

# Pull Request Commands

Project Skara provides contributors and reviewers with additional pull request commands that enable additional functionality. A *pull request command* is a comment made to a pull request that starts with a slash ("/"), for example `/integrate`, `/csr` or `/sponsor`. The command may appear on any line and with an arbitrary amount of whitespace in front of it, but it must be the first non whitespace characters that appear on a particular line. Pull request commands can also be placed *at the end* of the pull request body. Pull request commands can be used with [draft](#) pull requests.

Skara will only evaluate a given command once, so if you make a mistake and get an error message back, you need to enter a new command in a new comment. Editing the previous comment will not have any effect.

Note that if you are using Skara on Gitlab, there are built in commands that clash with some of the Skara commands. To enforce use of the Skara command and not the Gitlab variant, you can put some whitespace in front of the command in the comment.

## Commands

- [/integrate](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/sponsor](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/issue](#)
  - [Syntax](#)
  - [Alias](#)
  - [Description](#)
  - [Examples](#)
- [/summary](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/contributor](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/csr](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/reviewer](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/reviewers](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)
- [/label](#)
  - [Syntax](#)
  - [Alias](#)
  - [Description](#)
  - [Examples](#)
- [/clean](#)
  - [Syntax](#)
  - [Description](#)
- [/help](#)
  - [Syntax](#)
  - [Description](#)
  - [Examples](#)

## `/integrate`

### Syntax

```
/integrate [auto|defer|undefer] <hash>
```

### Description

The pull request command that all contributors will use is the `/integrate` command that integrates an approved pull request into a repository. This is an example where the Skara workflow differs slightly from the workflow offered by most external Git source code hosting providers - almost all external Git source code hosting providers require that a reviewer/maintainer integrates a pull request into a repository. Skara instead enables the *contributor* to integrate the pull request with the `/integrate` command, but the contributor can only issue the `/integrate` command once the pull request passes all pre-integration checks (e.g. jcheck).

The `/integrate` command will by default [squash](#) all commits in the pull request into one, [rebase](#) the resulting commit on top of the target branch and automatically create an appropriate commit message. The squashing of all commits in the pull request enables contributors to update a pull request by simply pushing to the branch in the contributor's personal fork the pull request was created from. The rebasing of the resulting commit enables contributors to simply merge the target branch into the source branch for the pull request whenever changes from the target branch needs to be incorporated (instead of doing complicated rebases and force pushes). The automatic formatting of the commit message frees contributors from having to consider the details of the commit message format.

A hash can be supplied to `/integrate` and in that case an *atomic* integration is performed. An atomic integration squashes and rebases on the commits on top of the given hash, and then tries to push the result. An atomic integration will fail if the supplied hash is *not* the head of the target branch at the moment of the push. This means that you can be sure that if you supply a hash to `/integrate`, then your pull request will only be squashed and rebased on top of the given commit, nothing else. This can be useful for large and complicated changes when you are unsure about potential conflicts with other commits.

The `auto` parameter is used to label a pull request to be automatically integrated as soon as all pre-integration checks are passing. This can be a good idea to save time when a change is comparatively benign and only the minimum amount of review is needed.

If a contributor of a pull request will be unable to perform the integration at a suitable time, they may defer the ability to integrate to any other committer in the project. This is done using `/integrate defer`. Once performed, any committer in the project is able to issue the `/integrate` command in the affected pull request. This can be undone by the contributor running `/integrate undefer`.

## Examples

- `/integrate`
- `/integrate 38d3c3d937675ac5d550659825b7e99ed1eb3921`

## /sponsor

### Syntax

```
/sponsor [<hash>]
```

### Description

Marks you as the [sponsor](#) of the pull request *and* integrates the pull request. A contributor who is not a [Committer](#) must first issues the `/integrate` pull request command to mark a pull request as ready for integration. Once the pull request author has issued the `/integrate` pull request command, a [Committer](#) must then issue the `/sponsor` pull request command to integrate the pull request. The `/sponsor` pull request take an optional hash for atomic integrations, just like the `/integrate` pull request command.

## Examples

- `/sponsor`
- `/sponsor 38d3c3d937675ac5d550659825b7e99ed1eb3921`

## /issue

### Syntax

```
/issue [add|remove] <id>[,<id>,...]
```

### Alias

```
/solves
```

### Description

Mark or clear one or more issues as solved by this pull request. The default action is to mark an issue as being solved. All issues solved by the pull request will be part of the resulting commit message. An issue that has wrongly been marked as solved by this pull request can be removed by the command `/issue remove <id>`. It is allowed to prefix the issue numeric id with the JBS project name, but it is not necessary.

