HiveDeveloperFAQ

Hive Developer FAQ

Maven

Run the test and generate the output file using the appropriate `-Dtest`Hive is using Maven as its build tool. Versions prior to 0.13 were using Ant.

Developing

How do I move some files?

Post a patch for testing purposes which simply does add and deletes. SVN will not understand these patches are actually moves, therefore you should actually upload the following, **in order** so the last upload is the patch for testing purposes:

1. A patch which has only the non-move changes for commit e.g. HIVE-XXX-for-commit.patch
2. A script of of commands required to make the moves HIVE-XXX-moves.sh
3. A patch for testing purposes HIVE-XXX.patch

The script should be a set of `svn mv` commands along with any `perl` commands required for find/replace. For example:

```
$ svn mv MyCLass.java MyClass.java
$ perl -i -pe 's<at:var at:name="MyCLass" />MyClass@g' MyClass.java
```

How do I add a new MiniDriver test?

See MiniDriver Tests for information about MiniDriver and Beeline tests.

For FAQ about how to run tests, see Testing below.

Building

- See Getting Started: Building Hive from Source for detailed information about building Hive releases 0.13 and later with Maven.

- See Installing from Source Code (Hive 0.12.0 and Earlier) for detailed information about building Hive 0.12 and earlier with Ant.

Maven settings

You might have to set the following Maven options on certain systems to get build working: Set MAVEN_OPTS to "-Xmx2g -XX:MaxPermSize=256M".

How to build all source?

The way Maven is set up differs between the master branch and branch-1. In branch-1, since both Hadoop 1.x and 2.x are supported, you need to specify whether you want to build Hive against Hadoop 1.x or 2.x. This is done via Maven profiles. There is a profile for each version of Hadoop, `hadoop-1` and `hadoop-2`. For most Maven operations one of these profiles needs to be specified or the build will fail.

In master, only Hadoop 2.x is supported, thus there is no need to specify a Maven profile for most build operations.

In master:

```
mvn clean install -DskipTests
cd itests
mvn clean install -DskipTests
```

In branch-1, MVN:

```
mvn clean install -DskipTests -Phadoop-2
cd itests
mvn clean install -DskipTests -Phadoop-2
```

To build against Hadoop 1.x, switch the above to `-Phadoop-1`.

For the remainder of this page we will assume master and not show the profiles. However, if you are working on branch-1 remember that you will need to add in the appropriate profile.

How do I import into Eclipse?

Build and generate Eclipse files (the conservative method).

For master – without using a local Maven repository:

```
$ mkdir workspace
$ cd workspace
$ git clone https://github.com/apache/hive.git
$ cd hive
$ mvn clean package eclipse:clean eclipse:eclipse -Pitests -DskipTests -DdownloadSources -DdownloadJavadocs
```

NOTE: The presence of the **package** target is crucial in successfully generating the project files.

NOTE: It is recommended to wipe out or rename the *org/apache/hive* subtree of the local m2 repository prior to generating the project files; and you may want to check after its execution that it does not contain any files – if it does contain something, then your project files may not be fully independent of downloaded Maven artifacts.

This will work on all branches; but keep in mind that in this case you are installing the Hive artifacts into your local repository.

```
$ mkdir workspace
$ cd workspace
$ git clone https://github.com/apache/hive.git
$ cd hive
$ mvn clean install -DskipTests
$ mvn eclipse:clean
$ mvn eclipse:eclipse -DdownloadSources -DdownloadJavadocs
$ cd itests
$ mvn clean install -DskipTests
$ mvn eclipse:clean
$ mvn eclipse:eclipse -DdownloadSources -DdownloadJavadocs
```

In Eclipse define M2_REPO in Preferences -> Java -> Build Path -> Classpath Variables to either:

Mac Example

```
/Users/$USER/.m2/repository
```

Linux Example

```
/home/$USER/.m2/repository
```

Windows Example

```
C:/users/$USER/.m2/repository
```

Then import the workspaces. If you get an error about "restricted use of Signal" for Beeline and CLI, follow these instructions.

