HowToContribute

How to Contribute to ZooKeeper

This page describes the mechanics of *how* to contribute software to ZooKeeper. For ideas about *what* you might contribute, please see the ProjectSuggestions page.

Getting the source code

First of all, you need the ZooKeeper source code. Go to https://github.com/apache/zookeeper (which is a read only mirror of zookeeper git hosted on Apache) and click the fork button to create your own fork. After forking download your fork to local disk. Most development is done on the "master":

```
$ git clone https://github.com/<userid>/zookeeper.git my-zookeeper
```

You may also want to develop against a specific release. You can list the release branches and checkout 3.4 by going into the zookeeper directory and running:

```
git branch -la
git checkout remotes/origin/branch-3.4
```

Making Changes

Before you start, send a message to the ZooKeeper developer mailing list, or file a bug report in Jira. Describe your proposed changes and check that they fit in with what others are doing and have planned for the project. Be patient, it may take folks a while to understand your requirements.

You will want to create a local branch to work under. It's convenient to name it the JIRA number:

```
git checkout -b ZOOKEEPER-1234 # this is the same as git branch ZOOKEEPER-1234 && git checkout ZOOKEEPER-1234
```

Modify the source code and add some (very) nice features using your favorite IDE.

But take care about the following points

- All public classes and methods should have informative Javadoc comments.
  - Do not use @author tags.
- Code should be formatted according to Sun's conventions, with these exceptions:
  - Indent four spaces per level, not two or six or eight, etc... four.
  - No tabs for indentation, spaces only
- Contributions should pass existing unit tests.
- New unit tests should be provided to demonstrate bugs and fixes. JUnit is our test framework:
  - You must implement a class that extends `junit.framework.TestCase` and whose class name ends with `Test`.
  - Define methods within your class whose names begin with `test`, and call JUnit's many assert methods to verify conditions; these methods will be executed when you run `ant test`.
  - By default, do not let tests write any temporary files to `/tmp`. Instead, the tests should write to the location specified by the `test.build.data` system property.
  - Place your class in the `src/java/test` tree.
  - `ClientTest.java` is an example of a client-server test.
  - You can run all the unit test with the command `ant test`, or you can run a specific unit test with the command `ant -Dtestcase=<class name without package prefix> test` (for example `ant -Dtestcase=ClientTest test`)

Contribute Work via Github Pull Request

We accept pull requests via github against the apache/zookeeper repository. Submitting to Apache ZooKeeper via github is no different than submitting any other pull request against a github repository. The developer needs to follow the contribution guidelines to produce changes, just like with patch files, and once it is ready, the developer produces a pull request from his/her own branch. The one constraints we have is that the title of the pull request must match the paring JIRA issue. If the JIRA issue is "ZOOKEEPER-222222: My lovely bug", then the pull request issue must also match "ZOOKEEPER-222222: My lovely bug".

We currently perform QA on pull requests, although the process is still being polished. We also have not completed the JIRA workflow when the developer submits a pull request, which means that the JIRA does not transition to **Patch Available** when a pull request is submitted. We are currently discussing on the list how to adapt our workflow.

Example workflow

there are many ways to setup your git environment but this one example:

fork the apache github repo

go to https://github.com/apache/zookeeper and click the fork button to create your own fork.

make a local clone of your fork

on your local machine run

```
$ git clone https://github.com/<userid>/zookeeper.git my-zookeeper
```

add the apache github repo

```
$ git remote add apache-github https://github.com/apache/zookeeper.git
```

make sure you are uptodate

```
$ git pull apache-github
```

create a branch named after the issue you are working on

```
$ git checkout remotes/apache-github/master
$ git checkout -B ZOOKEEPER-<jiraNumber>
```

<jiraNumber> is the JIRA you will be working on.

make your changes

sorry can't help you here. it's all up to you to fix the bug 🙂

commit everything

do you will want to do some combination of `git add` and `git commit` to get all your changes committed. when you run git commit, you only need a single line in the commit message it should have the form `ZOOKEEPER-jiraNumber: jiraTitle`.

push your change to git hub

```
$ git push origin ZOOKEEPER-<jiraNumber>
```

generate a pull request

go to the github page that lists your branches. you will see a pull request button that you can press. Please make sure your pull request description captures enough information so reviewers can get an idea of what the pull request is targeting without digging into the JIRA or looking into the code. When the pull request is merged, the description will be part of the commit message. Here is an example of a descriptive pull request looks like: https://github.com/apache/zookeeper/pull/211

Check List for Pull Request

This is a check list of things we'd like a contributor to do before opening a pull request.

