How To Contribute

How to Contribute to Apache Hadoop

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

- Dev Environment Setup
  - Get the source code
  - Read BUILDING.txt
  - Integrated Development Environment (IDE)
  - Build Tools
  - Native libraries
  - Hardware Setup
- Making Changes
  - Generating a patch
    - Choosing a target branch
    - Unit Tests
    - Javadoc
  - Provide a patch
  - Testing your patch
  - Changes that span projects
- Contributing your work
  - Submitting patches against object stores such as Amazon S3, OpenStack Swift and Microsoft Azure
- Requesting for a Jira account
- Jira Guidelines
- Stay involved
- See Also

Dev Environment Setup

Here are some things you will need to build and test Hadoop. Be prepared to invest some time to set up a working Hadoop dev environment. Try getting the project to build and test locally first before you start writing code.

Get the source code

First of all, you need the Hadoop source code. The official location for Hadoop is the Apache Git repository. See Git And Hadoop

Read BUILDING.txt

Once you have the source code, we strongly recommend reading BUILDING.txt located in the root of the source tree. It has up to date information on how to build Hadoop on various platforms along with some workarounds for platform-specific quirks. The latest BUILDING.txt for the current trunk can also be viewed on the web.

Integrated Development Environment (IDE)

You are free to use whatever IDE you prefer or your favorite text editor. Note that:

- Building and testing is often done on the command line or at least via the Maven support in the IDEs.
- Set up the IDE to follow the source layout rules of the project.
- Disable any added value "reformat" and "strip trailing spaces" features as it can create extra noise when reviewing patches.

Build Tools

Please see BUILDING.txt for the detail.

As the Hadoop builds use the external Maven repository to download artifacts, Maven needs to be set up with the proxy settings needed to make external HTTP requests. The first build of every Hadoop project needs internet connectivity to download Maven dependencies.

1. Be online for that first build, on a good network
2. To set the Maven proxy setttings, see http://maven.apache.org/guides/mini/guide-proxies.html
3. Because Maven doesn't pass proxy settings down to the Ant tasks it runs HDFS-2381 some parts of the Hadoop build may fail. The fix for this is to pass down the Ant proxy settings in the build Unix: mvn $ANT_OPTS; Windows: mvn %ANT_OPTS%.
4. Tomcat is always downloaded, even when building offline. Setting -Dtomcat.download.url to a local copy and -Dtomcat.version to the version pointed to by the URL will avoid that download.

If you are failing to fetch a artifact from the external maven repository, you may need to delete the related files from your local cache (i.e. ~/.m2 directory).

- Ref: **HADOOP-16577** - Getting issue details... STATUS

Native libraries

On Linux, you need the tools to create the native libraries: LZO headers,zlib headers, gcc, OpenSSL headers, cmake, protobuf dev tools, and libtool, and the GNU autotools (automake, autoconf, etc).

For RHEL (and hence also CentOS):

```
 yum -y install  lzo-devel  zlib-devel  gcc gcc-c++ autoconf automake libtool openssl-devel fuse-devel cmake
```

For Debian and Ubuntu:

```
 apt-get -y install maven build-essential autoconf automake libtool cmake zlib1g-dev pkg-config libssl-dev
libfuse-dev
```

Native libraries are mandatory for Windows. For instructions see Hadoop2OnWindows.

Hardware Setup

- Lots of RAM, especially if you are using a modern IDE. ECC RAM is recommended in large-RAM systems.
- Disk Space. Always handy.
- Network Connectivity. Hadoop tests are not guaranteed to all work if a machine does not have a network connection -and especially if it does not know its own name.
- Keep your computer's clock up to date via an NTP server, and set up the time zone correctly. This is good for avoiding change-log confusion.

Making Changes

Before you start, send a message to the Hadoop 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. If you want to start with pre-existing issues, look for Jiras labeled newbie. You can find them using this filter.

