to matching branches. Required status checks will still prevent these people from merging if the checks fail.

1 Don’t commit code as an unrecognized author

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.



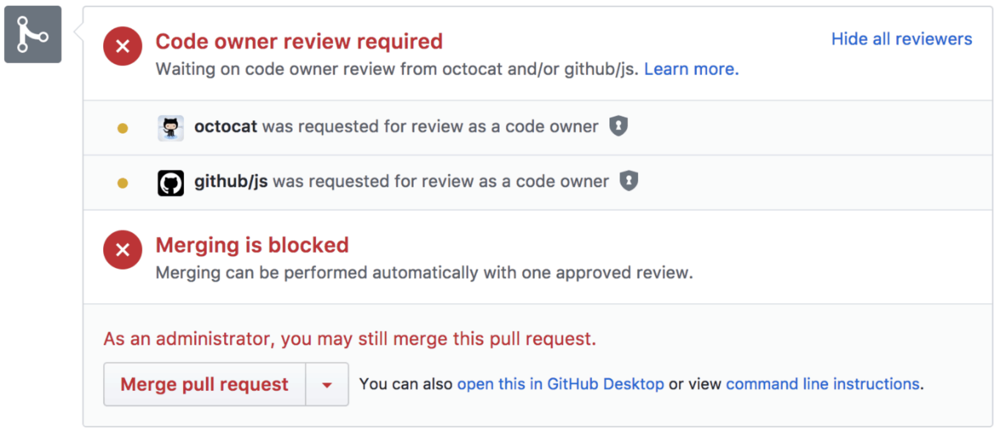
Added trouleshooting section
master
Ubuntu committed on Oct 9, 2017

testing traversal
master (#217)
apetesh committed 23 hours ago

2 Define code owners in your codebase

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



Code owner review required
Waiting on code owner review from octocat and/or github/js. Learn more.
Hide all reviewers

octocat was requested for review as a code owner

github/js was requested for review as a code owner

Merging is blocked
Merging can be performed automatically with one approved review.

As an administrator, you may still merge this pull request.

Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

3 Don’t let secrets leak into source control

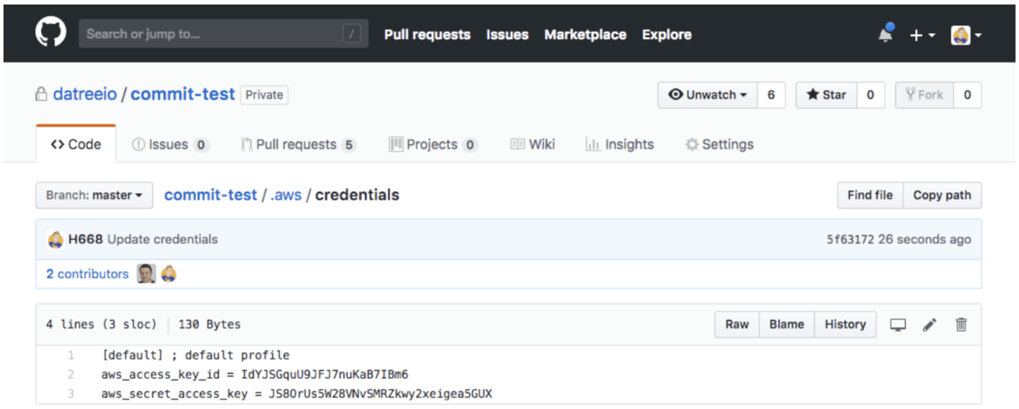
Secrets, or secret keys or secret credentials, include things like account passwords, API keys, private tokens, and SSH keys. You should not check them into your source code.

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 many 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 predefined 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.



commit-test / .aws / credentials

H668 Update credentials 5f63172 26 seconds ago

2 contributors

4 lines (3 sloc) | 130 Bytes

```
[default] ; default profile
aws_access_key_id = IdYJSGquU9JFJ7nuKaB7IBm6
aws_secret_access_key = JS8OrUs5W28VNvSMRZkwy2xeigea5GUX
```

4 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.

