HowToCommit

Guide for Hadoop Core Committers

This page contains Hadoop Core-specific guidelines for committers.

New committers

New committers are encouraged to first read Apache's generic committer documentation:

- Apache New Committer Guide
- Apache Committer FAQ

The first act of a new core committer is typically to add their name to the credits page. This requires changing the site source in https://github.com/apache/hadoop-site/blob/asf-site/src/who.md. Once done, update the Hadoop website as described here (TLDR; don't forget to regenerate the site with hugo, and commit the generated results, too).

Review

Hadoop committers should, as often as possible, attempt to review patches submitted by others. Ideally every submitted patch will get reviewed by a committer within a few days. If a committer reviews a patch they've not authored, and believe it to be of sufficient quality, then they can commit the patch, otherwise the patch should be cancelled with a clear explanation for why it was rejected.

The list of submitted patches is in the Hadoop Review Queue. This is ordered by time of last modification. Committers should scan the list from top-to-bottom, looking for patches that they feel qualified to review and possibly commit.

For non-trivial changes, it is best to get another committer to review your own patches before commit. Use "Submit Patch" like other contributors, and then wait for a "+1" from another committer before committing.

Reject

Patches should be rejected which do not adhere to the guidelines in HowToContribute and to the CodeReviewChecklist. Committers should always be polite to contributors and try to instruct and encourage them to contribute better patches. If a committer wishes to improve an unacceptable patch, then it should first be rejected, and a new patch should be attached by the committer for review.

Commit individual patches

Hadoop uses git for the main source. The writable repo is at - https://gitbox.apache.org/repos/asf/hadoop.git

Initial setup

We try to keep our history all linear and avoid merge commits. To this end, we highly recommend using git pull --rebase. In general, it is a good practice to have this *always* turned on. If you haven't done so already, you should probably run the following:

```
$ git config --global branch.autosetuprebase always
```

Also, we highly recommend setting username and email for git to use:

```
$ git config [--global] user.name <real-name>
$ git config [--global] user.email <email>@apache.org
```

More recommendations on how to use git with ASF projects can be found here

Committing a patch

When you commit a patch, please follow these steps:

1. **Commit locally:** Commit the change locally to the appropriate branch (should be *trunk* if it is not a feature branch) using `git commit -a -m <commit-message>`. The commit message should include the JIRA issue id, along with a short description of the change and the name of the contributor if it is not you. *Note:* Be sure to get the issue id right, as this causes JIRA to link to the change in git (use the issue's "All" tab to see these). Verify all the changes are included in the commit using `git status`. If there are any remaining changes (previously missed files), please commit them and squash these commits into one using `git rebase -i`.

2. **Pull latest changes from remote repo:** Pull in the latest changes from the remote branch using `git pull --rebase` (--rebase is not required if you have setup git pull to always --rebase). Verify this didn't cause any merge commits using `git log [--pretty=oneline]`

3. **Push changes to remote repo:** Build and run a test to ensure it is all still kosher. Push the changes to the remote (main) repo using `git push <remote> <branch>`.
4. **Backporting to other branches:** If the changes were to trunk, we might want to apply them to other appropriate branches.
    a. Cherry-pick the changes to other appropriate branches via `git cherry-pick -x <commit-hash>`. The -x option records the source commit, and reuses the original commit message. Resolve any conflicts.
    b. If the conflicts are major, it is preferable to produce a new patch for that branch, review it separately and commit it. When committing an edited patch to other branches, please follow the same steps and make sure to include the JIRA number and description of changes in the commit message.
    c. When backporting to branch-2.7 or older branches, we need to update CHANGES.txt.
5. Resolve the issue as fixed, thanking the contributor. Follow the rules specified at Apache Hadoop Release Versioning for how to set fix versions appropriately, it's important for tracking purposes with concurrent release lines.
6. Set the assignee if it is not set. If you cannot set the contributor to the assignee, you need to add the contributor into Contributors role in the project. Please see Adding Contributors role for the detail.

This How-to-commit video has guidance on the commit process, albeit using svn. Most of the process is still the same, except that we now use git instead.