## Examples

- `/issue JDK-4567890`
- `/solves JDK-456789`
- `/issue add JDK-4567890`
- `/issue add 4567890`
- `/issue add 1234567,4567890`
- `/issue remove JDK-4567890`

## /summary

### Syntax

```
/summary [<text>]
```

### Description

Add a summary section to the resulting commit message of the pull request. For details on the commit message syntax, see [JEP 357](#). A summary command alone without any text will remove an existing summary.

### Examples

- `/summary This is a one-line summary`
- `/summary`  
This is a multi-line summary.  
You can add as many lines as you like.
- `/summary`  
This is a multi-line, multi-paragraph summary.  
You can have as many lines and as many paragraphs as you like.  
  
This is the first line second paragraph,  
and this is the second line in the second paragraph.
- `/summary`

## /contributor

### Syntax

```
/contributor (add|remove) [@user | openjdk-user | Full Name <email@address>]
```

### Description

Adds or removes a user as a contributor to this pull request. A contributor can be specified either by their GitHub username (e.g. @openjdk-bot), their OpenJDK username (e.g. duke) or via a full-name and email combination (e.g. J. Duke <duke@openjdk.org>). A contributor that has incorrectly been listed as a contributor can be unlisted by issuing the command `/contributor remove <id>`. The contributors will be included in the final commit message for the pull request. For full details on the commit message syntax see [JEP 357](#).

### Examples

- `/contributor add ehelin`
- `/contributor add @edvbl`
- `/contributor add J. Duke <duke@openjdk.org>`
- `/contributor remove @edvbl`
- `/contributor remove rwestberg`
- `/contributor remove J. Duke <duke@openjdk.org>`

## /csr

### Syntax

```
/csr [needed|unneeded]
```

### Description

Requires that the pull requested has a [JBS](#) issue associated *and* that the [JBS](#) issue has [CSR](#) request associated with it *and* that the [CSR](#) request is approved *before* the pull request can be integrated.

### Examples

- `/csr needed`
- `/csr unneeded`
- `/csr`

## /reviewer

### Syntax

```
/reviewer (credit|remove) <username>[,<username>,...]
```

## Description

Give additional users credit for reviewing a pull request. The usernames can either be a username of the source code hosting provider (e.g. a GitHub username) or an OpenJDK username. Note that not all OpenJDK projects allows the pull request author to credit additional reviewers. A reviewer credited via `/reviewer credit` will not count as a verified reviewer and some OpenJDK projects do not count unverified reviewers as enough for integration. A reviewer can be removed by issuing the `/reviewer remove` command.

## Examples

- `/reviewer credit @edvbld`
- `/reviewer credit ehelin`
- `/reviewer remove ehelin`

## /reviewers

### Syntax

```
/reviewers N [role]
```

### Description

Require that at least `N` users with given `role` (defaults to [Author](#)) review the pull request before it can be integrated. The requirements are in *addition* to the ones specified by the `.jcheck/conf` file. For example, if the `.jcheck/conf` file requires [1 Reviewer](#), then issuing the command `"/reviewers 2"` means that [1 Reviewer](#) and [1 Author](#) is required to integrate the pull request.

### Examples

- `/reviewers 2`
- `/reviewers 3 reviewer`

## /label

### Syntax

```
/label [add|remove] <label>[,<label>,...]
```

### Alias

```
/cc
```

### Description

Adds or removes labels on the pull request. If no action is specified, then the action defaults to "add". Labels have the same name as the development mailing lists without the `-dev` suffix, e.g. the label "hotspot" corresponds to the "hotspot-dev" mailing list. The mailing list bridge will send the RFR e-mail according to the labels on the pull request. Note that only labels associated with mailing lists can be controlled with this command and not Skara specific labels which are controlled by the bots or other specific commands (e.g. `clean`, `rfr`).

### Examples

- `/label add hotspot`
- `/label remove build,core-libs`
- `/label client`
- `/cc hotspot-gc hotspot-runtime`
- `/cc core-libs`

## /clean

### Syntax

```
/clean
```

### Description

Marks a backport pull request as clean. Skara will attempt to automatically determine if a backport pull request is a clean backport, but sometimes this automatic check fails even if the pull request could be considered clean.

`/help`

## **Syntax**

`/help`

## **Description**

Shows help for all pull request commands.

## **Examples**

- `/help`