5 Don’t commit config 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.

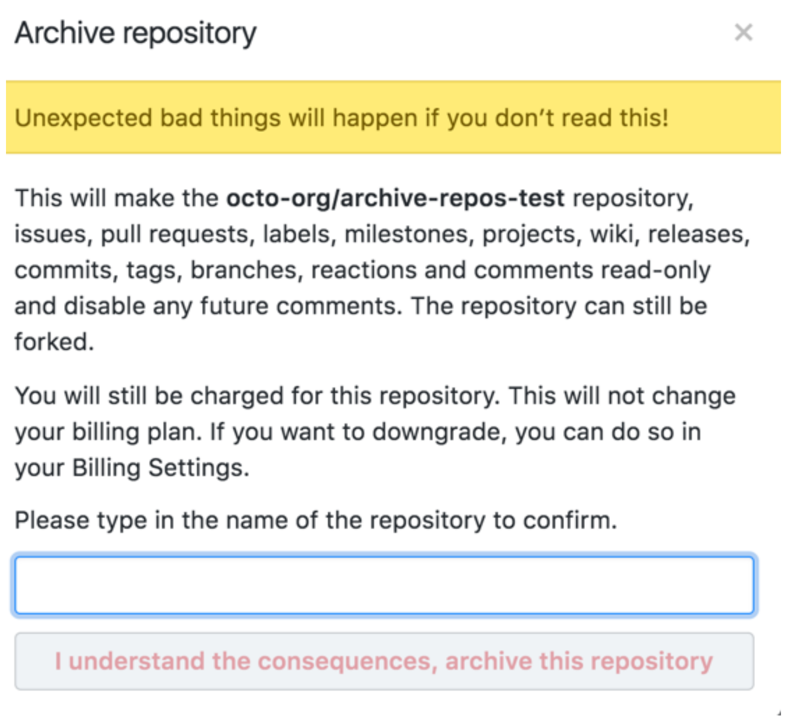
6 Create a meaningful git ignore file

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.

7 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.


Archive repository

Unexpected bad things will happen if you don’t read this!

This will make the **octo-org/archive-repos-test** repository, issues, pull requests, labels, milestones, projects, wiki, releases, commits, tags, branches, reactions and comments read-only and disable any future comments. The repository can still be forked.

You will still be charged for this repository. This will not change your billing plan. If you want to downgrade, you can do so in your Billing Settings.

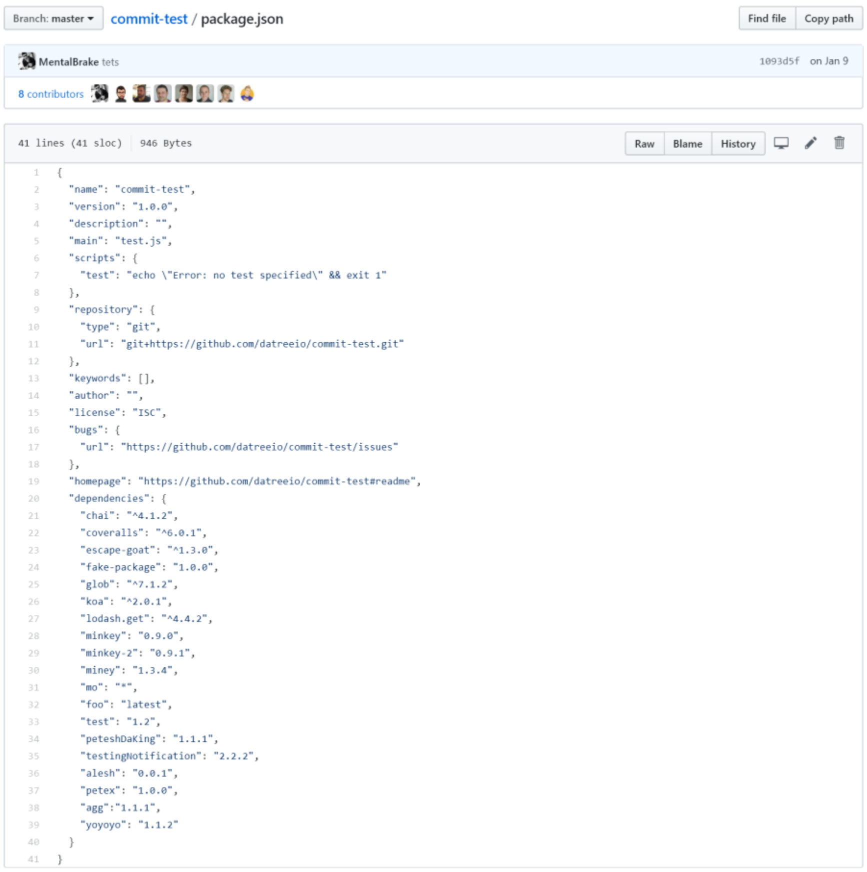
Please type in the name of the repository to confirm.

