ANGULAR ROUTING ALL RXJS STATE FORMS NGRX STENCIL JEST UNIT E2E TDD JWT AUTH SECURITY ANIMATION GESTURES REACT REDUX CAPACITOR CORDOVA PLUGINS NESTJS SWIFT XSS GIT AGILE CD NETLIFY DECLARATIVE FIRESTORE

search site by tag or title...

Requiring Pull Requests and Reviews in Your Git Workflow

Josh Morony

August 31, 2021 7 min read

ARCHITECTURE ALL GIT AGILE

Originally published **August 31, 2021**

This is part of a series, check out the other parts below:

Earlier in this series we covered how to use **branch-per-issue** or **task branching** to **isolate work into short lived feature branches** in Git that are then merged back into the `main` branch. I would recommend reading that tutorial for more information on the philosophy of this process, but the basic idea is this:

- An issue is created for each bit of work to be performed (these are generally small units of work)
- A new branch is created for the issue
- Work is performed/completed in that branch
- That branch is then merged back into the `main` branch once work is complete (or perhaps an `integration` branch if you prefer)

One thing that has been lacking from anything we have covered so far is **governance features**. We have been operating under the assumption that anybody who is working on our repository will have full access to everything and will act according to the rules/processes we have set forth. In practice, not everyone is always going to follow these rules (especially in larger teams).

With the set up we have now, there is nothing stopping ourselves or one of our developers from just pulling down the `main` branch and pushing straight back to it. As silly as it might sound, you might even want to set up governance features if it is just **you** and **only you** working on the repository. This won't actually add any extra protection in this case because you will be able to bypass restrictions with administrator privileges, but it will help you develop good Git/Github workflow hygiene. Setting up and following governance features for yourself can help improve and enforce your own software development process, and it will also help a great deal if you ever do need to transition to a more team oriented environment.

Become a PRO member

Source code

email...

Join the newsletter. I'll give you exact details in the double opt-in email confirmation.

We are now going to start exploring how to add governance features to more strictly control what can and can not be done in our repositories, starting with: **pull requests**.

Outline

Source code

What is a pull request?

If you've been using Git for any substantial amount of time then you have probably at least heard of **pull requests**. However, pull requests are not necessarily an integral part of everybody's Git workflow. You could use Git/GitHub for years and never once have to create/merge a pull request.

A **pull request** isn't really anything special. If I have been working on a branch in the repository, and I push that back up to the remote branch on GitHub (or wherever), a pull request is basically a formal method to say:

> *"Hey, I've got some changes over in this branch I want to merge into the main branch"*

Creating a pull request will allow you to review the changes that will be included and make comments about the pull request for the person(s) who will be reviewing that pull request.

This then creates a more formal environment where the changes can be reviewed/discussed, and if necessary you (or others) can push further changes to that branch which will also be included in the pull request.

A pull request can then be merged into the main branch either directly through Github (just by clicking a button), or by checking out the pull request locally and merging it using the command line (this will be required if there are merge conflicts that need to be resolved).

NOTE: We are talking about a **shared repository model** in this tutorial where multiple contributors all have push access to the same shared repository. Pull requests are also heavily used in open source software where you might typically **fork** a repository you don't have push access to, make some changes in your own fork, and then create a **pull request** to request changes from your fork be included in the original repository.

Why do we want to use pull requests?

As I mentioned at the beginning, there is nothing stopping any contributor in our repository from just pulling down the `main` branch and pushing right back to it. This violates the **task branching** model we are trying to follow, and we just have to hope that people follow the rules. Even if people do make a good faith effort to follow the rules in general, there will almost certainly be times where these rules are violated.

Although this will be our main motivation for enforcing that pull requests are used, using pull requests and reviews is also just a good idea from a collaboration and code review perspective.