Unit Tests

Before submitting your pull request, you are encouraged to run the same tools that the automated Jenkins CI system will run on your patch. This enables you to fix problems with your pull request before you submit it. The `test` Ant target will run your patch through the same checks that Jenkins currently does.

To use this target, you must run it from a clean workspace (ie git `status` shows no modifications or additions). From your clean workspace, run:

```
mvn verify spotbugs:check checkstyle:check -Pfull-build -Dsurefire-forkcount=4
```

At the end, you should get a message on your console that indicates success.

Please make sure that all unit tests succeed before submitting pull request and that no new javac compiler warnings are introduced by your pull request

After a while, if you see

```
BUILD SUCCESSFUL
```

all is ok, but if you see

```
BUILD FAILED
```

then please examine error messages in `build/test` and fix things before proceeding.

Javadoc

Please also check the javadoc.

```
> mvn install -DskipTests -pl zookeeper-docs
> firefox build/docs/api/index.html
```

Examine all public classes you've changed to see that documentation is complete, informative, and properly formatted. Your patch must not generate any javadoc warnings.

Documentation

The source documentation is available in src/docs directory. It can be regenerated using forrest (as documented here) however the contributor is not responsible for submitting this as part of the pull request. To be clear: the pull request should include changes to src/docs but not the toplevel (of the source tree) docs directory.

Please don't do in pull request:

- reformat code unrelated to the bug being fixed: formatting changes should be separate patches/commits.
- comment out code that is now obsolete: just remove it.
- insert comments around each change, marking the change: folks can use subversion to figure out what's changed and by whom.
- make things public which are not required by end users.

Please do in pull request:

- try to adhere to the coding style of files you edit;
- comment code whose function or rationale is not obvious;

Use GitHub’s "Co-authored-by" when there are multiple authors

Note: This is in case of manual copies, ToDo for me: I will check the commit script to integrate this functionality

There are occasions when there are multiple author for a patch. For example when there is a patch that is abandoned by its original author for any reason, and it can no longer be merged, or it's unfinished, someone might pick it up and finish it. (After asking the original author if he is still working on it or not).

In these occasions, we should also attribute the original author by adding one or more `Co-authored-by` trailers to the commit’s message. See the GitHub documentation for "Creating a commit with multiple authors".

As this is a GitHub feature, Co-author will show up on GitHub's statistics.

In short, these are the steps to add Co-authors that will be tracked by GitHub:

1. Collect the name and email address for each co-author.
2. Commit the change, but after your commit description, instead of a closing quotation, add two empty lines. (Do not close the commit message with a quotation mark)
3. On the next line of the commit message, type `Co-authored-by: name <name@example.com>`. After the co-author information, add a closing quotation mark.

Here is the example from the GitHub page, using 2 Co-authors:

```
$ git commit -m "Refactor usability tests.
>
>
Co-authored-by: name <name@example.com>
Co-authored-by: another-name <another-name@example.com>"
```

Final Checks on Pull Request

Please note that the attachment should be granted license to ASF for inclusion in ASF works (as per the Apache License §5).

Folks should run a full check (mvn verify spotbugs:check checkstyle:check -Pfull-build -Dsurefire-forkcount=4) before submitting pull request. Tests should all pass. Javadoc should report **no** warnings or errors. Jenkin's tests are meant to double-check things, and not be used as a primary patch tester, which would create too much noise on the mailing list and in Jira. Submitting patches that fail Jenkins testing is frowned on, (unless the failure is not actually due to the patch).