8 Lock package versions

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.



```
{
  "name": "commit-test",
  "version": "1.0.0",
  "description": "",
  "main": "test.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/datreeio/commit-test.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/datreeio/commit-test/issues"
  },
  "homepage": "https://github.com/datreeio/commit-test#readme",
  "dependencies": {
    "chai": "^4.1.2",
    "coveralls": "^6.0.1",
    "escape-goat": "^1.3.0",
    "fake-package": "1.0.0",
    "glob": "^7.1.2",
    "koa": "^2.0.1",
    "lodash.get": "^4.4.2",
    "minkey": "0.9.0",
    "minkey-2": "0.9.1",
    "miney": "1.3.4",
    "mo": "*",
    "foo": "latest",
    "test": "1.2",
    "peteshDaKing": "1.1.1",
    "testingNotification": "2.2.2",
    "alesh": "0.0.1",
    "petex": "1.0.0",
    "agg":"1.1.1",
    "yoyoyo": "1.1.2"
  }
}
```

9 Specify standard package versions

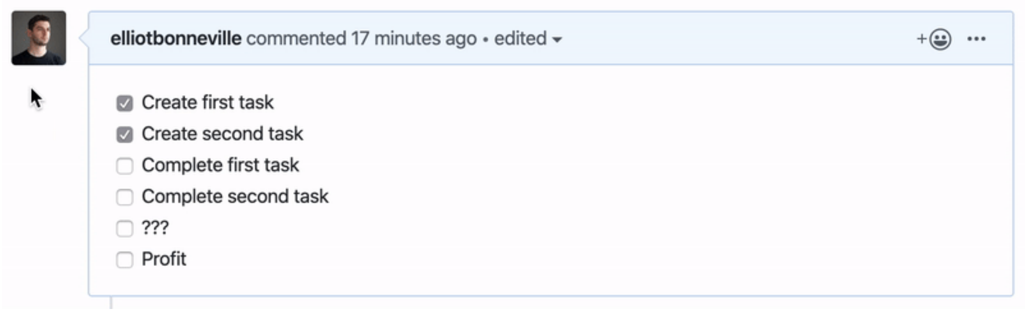
Even when everyone on your team are using the same packages, reusing code and tests across different projects can still be difficult if the packages are of different versions.

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

10 Leverage tasks list

Tasks lists provide a way for you to track to-dos directly within comments, issues, and even MarkDown files (*.MD) within your repository (users must have write access to the repository to make changes to MarkDown files).

Tasks lists provide an excellent way to capture a high-level overview of a task or issue, as well as keep others updated on its state. Make sure to take advantage of this powerful new feature!



elliotbonneville commented 17 minutes ago • edited

- [x] Create first task
- [x] Create second task
- [ ] Complete first task
- [ ] Complete second task
- [ ] ???
- [ ] Profit

11 Use branch naming convention

Adopting a consistent branch naming convention is essential to keeping your repository organized as your team grows in size. An efficient naming convention will allow you to keep merge conflicts at a minimum while ensuring your developers are as productive as possible.