Enforcing pull requests

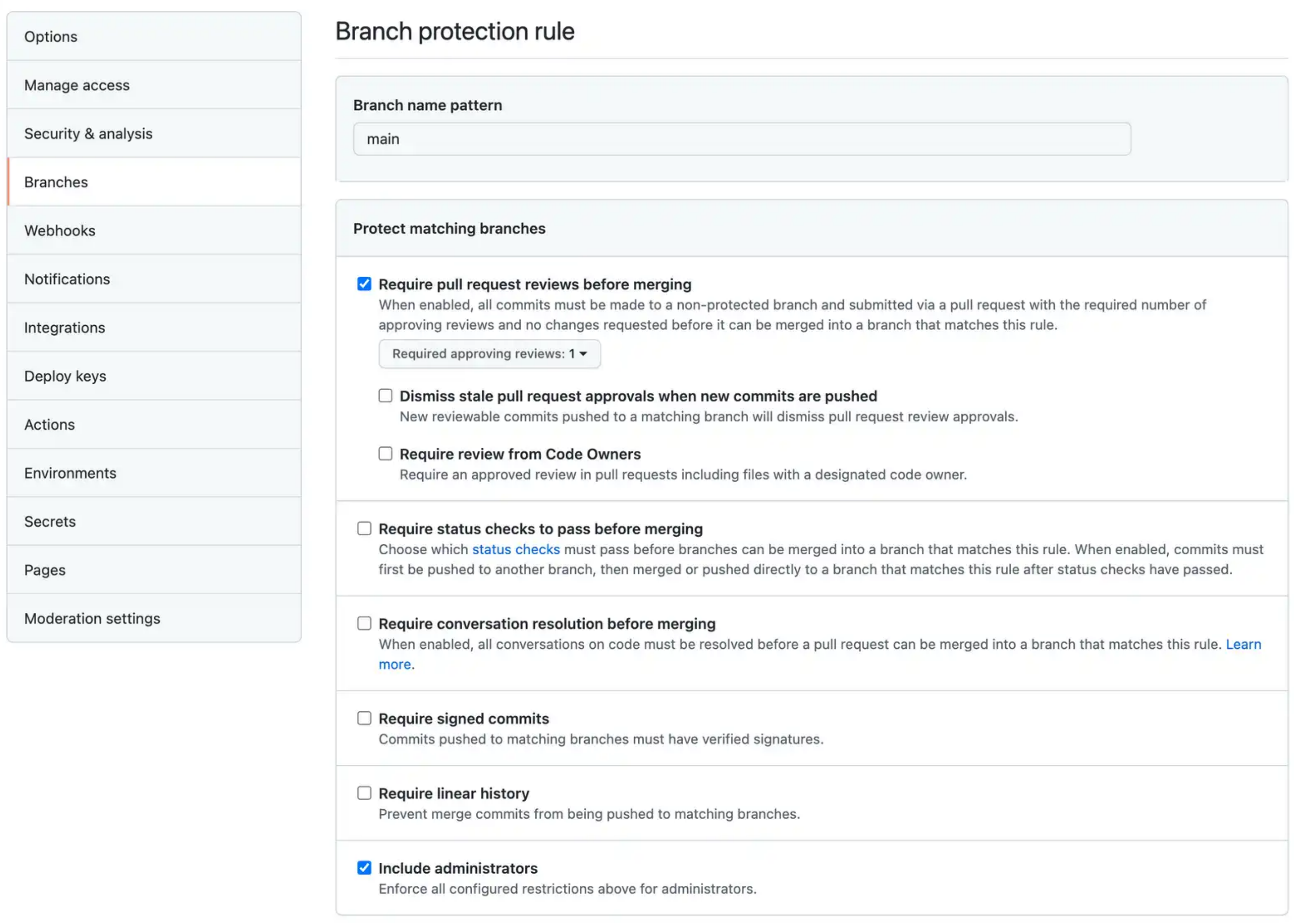
Making a pull request required for merging into the `main` branch (or any branch) is quite easy with Github. Let's do it now.

1. Go to `Settings > Branches`
2. Click `Add rule` on `Branch protection rules`
3. Add `main` as the `Branch name pattern`

Now you can set whatever rules you want. We are going to add the following rules:

```
Require pull request reviews before merging
- Required approving reviews: 1
```

```
Include Administrators
```



Options
Manage access
Security & analysis
Branches
Webhooks
Notifications
Integrations
Deploy keys
Actions
Environments
Secrets
Pages
Moderation settings

Branch protection rule

Branch name pattern
main

Protect matching branches

[✓] 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.
Required approving reviews: 1

[ ] Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

[ ] Require review from Code Owners
Require an approved review in pull requests including files with a designated code owner.

[ ] 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 conversation resolution before merging
When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. Learn more.

[ ] Require signed commits
Commits pushed to matching branches must have verified signatures.

[ ] Require linear history
Prevent merge commits from being pushed to matching branches.

[✓] Include administrators
Enforce all configured restrictions above for administrators.

Once you have enabled the rules you need, you just need to click `Create`.