Merging a feature branch

When merging a feature branch to trunk, use no fast forward option to provide a single commit to digest history of the feature branch. Commit history of feature branch will remain in feature branch.

```
# Start a new feature
git checkout -b new-feature trunk
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Merge in the new-feature branch
git checkout trunk
git merge --no-ff new-feature
```

Committing Documentation

Hadoop's official documentation is authored using hugo. To commit documentation changes you must have Hugo installed (single binary available for all the platforms, part of the package repositories, brew/pacman/yum...). Documentation is of two types:

1. End-user documentation, versioned with releases; and,
2. The website.

The end user documentation is maintained in the main repository (hadoop.git) and the results are committed to the hadoop-site during each release. The website itself is managed in the hadoop-site.git repository (both the source and the rendered form).

To commit end-user documentation create a patch as usual and modify the content of src/site directory of any hadoop project (eg. ./hadoop-common-project/hadoop-auth/src/site). You can regenerate the docs with `mvn site`. End-user documentation is only published to the web when releases are made, as described in HowToRelease.

To commit changes to the website and re-publish them:

```
git clone https://gitbox.apache.org/repos/asf/hadoop-site.git -b asf-site
#edit site under ./src
hugo
# add both the ./src and ./content directories (source and rendered version)
git add .
git commit
git push
```

The commit will be reflected on Apache Hadoop site automatically.

Note: you can check the rendering locally: with `hugo serve && firefox http://localhost:1313`

Patches that break HDFS, YARN and MapReduce

In general, the process flow is that Jenkins notices the checkin and automatically builds the new versions of the common libraries and pushes them to Nexus, the Apache Maven repository.

However, to speed up the process or if Jenkins is not working properly, developers can push builds manually to Nexus. To do so, they need to create a file in ~/.m2/settings.xml that looks like:

```
<settings>
  <servers>
    <!-- To publish a snapshot of some part of Maven -->
    <server>
      <id>apache.snapshots.https</id>
      <username> <!-- YOUR APACHE SVN USERNAME --> </username>
      <password> <!-- YOUR APACHE SVN PASSWORD --> </password>
    </server>
    <!-- To publish a website of some part of Maven -->
    <server>
      <id>apache.website</id>
      <username> <!-- YOUR APACHE SSH USERNAME --> </username>
      <filePermissions>664</filePermissions>
      <directoryPermissions>775</directoryPermissions>
    </server>
    <!-- To stage a release of some part of Maven -->
    <server>
      <id>apache.releases.https</id>
      <username> <!-- YOUR APACHE SVN USERNAME --> </username>
      <password> <!-- YOUR APACHE SVN PASSWORD --> </password>
    </server>
    <!-- To stage a website of some part of Maven -->
    <server>
      <id>stagingSite</id>
      <!-- must match hard-coded repository identifier in site:stage-deploy -->
      <username> <!-- YOUR APACHE SSH USERNAME --> </username>
      <filePermissions>664</filePermissions>
      <directoryPermissions>775</directoryPermissions>
    </server>
  </servers>
</settings>
```

After you have committed the change to Common, do an "ant mvn-publish" to publish the new jars.

As a security note, since the settings.xml file contains your Apache svn password in the clear, I prefer to leave the settings file encrypted using gpg when I'm not using it. I also don't ever publish from a shared machine, but that is just me being paranoid. 🙂

Adding Contributors role

There are about three roles (Administrators, Committers, Contributors) for each project. (Common/HDFS/MapReduce/YARN)

- Contributors who have Contributors role can become assignee of the issues in the project.
- Committers who have Committers role can set arbitrary roles in addition to Contributors role.
- Committers who have Administrators role can edit or delete all comments, or even delete issues in addition to Committers role.

How to set roles

1. Login to ASF JIRA
2. Go to the project page (e.g. https://issues.apache.org/jira/browse/HADOOP)
3. Hit "Administration" tab
4. Hit "Roles" tab in left side
5. Add Administrators/Committers/Contributors role

"Contributors 1" role was created by INFRA-12487 because we hit the upper limit of the Contributors role. If you cannot add a contributor to Contributors role, try "Contributors 1".

Dialog

Committers should hang out in the #hadoop room on irc.freenode.net for real-time discussions. However any substantive discussion (as with any off-list project-related discussion) should be re-iterated in JIRA or on the developer list.