

Exploring the Basic Pull Request Process Through ChatOps

This lesson explores the step by step lifecycle of a pull request through the perspective of ChatOps.

We'll cover the following

- Creating a new project
- Creating a pull request
- Inspecting the automatically created comment and checks
 - serverless-jenkins
 - tide
- Assigning the PR
- Unassigning the PR
- Approving the PR
 - /approve and /lgtn
- Exploring the OWNERS file
- Modifying the OWNERS file
- Adding a collaborator in the project to approve PR

The best way to explore the integration Jenkins X provides between Git, **Prow**, and the rest of the system is through practical examples. The first thing we'll need is a project, so we'll create a new one.

Creating a new project



Make sure that you are not inside an existing repository before executing the command that follows.

```
jx create quickstart \  
--filter golang-http \  

```



```
--project-name jx-prow \  
--batch-mode  
  
cd jx-prow  
  
jx get activities \  
--filter jx-prow --watch
```

We created a Go-based project called `jx-prow`, entered the local copy of the Git repository `jx` created for us, and started watching the activity. After a while, the steps in the output will be in the `Succeeded` status, and we can stop the watcher by pressing `ctrl+c`.

Creating a pull request

Since most of the ChatOps features apply to pull requests, we need to create one.

```
git checkout -b chat-ops  
  
echo "ChatOps" | tee README.md  
  
git add .  
  
git commit \  
--message "My first PR with prow"  
  
git push --set-upstream origin chat-ops
```

We created a new branch `chat-ops`, we made a silly change to `README.md`, and we pushed the commit.

Now that we have the branch with the change to the source code, we should create a pull request. We could do that by going to GitHub UI but, as you already know from the [Creating Pull Requests](#) lesson, `jx` already allows us to do that through the command line. Given that I prefer the terminal screen over UIs (and you don't have a say in that matter), we'll go with the latter option.

```
jx create pullrequest \  
--title "PR with prow" \  
--body "What I can say?" \  
--batch-mode
```

We created a pull request and are presented with a confirmation message with a link. Please open it in your favorite browser.

Inspecting the automatically created comment and checks

checks

You will notice a few things right away. A comment was created describing the process we should follow with pull requests. In a nutshell, the PR needs to be approved. Someone should review the changes we are proposing. That might mean going through the code, performing additional manual tests, or anything else that the approver might think is needed before they give their OK.

Near the bottom, you'll see that a few checks are running.

serverless-jenkins

The *serverless-jenkins* process is executing the part of the pipeline dedicated to pull requests. At the end of the process, the application will be deployed to a temporary PR-specific environment. This is all the same as before; however, there is an important aspect of PR that we haven't explored yet. There are rules that we need to follow before we merge to the master and the communication happening between Git and **Prow**.

tide

The second activity is called *tide*. It will be in the *pending* state until we complete the process described in the comment.

Tide is one of the **Prow** components. It is in charge of merging the pull request to the master branch, and it is configured to do so only after we send it the `/approve` command. Alternatively, Tide might close the pull request if it receives `/approve cancel`. We'll explore both soon.

[APPROVALNOTIFIER] This PR is **NOT APPROVED**

This pull-request has been approved by:

To fully approve this pull request, please assign additional approvers.

We suggest the following additional approver: **vfarcic**

If they are not already assigned, you can assign the PR to them by writing `/assign @vfarcic` in a comment when ready.

The full list of commands accepted by this bot can be found [here](#).

The pull request process is described [here](#)

▼ Details

Needs approval from an approver in each of these files:

- **OWNERS**

Approvers can indicate their approval by writing `/approve` in a comment

Approvers can cancel approval by writing `/approve cancel` in a comment

 **vfarcic** added the `size/XS` label 4 minutes ago

Add more commits by pushing to the **chat-ops** branch on **vfarcic/jx-prow**.



Some checks haven't completed yet

[Hide all checks](#)

2 pending checks

●  **serverless-jenkins** Pending — Not all Tasks in the Pipeline have finished exe...

●  **tide** Pending — Not mergeable. Needs approved label.

[Details](#)



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

The message that explains the rules governing the pull request process

Next, we'll wait until the Jenkins X build initiated by the creation of the PR is finished. Once it's done, you'll see a comment stating *"PR built and available in a preview environment."* Feel free to click the link next to it to open the application in a browser.