NOTE: If you are the only person working on this repository you should **not** check the `Include Administrators` option. You can not review your own pull requests, so you will need the ability to bypass the review requirement. Unfortunately, this will also allow you to push directly to the `main` branch so it defeats the purpose of doing this in the first place. As I mentioned, I still think it is a good idea to set up and follow this process as if it were required, but just know that your admin privileges remove the protections we are putting in place.

An example of the workflow with pull requests

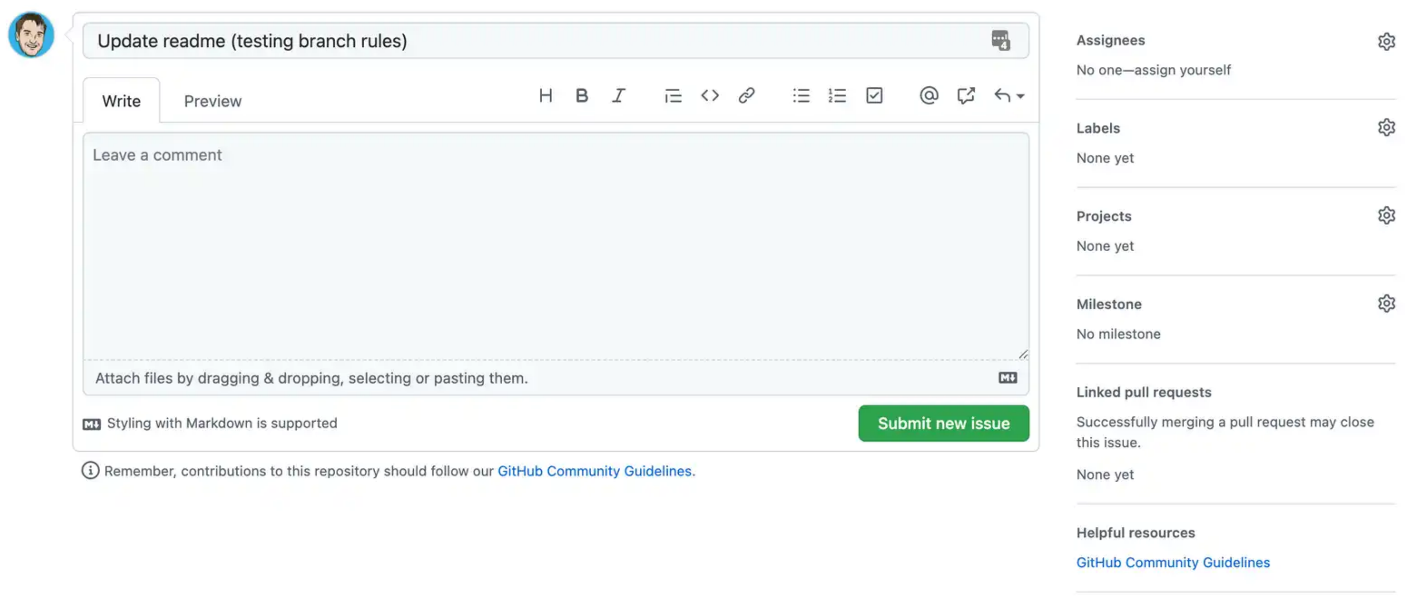
Let's see what our workflow looks like now. One of our main goals here was to stop people from pushing changes directly to the `main` branch. Let's see what happens if we accidentally worked on the `main` branch and tries to push it:

1. `git checkout main`
2. Make a change
3. `git add .`
4. `git commit -m "pushing straight to main"`
5. `git push`

```
remote: error: GH006: Protected branch update failed for
refs/heads/main.
remote: error: At least 1 approving review is required by
reviewers with write access.
To github.com:joshuamorony/pm-tutorial.git
 ! [remote rejected] main -> main (protected branch hook
declined)
error: failed to push some refs to
'github.com:joshuamorony/pm-tutorial.git'
```

NOTE: If you have not set the `Include Administrators` option mentioned above the push to `main` will work.

Perfect, our push was rejected. Realising our error we might now want to move our commits onto a new branch. First, we will create an issue for the branch we are creating:


Update readme (testing branch rules)

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Helpful resources
GitHub Community Guidelines

NOTE: Creating an issue isn't actually enforced.

