Committing changes

THIS IS THE OLD PROCEDURE! PLEASE USE: Merging Github Pull Requests

(i'm leaving this here for the moment until we decide if we want to archive it or just replace it with the other page.)

This document describes how to commit changes to ZooKeeper. It assumes a knowledge of git. While it is for committers to use as a guide, it also provides contributors an idea of how the commit process actually works.

In general we are very conservative about changing the ZooKeeper code base. It is ground truth for systems that use it, so we need to make sure that it is reliable. For this reason we use Review Then Commit (RTC) http://www.apache.org/foundation/glossary.html#ReviewThenCommit change policy.

Except for some very rare cases any change to the ZooKeeper code base will start off as a Jira. (In some cases a change may relate to more than one Jira. Also, there are cases when a Jira results in multiple commits.) Generally, the process of getting ready to commit when the Jira has a patch associated with it and the contributor decides that it is ready for review and marks it patch available. Sometimes, for bigger patches, review board is used to manage the reviews (reviews.apache.org). Once it is patch available hudson will run against the patch. **(First checklist item for committing: does the patch pass hudson?)**

A committer must sign off on a patch. It is very helpful if the community also reviews the patch, but in the end a committer must take responsibility for the correctness of the patch. If the match is simple enough and the committer feels confident in the review, a single +1 from a committer is sufficient to commit the patch. (Remember committers cannot review their own patch, so if a committer submits a patch, they should make sure that another committer reviews it.) For more complicated patches at least two committers should sign off (+1 it). The bylaws say that a -1 from a committer can be overridden, but hopefully we can avoid those situations. Even -1s from non-committers should be taken into account and hopefully resolved before committing. **(2nd checklist: have the appropriate number of committers accepted the change and no -1s?)**

With the required number of approvals, a committer (any committer can do this including the one that committed the patch) can now make the change to the code base. Here are the recommended steps for committing:

1. is this a github pull request? follow the instructions for `zk-merge-pr.py`. you don't need to follow the procedure below.
2. make sure the code is up-to-date
   `git pull`
3. create a branch for the patch
   `git checkout -b ZOOKEEPER-1234 # ZOOKEEPER-1234 is the JIRA number`
4. apply the patch
   `patch -p0 < patch_path`
5. run the tests are your machine as a final check. (seems redundant, but sometimes issues pop up.)
6. edit the CHANGES.TXT file and put the jiras that correspond to the patch in the appropriate section. add the Jira number. description (contributor via committer). a convention has emerged of using the committers id for brevity, but using the full names of non-committers. **(i don't think we do this any more)**
7. If the patch includes documentation changes, then regenerate the rendered docs by running "ant -Dforrest.home=<path to Forrest> clean docs". Modifications should now be shown under the docs folder. **However, DO NOT commit changes in docs/releasenotes.html or docs /releasenotes.pdf.** Release notes are handled separately as a release management activity, as documented here: HowToRelease - OUTDATED! .
8. git commit and push the changes
   ```
   git commit -a -m "ZOOKEEPER-1234: description (author via committer)" --author="John Doe <john@doe.org>"
   git push origin ZOOKEEPER-1234:master
   ```
9. resolve (make sure you "resolve" and not "close".) the jiras that correspond to the patch and put the commit message (that one that has the new revision number) in the resolution comment field.

If the Jira is a bug fix you may also need to commit the patch to the latest branch in git.

*NOTE: if you by mistake push a branch remotely (by just running `git push` for example) you can delete it using:

```
git push origin --delete ZOOKEEPER-1234  # where ZOOKEEPER-1234 is the remote branch to delete
```