The "real" indication that the build is finished is the *serverless-jenkins — All Tasks have completed executing* message in the *checks* section near the bottom of the screen. When we created the PR, a webhook request was sent to **Prow** which, in

turn, notified the system that it should run a build of the associated pipeline (the one defined in `jenkins-x.yml`). Not only did **Prow** initiate the build, but it also monitored its progress. Once the build was finished, **Prow** communicated the outcome to GitHub.

Assigning the PR

If you take another look at the description of the pull request process, you'll see that you can assign it to someone. We could, as you probably already know, do that through the standard GitHub process by clicking a few buttons. Since we're trying to employ the ChatOps process, we'll write a comment with a slash command instead.

Please type `/assign` as the comment. Feel free to add any text above or below the command. You can, for example, write something similar to the following text.

```
/assign
```

```
This PR is urgent, so please review it ASAP
```

The text does not matter, it is only informative, and you are free to write anything you want. What matters is the slash command `/assign`. When we create the comment, GitHub will notify **Prow**. It will parse it, it will process all slash commands, and it will discard the rest. Please click the *Comment* button once you're done writing a charming humanly readable message that contains `/assign`. Just remember that you cannot mix commands and text in the same line.



GitHub might not refresh automatically. If you do not see the expected result, try reloading the screen.

A moment after we created the comment, the list of assignees in the right-hand menu changed. **Prow** automatically assigned the PR to you. Now, that might seem silly at first. *Why would you assign it to yourself?* We need someone to review it, and that someone should be anybody but you. You already know what's inside the PR and you are confident that it worked as expected, otherwise, you wouldn't have created it. We need a second pair of eyes. However, we don't have any collaborators on the project yet, so you're the only one **Prow** could assign it to. We'll change that soon.

PR with prow #1

Open

vfarcic wants to merge 1 commit into master from chat-ops

Edit

Conversation 0

Commits 1

Checks 0

Files changed 1

+1 -1

vfarcic commented 4 minutes ago

Owner + 😊 ...

What I can say?

My first PR with prow 2bbfa3d vfarcic

vfarcic commented 4 minutes ago

Author Owner + 😊 ...

[APPROVALNOTIFIER] This PR is **NOT APPROVED**

This pull-request has been approved by:

To fully approve this pull request, please assign additional approvers.

We suggest the following additional approver: **vfarcic**

If they are not already assigned, you can assign the PR to them by writing `/assign @vfarcic` in a comment when ready.

The full list of commands accepted by this bot can be found [here](#).

The pull request process is described [here](#)

▼ Details

Needs approval from an approver in each of these files:

Reviewers

No reviews

Assignees

vfarcic

Labels

size/XS

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you were assigned.

Assigned pull request

Unassigning the PR

Just as we can assign a PR to someone, we can also unassign it through a slash command. Please type `/unassign` as a new comment and click the *Comment* button. A moment later your name will disappear from the list of assignees.

🔍 From now on, I'll skip surrounding slash commands with humanly-readable text. It's up to you to choose whether to make it pretty or just type bare-bones commands like in the examples. From the practical perspective, the result will be the same since **Prow** cares only about the commands and it discards the rest of the text. Just remember that command and text cannot be mixed in the same line.

When we issued the command `/assign`, and since we were not specific, **Prow** decided who it would be assigned to. We can be more precise and choose who reviews our pull request. Since you are currently the only collaborator on the

project, we have limited possibilities, but we'll try it out nevertheless.

Please type `/assign @YOUR_GITHUB_USER` as the comment. Make sure to replace `YOUR_GITHUB_USER` with your user (keep `@`). Once you're done, click the *Comment* button. The result should be the same as when we issued the `/assign` command without a specific user simply because there is only one collaborator this pull request can be assigned to. If we'd had more (as we will soon), we could have assigned it to someone else.


Approving the PR

Next, a reviewer would go through the code changes and confirm that everything works correctly. The fact that *serverless-jenkins* check completed without issues provides an indication that all the validations executed as part of the pipeline build were successful. If still in doubt, a reviewer can always open the application deployed to the PR-specific temporary environment by clicking the link next to the *PR built and available in a preview environment* comment.