Then we will create a new branch that references that issue (using the branch name format we discussed in the first tutorial in this series):

```
git checkout -b jm-15
```

BONUS TIP: Moving work committed onto the main branch to a feature branch

Since this branch was just created from our local `main` branch it will include the changes that we committed to the `main` branch. This allows us to carry our work over to this new branch. However, if this branch already existed then this **would not work**. If we already had a branch called `jm-15`, and we wanted to move commits from `main` to our feature branch, then what we could do instead is on the `main` branch run:

```
git log
```

To view the most recent commits. Make a note of the **hash** of each commit that you want to move over to the existing branch, e.g:

```
commit e2586817c9544734899ab41cd92fb2bc99fc3b4c
```

NOTE: Type `:q` to exit the editor

Then you can change back to your existing feature branch:

```
git checkout jm-15
```

and pull those commits into the feature branch using `cherry-pick` for each commit that you want to keep:

```
git cherry-pick e2586817c9544734899ab41cd92fb2bc99fc3b4c
```

To remove the commits from the `main` branch and get that back into a fresh state from the origin, you can just run:

```
git checkout main
```

```
git reset --hard origin/main
```

This will remove any staged commits and destroy your local work. If you switch between `main` and `jm-15` now, you should see that `main` does not contain any changes and `jm-15` does, which is what we should have done initially.

Creating and merging a pull request

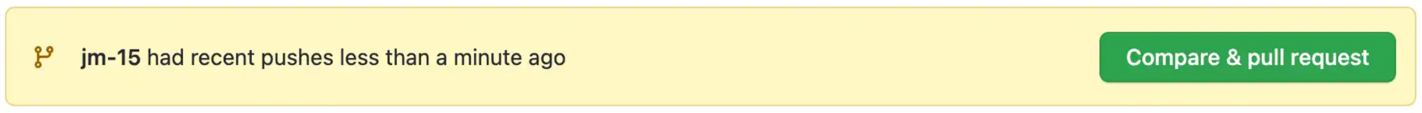
Ok, we have our new work in its own branch now, which we were forced to do by the branch protection rules that we created. Now let's see how we actually go about getting our changes merged into `main`.

First, we will need to push our changes up to the remote repository:

```
git push --set-upstream origin jm-15
```

NOTE: `--set-upstream origin jm-15` is only required if this is the first time you are pushing the branch to the remote repository (i.e. the branch does not exist in the remote repository yet)

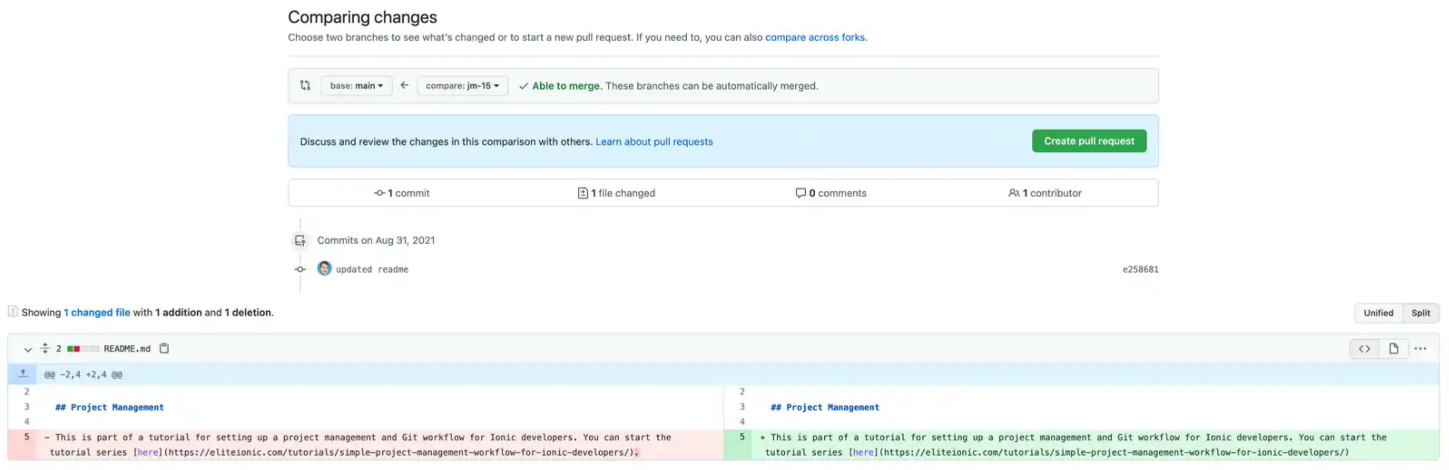
If you go to Github now you will see a notice like this:



jm-15 had recent pushes less than a minute ago

You can just click **Compare & pull request** here but that notice won't be displayed forever. You can also create a pull request by going to the **Pull requests** tab and then clicking **New pull request**.