While there are many branch naming conventions, one of the most popular ones is known as git flow:

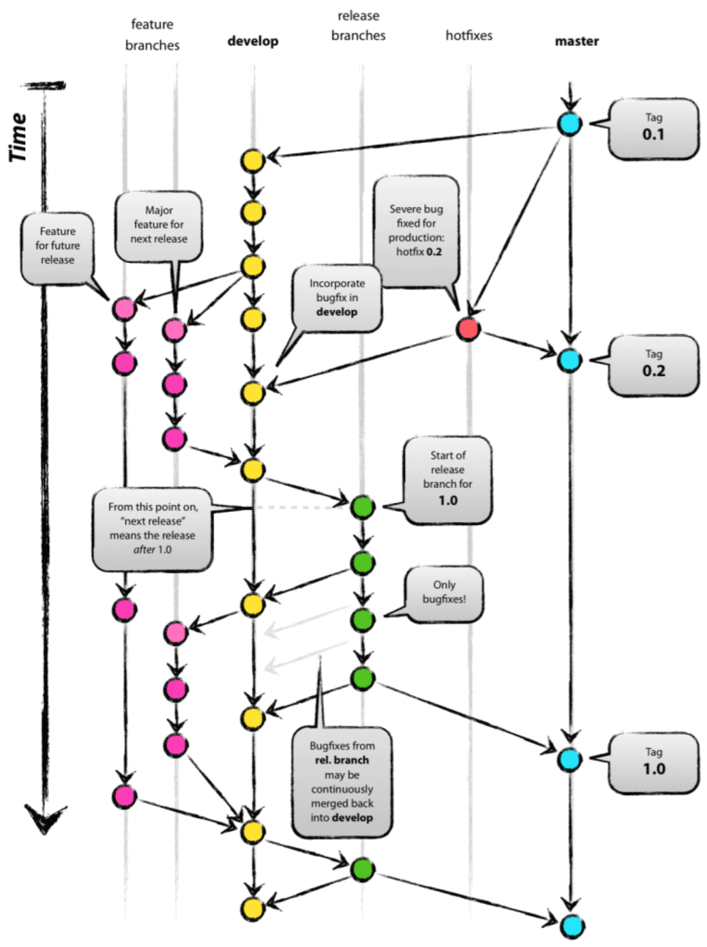

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Image credit: Vincent Driessen, nvie.com

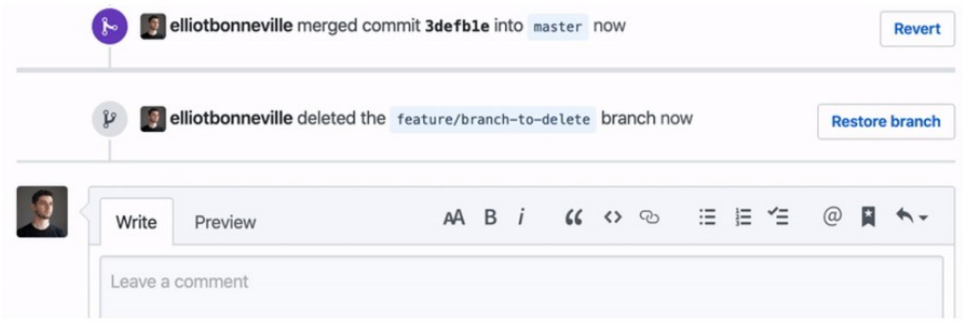
12 Delete stale branches

Every time one branch is merged into another, the branch that is merged in becomes stale, assuming further work isn't being done in it.

While it may seem useful or even necessary to keep the extra data on hand, the reality is that stale branches are abandoned 98% of the time and simply clutter up your project.

Even if you delete a branch when you shouldn't have, you can restore it - and if you don't trust GitHub's restore feature, chances are it's safe on somebody's computer, thanks to the magic of distributed versioning.

Don't be a branch hoarder: delete your stale branches.


elliotbonneville merged commit `3defb1e` into `master` now

elliotbonneville deleted the `feature/branch-to-delete` branch now

13 Keep branches up to date

Let's say you've finally completed some work on a long-outstanding branch and you're ready to merge it into master. You pull from remote, hit merge, and suddenly you're faced with a barrage of merge conflicts.

What happened?

You failed to keep your branch up-to-date with the branch you're attempting to merge into. Lots of commits went by and some conflicted with your changes... now you're faced with spending time and energy resolving an unnecessary amount merge conflicts.