We'll assume that you (acting as a reviewer) believe that the change is safe to merge to the master branch. You already know that will initiate another pipeline build that will end with the deployment of the new release to the staging environment.

`/approve` and `/lgTM`

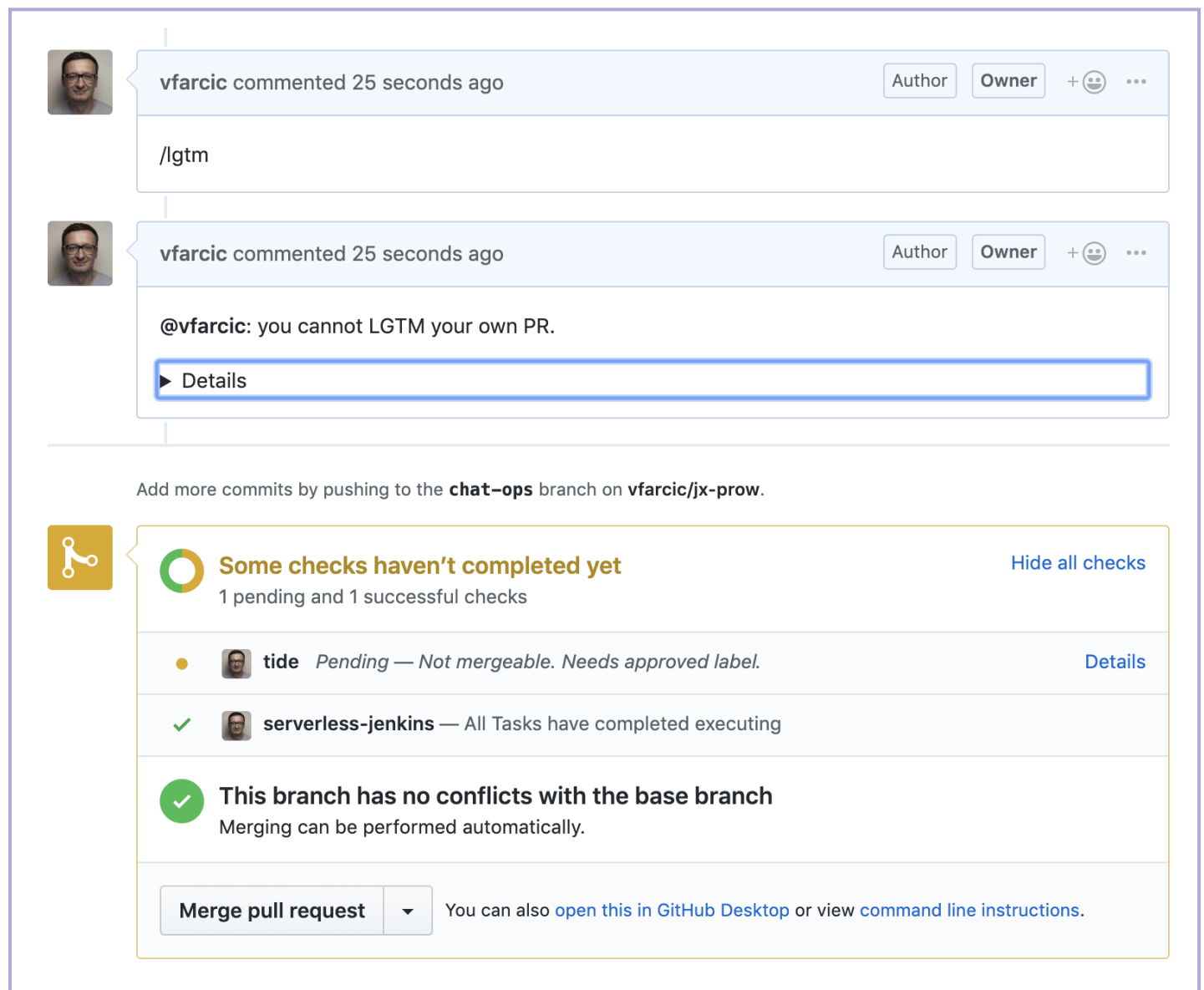
As you already saw by reading the description of the PR process, all we have to do is type `/approve`. But we won't do that. Instead, we'll use the `lgTM` abbreviation that stands for *looks good to me*.