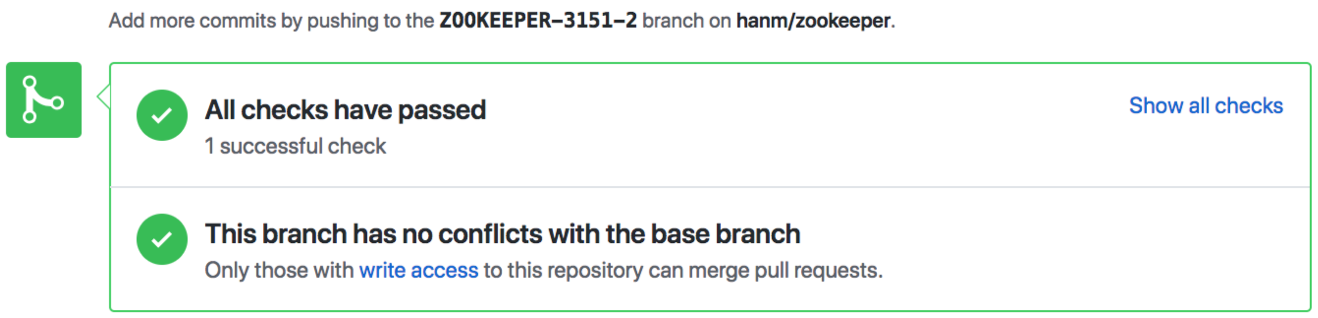
If your PR involves performance optimizations, they should be validated by benchmarks that demonstrate an improvement.

If your PR creates an incompatibility with the latest major release, then you must set the **Incompatible change** flag on the issue's Jira 'and' fill in the **Release Note** field with an explanation of the impact of the incompatibility and the necessary steps users must take.

If your PR implements a major feature or improvement, then you must fill in the **Release Note** field on the issue's Jira with an explanation of the feature that will be comprehensible by the end user.

Jenkins Pre-commit Check

Once a PR is created, a Jenkins job will be triggered to run a set of pre-commit checks on the pull request, including all Java and C unit tests, find bug test, release audit check, etc. If all checks pass, you should see something like this in your pull request page:


Add more commits by pushing to the **ZOOKEEPER-3151-2** branch on **hanm/zookeeper**.

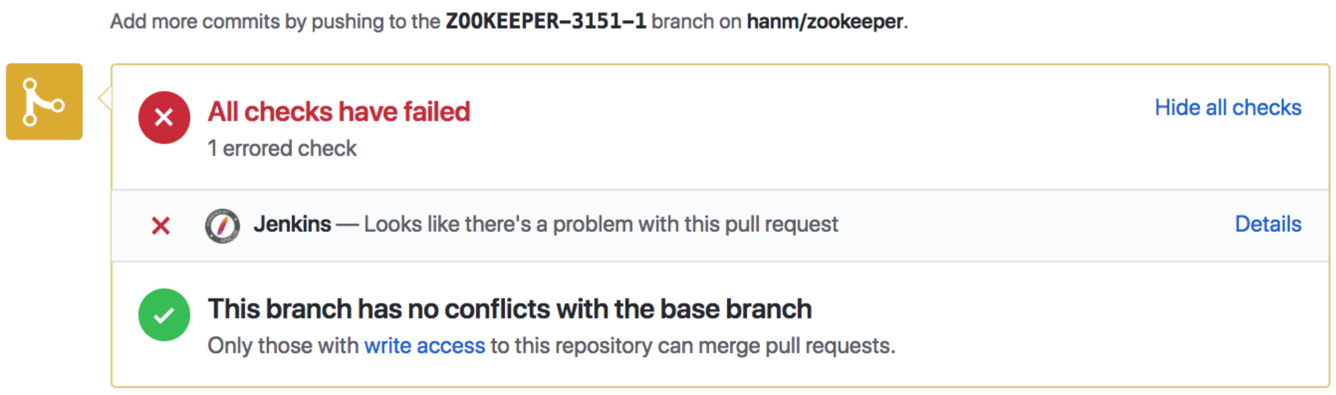
All checks have passed
1 successful check

Show all checks

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

After code review and committers sign off, the pull request will be merged.