The best practice here is to ensure that you're consistently merging your base branch into your current branch as you work, especially if it's a long-outstanding branch.

14 Remove inactive GitHub members

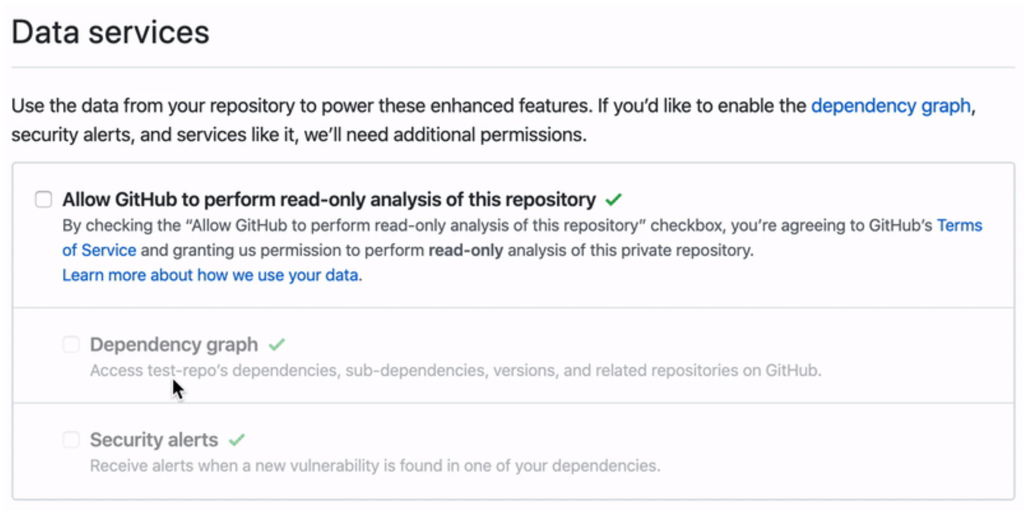
While it might seem obvious, it's worth mentioning in a comprehensive list of best practices... Be sure to remove contributors from your organization that are no longer contributing to your codebase.

If you remove somebody from your organization for any reason, revoke their GitHub access immediately as well. Even in completely amicable situations, it's better safe than sorry!

15 Enable security alerts

Security alerts are another feature new to GitHub. You can read about them here, but the gist is that GitHub now tracks reported security vulnerabilities in some dependencies and will even suggest fixes for you.

This is turned on automatically for all public repositories, but if your repository is private, you'll need to opt in manually.



Data services

Use the data from your repository to power these enhanced features. If you'd like to enable the dependency graph, security alerts, and services like it, we'll need additional permissions.

- [ ] **Allow GitHub to perform read-only analysis of this repository** ✓
  By checking the "Allow GitHub to perform read-only analysis of this repository" checkbox, you're agreeing to GitHub's Terms of Service and granting us permission to perform **read-only** analysis of this private repository.
  Learn more about how we use your data.
  - [ ] Dependency graph ✓
    Access test-repo's dependencies, sub-dependencies, versions, and related repositories on GitHub.
  - [ ] Security alerts ✓
    Receive alerts when a new vulnerability is found in one of your dependencies.

Conclusion

Developers spend a lot of time working with git and GitHub, so investing in improving your GitHub practices makes a lot of sense. Implementing best practices in this guide could help the team improve developer productivity and reduce security risks.

Final Thoughts

Ensuring consistent adoption of best practices could be very challenging, especially in fast-growing or large teams.

The ways people try to solve this problem, like writing down policies in a shared document or wiki, sending mass emails from time to time to entire teams, Slacking them in the team channel, or hoping code reviews will catch everything, are not consistently effective – let alone scalable.

Whether you decide to build in-house or use a purpose-built commercial tool, investing in automating best practice adoption is a good idea and will help you prevent costly mistakes.

datree.io

About Datree

Datree helps teams automatically adopt development best practices, coding standards, and security policies.

It does that by performing automated GitHub checks that run like your CI tests.

Every time new code is committed, Datree checks if the rules you've set are followed - and tells the developer when they aren't.

Sign up for free