 Originally, `/lgTM` is meant to provide a label that is typically used to gate merging. It is an indication that a reviewer confirmed that the pull request could be approved. However, Jenkins X implementation sets it to act as approval as well. Therefore, both `/approve` and `/lgTM` commands serve the same purpose. Both can be used to approve PRs. We'll use the latter mostly because I like how it sounds.

So, without further ado, please type `/lgTM` and click the *Comment* button.

A moment later, we should receive a notification (a comment) saying that “you

A moment later, we should receive a notification (a comment) saying that *you cannot LGTM your own PR*” (remember that you might need to refresh your screen). That makes sense, doesn’t it? Why would you review your own code? There is no benefit in that since it would neither result in knowledge sharing nor additional validations. The system is protecting us from ourselves and from making silly mistakes.



The screenshot displays a GitHub pull request interface. At the top, a comment by user 'vfarctic' is shown, stating they commented 25 seconds ago with the command '/lgtm'. Below this, another comment by 'vfarctic' is shown, stating they commented 25 seconds ago with the message '@vfarctic: you cannot LGTM your own PR.' and a 'Details' link. Below the comments, a section titled 'Add more commits by pushing to the chat-ops branch on vfarctic/jx-prow.' is visible. Below this, a section titled 'Some checks haven't completed yet' (with a link to 'Hide all checks') shows a summary of 1 pending and 1 successful check. The pending check is from 'tide' with the message 'Pending — Not mergeable. Needs approved label.' and a 'Details' link. The successful check is from 'serverless-jenkins' with the message 'All Tasks have completed executing'. Below the checks, a green checkmark indicates 'This branch has no conflicts with the base branch' with the message 'Merging can be performed automatically.' At the bottom, there is a 'Merge pull request' button and a link to 'open this in GitHub Desktop' or view 'command line instructions'.

Rejected lgtm attempt

If we are to proceed, we’ll need to add a collaborator to the project. Before we do that, I should comment that if `/lgtm` worked, we could use `/lgtm cancel` command. I’m sure you can guess what it does.

Before we explore how to add collaborators, approvers, and reviewers, we’ll remove you from the list of assignees. Since you cannot approve your own PR, it doesn’t make sense for it to be assigned to you.

Please type `/unassign` and click the *Comment* button. You’ll notice that your name

disappeared from the list of assignees.

Exploring the **OWNERS** file

We need to define who is allowed to review and who can approve our pull requests. We can do that by modifying the **OWNERS** file that is generated when we create the project through the Jenkins X quickstart. Since it would be insecure to allow a person who made the PR to change that file, the one that counts is the **OWNERS** file in the master branch. So, that's the one we'll explore and modify.

```
git checkout master  
  
cat OWNERS
```

The output is as follows.

```
approvers:  
- vfarci  
reviewers:  
- vfarci
```

The **OWNERS** contain the list of users responsible for the codebase of this repository. It is split between the **approvers** and **reviewers** sections. Such a split is useful if we'd like to implement a two-phase code review process where different people are in charge of reviewing and approving pull requests. However, more often than not, those two roles are performed by the same people so Jenkins X comes without a two-phase review process out of the box (it can be changed though).

Modifying the **OWNERS** file

To proceed, we need a real GitHub user, so please contact a colleague or a friend and ask them to give you a hand. Tell them that you'll need their help to complete some of the steps of the exercises that follow. Also, let them know that you need to know their GitHub user.



Feel free to ask someone for help in [DevOps20](#) Slack if you do not have a GitHub friend around. I'm sure that someone will be happy to act as a reviewer and an approver of your pull request.

We'll define two environment variables that will help us create a new version of

the `OWNERS` file. `GH_USER` will hold your username, while `GH_APPROVER` will contain

the user of the person that will be allowed to review and approve your pull requests. Typically, we would have more than one approver so that the review and approval tasks are distributed across the team. For demo purposes, the two of you should be more than enough.