If any of the checks fail, you will see this:


Add more commits by pushing to the **ZOOKEEPER-3151-1** branch on **hanm/zookeeper**.

All checks have failed
1 errored check

Hide all checks

Jenkins — Looks like there's a problem with this pull request

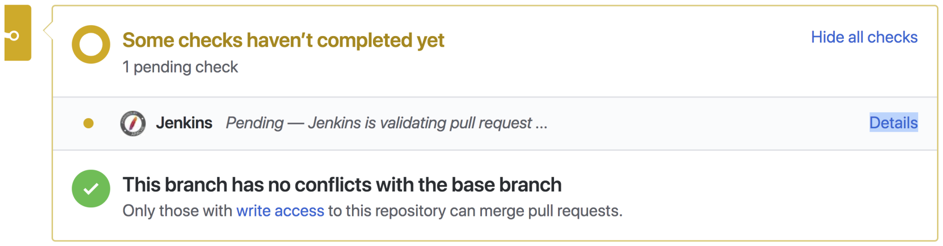
Details

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

In this case, please check the Jenkins output by following the "Details" link, and fix the issue, and try trigger a Jenkins build again. There are these ways to trigger a Jenkins build:

- Close and reopen the pull request. This is the recommended approach, and only the contributor who owns the pull request can do this.
- Update the PR by pushing a new commit or amend the previous commit (git commit --amend). This is also recommended approach and only contributor who owns the pull request can do this.
- Any committer or PMC member (actually anyone who is part of 'apache' github organization) can trigger the job just by adding a comment with the magic phrases:
    - '**retest ant build**' in order to re run ANT based build
    - '**retest maven build**' in order to re run MAVEN based build (update on Sep, 2020 - currently this trigger phrase is broken. Workaround is for committers to manually trigger the job in CI admin UI, or for contributors to do a commit with a new SHA - e.g. commit --amend then push to remote to trigger the job).

- Note for administrators: configuration of these phrases are in the precommit Jenkins jobs, it is using the GitHub Pull Request builder plugin)


Some checks haven't completed yet
1 pending check
Hide all checks
Jenkins Pending — Jenkins is validating pull request ...
Details
This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

For contributors who owns the pull request: please make sure to get a green build to prepare the patch in a landing state.

For committers: please make sure to get a green build before committing a pull request.

Code Review and Accept Pull Request

Once a "+1" comment is received from the automated patch testing system and a code reviewer has set the **Reviewed** flag on the issue's Jira, a committer should then evaluate it within a few days and either: commit it; or reject it with an explanation.

Please be patient. Committers are busy people too. If no one responds to your patch after a few days, please make friendly reminders. Please incorporate other's suggestions into your patch if you think they're reasonable. Finally, remember that even a patch that is not committed is useful to the community.

Should your patch receive a "-1" from the Jenkins testing, select the **Resume Progress** on the issue's Jira, update pull request with necessary fixes.

Committers: for non-trivial changes, it is best to get another committer to review your pull request before commit. Please also try to frequently review things in the patch queue.

Committing Guidelines for committers

Jira Guidelines

Please comment on issues in Jira, making their concerns known. Please also vote for issues that are a high priority for you.

Please refrain from editing descriptions and comments if possible, as edits spam the mailing list and clutter Jira's "All" display, which is otherwise very useful. Instead, preview descriptions and comments using the preview button (on the right) before posting them. Keep descriptions brief and save more elaborate proposals for comments, since descriptions are included in Jira's automatically sent messages. If you change your mind, note this in a new comment, rather than editing an older comment. The issue should preserve this history of the discussion.

Stay involved

Contributors should join the ZooKeeper mailing lists. In particular, the commit list (to see changes as they are made), the dev list (to join discussions of changes) and the user list (to help others).

See Also

- Apache contributor documentation
- Apache voting documentation