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 must be formatted according to Sun's conventions, with one exception:
    - Indent two spaces per level, not four.
- Code formatter xml is present here: https://github.com/apache/hadoop/tree/trunk/dev-support/code-formatter . IntelliJ users can directly import `hadoop_idea_formatter.xml`
- Contributions must 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 uses @Test annotations for all test methods. Please note, Hadoop uses JUnit v4.
    - 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 mvn test. Please add meaningful messages to the assert statement to facilitate diagnostics.
    - 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.
    - If a HDFS cluster or a MapReduce/YARN cluster is needed by your test, please use org.apache.hadoop.dfs.MiniDFSCluster and org.apache.hadoop.mapred.MiniMRCluster (or org.apache.hadoop.yarn.server.MiniYARNCluster), respectively. TestMiniMRLocalFS is an example of a test that uses MiniMRCluster.
    - Place your class in the src/test tree.
    - TestFileSystem.java and TestMapRed.java are examples of standalone MapReduce-based tests.
    - TestPath.java is an example of a non MapReduce-based test.
    - You can run all the project unit tests with mvn test, or a specific unit test with mvn -Dtest=<class name without package prefix> test. Run these commands from the hadoop-trunk directory.
- If you modify the Unix shell scripts, see the UnixShellScriptProgrammingGuide.

Generating a patch

Choosing a target branch

Except for the following situations it is recommended that all patches be based off trunk to take advantage of the Jenkins pre-commit build.

1. The patch is targeting a release branch that is not based off trunk e.g. branch-3.1, branch-2.10, etc.
2. The change is targeting a specific feature branch and is not yet ready for merging into trunk.

If you are unsure of the target branch then **trunk** is usually the best choice. Committers will usually merge the patch to downstream branches e.g. branch-3.2 as appropriate.

Unit Tests

Please make sure that all unit tests succeed before constructing your patch and that no new javac compiler warnings are introduced by your patch.

For building Hadoop with Maven, use the following to run all unit tests and build a distribution. The -Ptest-patch profile will check that no new compiler warnings have been introduced by your patch.

```
mvn clean install -Pdist -Dtar -Ptest-patch
```

Any test failures can be found in the target/surefire-reports directory of the relevant module. You can also run this command in one of the hadoop-common, hadoop-hdfs, or hadoop-mapreduce directories to just test a particular subproject.

Unit tests development guidelines HowToDevelopUnitTests

Javadoc

Please also check the javadoc.

```
mvn process-sources javadoc:javadoc-no-fork
firefox target/site/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.

Jenkins includes a javadoc run on Java 8 and Java 11, it will fail if there are unbalanced HTML tags or <p/> clauses (use <p> here.

If Jenkins rejects a patch due to Java 8 or Java 11 javadoc failures, it is considered an automatic veto for the patch.

Provide a patch

You need to create a pull request in https://github.com/apache/hadoop. Now attaching a patch in ASF JIRA does not work. You need to set the title which starts with the corresponding JIRA issue number (e.g. HADOOP-XXXXX. Fix a typo in YYY.) to integrate with the issue. If there is no corresponding issue, please create an issue in ASF JIRA before creating a pull request.

See also: GitHub Integration

Testing your patch

Before submitting your patch, you are encouraged to run the same tools that the automated Jenkins patch test system will run on your patch. This enables you to fix problems with your patch before you submit it. The dev-support/bin/test-patch script in the trunk directory will run your patch through the same checks that Jenkins currently does *except* for executing the unit tests. (See TestPatchTips for some tricks.)

Run this command from a clean workspace (ie git status shows no modifications or additions) as follows:

```
dev-support/bin/test-patch [options] patch-file | defect-number
```

At the end, you should get a message on your console that is similar to the comment added to Jira by Jenkins's automated patch test system, listing +1 and -1 results. Generally you should expect a +1 overall in order to have your patch committed; exceptions will be made for false positives that are unrelated to your patch. The scratch directory (which defaults to the value of ${user.home}/tmp) will contain some output files that will be useful in determining cause if issues were found in the patch.

Some things to note:

- the optional cmd parameters will default to the ones in your PATH environment variable
- the grep command must support the -o flag (Both GNU grep and BSD grep support it)
- the patch command must support the -E flag

Run the same command with no arguments to see the usage options.

Changes that span projects

You may find that you need to modify both the common project and MapReduce or HDFS. Or perhaps you have changed something in common, and need to verify that these changes do not break the existing unit tests for HDFS and MapReduce. Hadoop's build system integrates with a local maven repository to support cross-project development. Use this general workflow for your development:

- Make your changes in common
- Run any unit tests there (e.g. 'mvn test')
- *Publish* your new common jar to your local mvn repository:

  ```
  hadoop-common$ mvn clean install -DskipTests
  ```

- A word of caution: mvn install pushes the artifacts into your local Maven repository which is shared by all your projects.
- Switch to the dependent project and make any changes there (e.g., that rely on a new API you introduced in hadoop-common).
- Finally, create separate patches for your common and hdfs/mapred changes, and file them as separate JIRA issues associated with the appropriate projects.

Contributing your work

1. Please note that the commits in the GitHub PR should be granted license to ASF for inclusion in ASF works (as per the Apache License §5).
2. Folks should run mvn clean install javadoc:javadoc checkstyle:checkstyle before opening PR.

    a. Tests must all pass.
    b. Javadoc should report **no** warnings or errors.
    c. The Javadoc on java 8 must not fail.
    d. Checkstyle's error count should not exceed that listed at lastSuccessfulBuild/artifact/trunk/build/test/checkstyle-errors.html
3. Jenkins'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 PR.
4. If your patch involves performance optimizations, they should be validated by benchmarks that demonstrate an improvement.
5. If your patch 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.
6. If your patch 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.

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.

Submitting patches against object stores such as Amazon S3, OpenStack Swift and Microsoft Azure

The modules hadoop-aws, hadoop-openstack and hadoop-azure contain filesystem clients which work with Amazon S3, OpenStack Swift and Microsoft Azure storage respectively.

The test suites for these modules are not executed on Jenkins because they need credentials to work with.

Having Jenkins +1 any patch against an object store does not mean the patch works: it must be manually tested by the submitter, the committer and any other reviewers who can do so

If a Yetus patch run says +1 for an object store patch, all it means is "the compilation, javadoc and style checks passed". It does not mean the patch works, or that it is ready to be committed.

The details of how to test for these object stores are covered in the filesystem specification documentation.

When submitting a patch, make sure the patch does not include any of your secret credentials. The Hadoop .gitignore file is set to ignore specific XML test resources for this purpose.

```
hadoop-common-project/hadoop-common/src/test/resources/contract-test-options.xml
hadoop-tools/hadoop-openstack/src/test/resources/contract-test-options.xml
hadoop-tools/hadoop-aws/src/test/resources/auth-keys.xml
hadoop-tools/hadoop-aws/src/test/resources/contract-test-options.xml
hadoop-tools/hadoop-azure/src/test/resources/azure-auth-keys.xml
```

Please state which infrastructures you have tested against, —including which regions you tested against. If you have not tested the patch yourself, do not expect anyone to look at the patch.

We welcome anyone who can test these patches: please do so and again, declare what you have tested against. That includes in-house/proprietary implementations of the APIs as well as public infrastructures.

Requesting for a Jira account

If you wish to contribute to Hadoop and require a Jira account, such requests can be made via: https://selfserve.apache.org/jira-account.html

---

Note:

- Jira account is required only if someone has to report a bug or plan to contribute, other cases are unnecessary to request account.
- Serving such requests can take time upto one week or even more during holidays.
- Please refrain from sending the form multiple times or sending follow-ups on the mailing lists.

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.

Additionally, do not set the Fix Version. Committers use this field to determine which branches have had patches committed. Instead, use the Affects and Target Versions to notify others of the branches that should be considered.

Stay involved

Contributors should join the Hadoop 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