Before executing the commands that follow, please replace the first [...] with your GitHub user and the second with the user of the person that will approve your PR.

```
GH_USER=[...]  
GH_APPROVER=[...]
```

Now we can create a new version of the `OWNERS` file. As already discussed, we'll use the same users as both reviewers and approvers.

```
echo "approvers:  
- $GH_USER  
- $GH_APPROVER  
reviewers:  
- $GH_USER  
- $GH_APPROVER  
" | tee OWNERS
```

All that's left, related to the `OWNERS` file, is to push the changes to the repository.

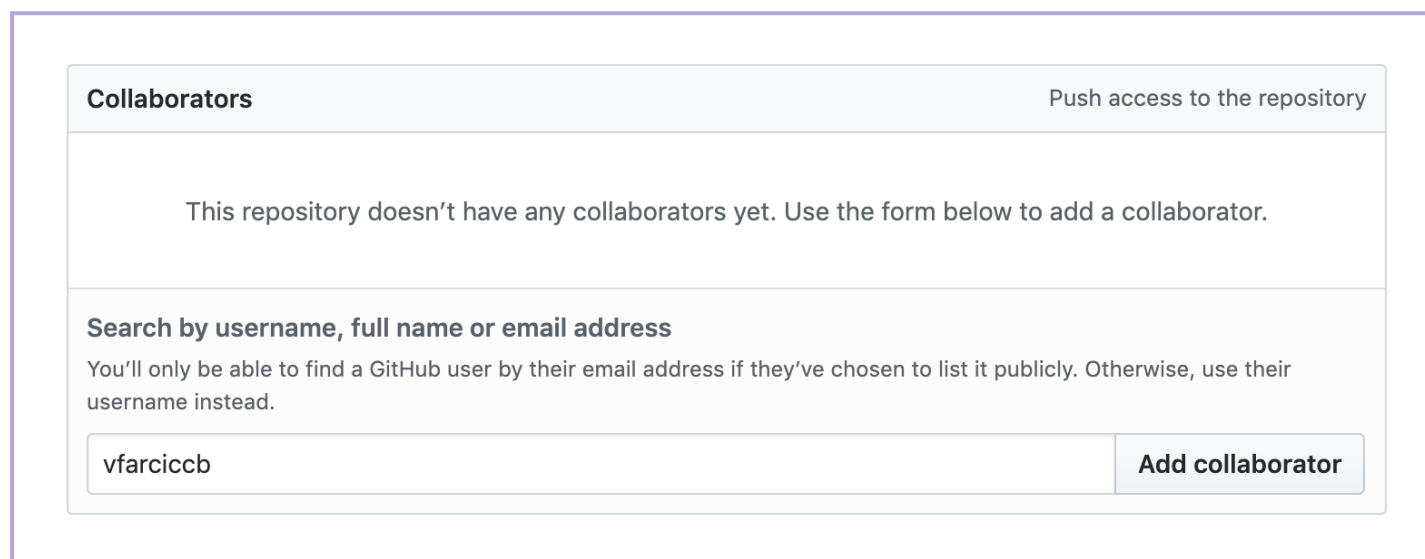
```
git add .  
  
git commit \  
    --message "Added an owner"  
  
git push
```

Adding a collaborator in the project to approve PR

Even though the `OWNERS` file defines who can review and approve pull requests, that would be useless if those users are not allowed to collaborate on your project. We need to tell GitHub that your colleague works with you by adding a collaborator (other Git platforms might call it differently).


```
open "https://github.com/$GH_USER/ix-new/settings/collaboration"
```

Please login if you're asked to do so. Type the user and click the *Add collaborator* button.

A screenshot of the GitHub 'Collaborators' interface. At the top, there's a header with 'Collaborators' on the left and 'Push access to the repository' on the right. Below this is a message: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Underneath is a section titled 'Search by username, full name or email address' with a sub-note: 'You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.' There is a search input field containing the text 'vfarciicb' and a button labeled 'Add collaborator' to its right.


GitHub collaborators screen

Your colleague should receive an email with an invitation to join the project as a collaborator. Make sure that they accept the invitation.

 Not all collaborators need to be in the **OWNERS** file. You might have people who collaborate on your project that are not allowed to review or approve pull requests.