You will want to set the **base** to `main` (or whatever your main branch is called) and the **compare** to `jm-15` (or whatever the branch you are merging is called). You will then be able to review the changes:



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: main ← compare: jm-15 ✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

1 commit · 1 file changed · 0 comments · 1 contributor

Commits on Aug 31, 2021

updated readme e258681

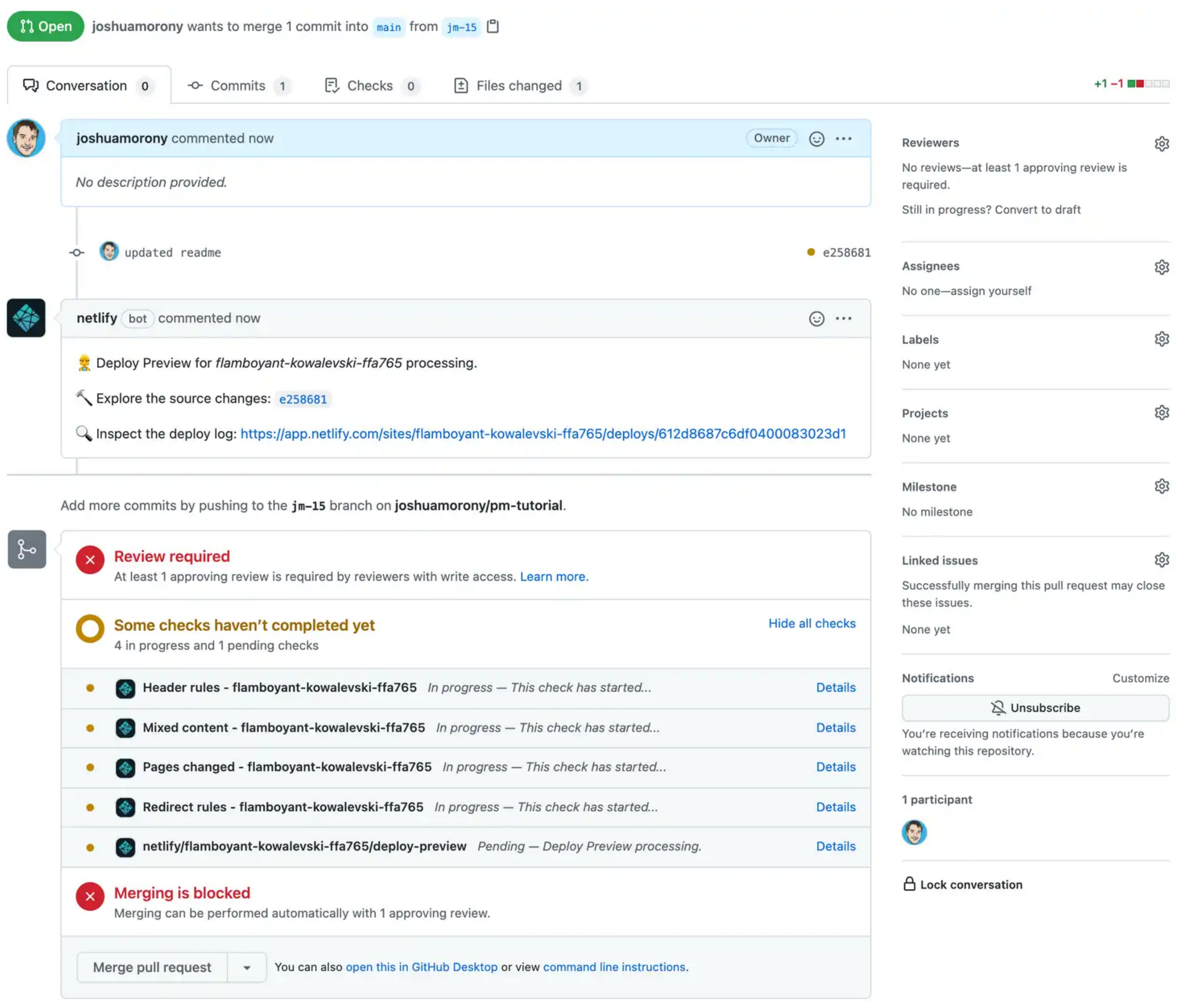
Showing 1 changed file with 1 addition and 1 deletion.

```
2 README.md
@@ -2,4 +2,4 @@

## Project Management

- This is part of a tutorial for setting up a project management and Git workflow for Ionic developers. You can start the tutorial series [here](https://eliteionic.com/tutorials/simple-project-management-workflow-for-ionic-developers/).
+ This is part of a tutorial for setting up a project management and Git workflow for Ionic developers. You can start the tutorial series [here](https://eliteionic.com/tutorials/simple-project-management-workflow-for-ionic-developers/)
```

Once you are satisfied, click **Create pull request**. You will then be given the chance to add a comment, and then you can click **Create pull request** again to actually create the pull request. Then you will see a rather intimidating screen like this:


Open joshuamorony wants to merge 1 commit into main from jm-15

Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +1 −1

joshuamorony commented now · Owner

No description provided.

updated readme · e258681

netlify bot commented now

Deploy Preview for *flamboyant-kowalevski-ffa765* processing.

Explore the source changes: e258681

Inspect the deploy log: https://app.netlify.com/sites/flamboyant-kowalevski-ffa765/deploys/612d8687c6df0400083023d1

Add more commits by pushing to the **jm-15** branch on **joshuamorony/pm-tutorial**.

Review required
At least 1 approving review is required by reviewers with write access. Learn more.

Some checks haven't completed yet
4 in progress and 1 pending checks · Hide all checks

- Header rules - flamboyant-kowalevski-ffa765 *In progress — This check has started...* Details
- Mixed content - flamboyant-kowalevski-ffa765 *In progress — This check has started...* Details
- Pages changed - flamboyant-kowalevski-ffa765 *In progress — This check has started...* Details
- Redirect rules - flamboyant-kowalevski-ffa765 *In progress — This check has started...* Details
- netlify/flamboyant-kowalevski-ffa765/deploy-preview *Pending — Deploy Preview processing.* Details

Merging is blocked
Merging can be performed automatically with 1 approving review.

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

Reviewers: No reviews—at least 1 approving review is required. Still in progress? Convert to draft

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

Notifications · Customize · Unsubscribe · You're receiving notifications because you're watching this repository.

1 participant

Lock conversation

The status checks aren't actually of any concern to us here as we haven't enabled status checks in our branch protection rules (we will likely cover that in another tutorial). What is blocking us at the moment though is the need for a review:

```
Review required
At least 1 approving review is required by reviewers with
write access.
```

To review a pull request you will need someone (who is not the person who created the pull request) to:

1. Open the pull request and click on the `Files changed` tab
2. Click `Review changes`
3. Leave a comment, check `Approve`, and then `Submit review`

4. Back on the main pull request page, click `Merge pull request`

NOTE: You can not not approve your own pull requests, so if you are working on the repository by yourself you will need to bypass the review restrictions by just clicking `Merge pull request` without submitting a review. The `Include administrators` option under the branch protection rules must be disabled in order to allow this.

And we are done!

Summary

We all make mistakes, and wherever possible we shouldn't rely on people manually following processes and following the rules. If mistakes from developers end up negatively affecting our codebase or product, then we should see it as a failure of our processes not of that developer. If we can use tools to help enforce the software development process we want to create, and protect/guide our developers in the process, then we should absolutely do that. It creates a much less stressful work environment for all stakeholders if you know there are safeguards in place.

Josh Morony

If you enjoyed this article, feel free to share it with others!

Tweet

Discussion

Need some help with this tutorial? Spotted an error? Got some helpful advice for others? Join the discussion on Twitter

If there are no active discussions, start one by including the URL of this post and tag me (**@joshuamorony**) in a new tweet.

I'll try to help out directly whenever I have the time, but you might also want to include other relevant tags to attract attention from others who might also be able to help. To make it super easy for others to help you out, you might consider setting up an example on **Stack Blitz** so others can jump right into your code.

Pull requests

If you find an error or some outdated code that you would like to help fix, feel free to **send me a pull request** on GitHub

Education and resources for creating ***NEXT LEVEL*** native web applications.

Become a PRO member

Get the Newsletter
Follow on Twitter
Subscribe on YouTube
Services
Pricing
Josh Morony
Books
About
Contact

Terms of Service Privacy Policy