Now we can assign the pull request to the newly added approver. Please go to the pull request screen and make sure that you are logged in (as you, not as the approver). Type **/assign @[...]** as the comment, where **[...]** is replaced with the username of the approver. Click the *Comment* button.

The approver should receive a notification email. Please let them know that they should go to the pull request (instructions are in the email), type **/approve**, and click the *Comment* button.

 Please note that **/approve** and **/lgtm** have the same purpose in this context. We're switching from one to another only to show that they both result in the pull request being merged to the master branch.

After a while, the PR will be merged, and a build of the pipeline will be executed. That, as you already know, results in a new release being validated and deployed to the staging environment.

You will notice that email notifications are flying back and forth between you and the approver. Not only that we are applying ChatOps principles, but we are simultaneously solving the need for notifications that let each person involved know what's going. Those notifications are sent by Git itself as a reaction to specific actions. The way to control who receives which notifications are particular to each Git platform and I hope that you already know how to subscribe, unsubscribe, or modify Git notifications you're receiving.

As an example, the email sent as the result of approving the PR is as follows.

```
[APPROVALNOTIFIER] This PR is APPROVED
```

```
This pull-request has been approved by: vfarcticb
```

```
The full list of commands accepted by this bot can be found here.
```

```
The pull request process is described here
```

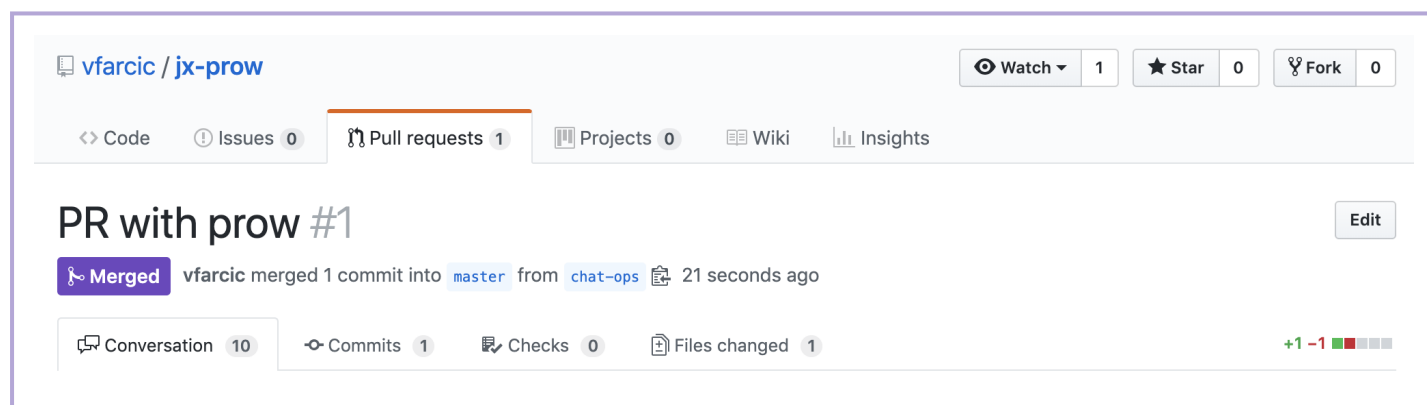
```
Needs approval from an approver in each of these files:
```

```
OWNERS [vfarcticb]
```

```
Approvers can indicate their approval by writing /approve in a comment
```

```
Approvers can cancel approval by writing /approve cancel in a comment
```

All in all, the pull request is approved. As a result, **Prow** merged it to the master branch, and that initiated a pipeline build that ended with the deployment of the new release to the staging environment.



vfarcic / jx-prow

Watch 1 Star 0 Fork 0

Code Issues 0 Pull requests 1 Projects 0 Wiki Insights

PR with prow #1

Merged vfarcic merged 1 commit into master from chat-ops 21 seconds ago

Conversation 10 Commits 1 Checks 0 Files changed 1

+1 -1

Please wait until the *All checks have passed* message appears in the PR.

We already know from past that a merge to the master branch initiates yet another build. That did not change with the introduction of ChatOps. When we approved the PR, **Prow** merged it to the master branch, and from there the same processes were executed as if we merged manually.

Next, we will look at some additional available slash commands.