Note that if you use the Hive git base directory as the Eclipse workspace, then it does not pick the right project names (for example, picks 'ant' instead of 'hive-ant'). Therefore it's recommended to have the workspace directory one up from the git directory. For example workspaces/*hive-workspace*/*hive* where *hive-workspace* is the Eclipse workspace and *hive* is the git base directory.

How to generate tarball?

```
mvn clean package -DskipTests -Pdist
```

It will then be located in the packaging/target/ directory.

How to generate protobuf code?

```
cd ql
mvn clean install -DskipTests -Pprotobuf
```

How to generate Thrift code?

```
mvn clean install -Pthriftif -DskipTests -Dthrift.home=/usr/local
```

Don't forget to update `hive_metastore.proto` when changing `hive_metastore.thrift`

Unable to render Jira issues macro, execution error.

How to run findbugs after a change?

```
mvn site -Pfindbugs
```

Note: Available in Hive 1.1.0 onward (see HIVE-8327).

How to compile ODBC?

```
cd odbc
mvn compile -Podbc -Dthrift.home=/usr/local -Dboost.home=/usr/local
```

How do I publish Hive artifacts to my local Maven repository?

```
mvn clean install -DskipTests
cd itests
mvn clean install -DskipTests
```

Testing

For general information, see Unit Tests and Debugging in the Developer Guide.

How do I run precommit tests on a patch?

Hive precommit testing is triggered automatically when a file is uploaded to the JIRA ticket:

1. Attach the patch file to a JIRA ticket: in the ticket's "More" tab, select "Attach Files" and use "Choose File" to upload the file, then add a descriptive comment.
2. Put the patch in the review queue: click the "Submit Patch" button. The button name will change to "Cancel Patch" and the ticket status will change to Patch Available.

See Hive PreCommit Patch Testing for more detail.

How do I rerun precommit tests over the same patch?

For patch updates, our convention is to number them like HIVE-1856.1.patch, HIVE-1856.2.patch, etc. And then click the "Submit Patch" button again when a new one is uploaded; this makes sure it gets back into the review queue.

How do I run a single test?

ITests

Note that any test in the itests directory needs to be executed from within the itests directory. The pom is disconnected from the parent project for technical reasons.

Single test class:

```
mvn test -Dtest=ClassName
```

Single test method:

```
mvn test -Dtest=ClassName#methodName
```

Note that a pattern can also be supplied to -Dtests to run multiple tests matching the pattern:

```
mvn test -Dtest='org.apache.hive.beeline.*'
```

For more usage see the documentation for the Maven Surefire Plugin.

How do I run all of the unit tests?

```
mvn test
cd itests
mvn test
```

Note that you need to have previously built and installed the jars:

```
mvn clean install -DskipTests
cd itests
mvn clean install -DskipTests
```

How do I run all of the unit tests except for a certain few tests?

Similar to running all tests, but define test.excludes.additional to specify a test/pattern to exclude from the test run. For example the following will run all tests except for the CliDriver tests:

```
cd itests
mvn test -Dtest.excludes.additional='**/Test*CliDriver.java'
```

How do I run the clientpositive/clientnegative unit tests?

All of the below require that you have previously run `ant package`.

To run clientpositive tests

```
cd itests/qtest
mvn test -Dtest=TestCliDriver
```

To run a single clientnegative test alter1.q

```
cd itests/qtest
mvn test -Dtest=TestNegativeCliDriver -Dqfile=alter1.q
```

To run all of the clientpositive tests that match a regex, for example the partition_wise_fileformat tests

```
cd itests/qtest
mvn test -Dtest=TestCliDriver -Dqfile_regex=partition_wise_fileformat.*

# Alternatively, you can specify comma separated list with "-Dqfile" argument
mvn test -Dtest=TestMiniLlapLocalCliDriver -Dqfile='vectorization_0.q,vectorization_17.q,vectorization_8.q'
```

To run a single contrib test alter1.q and overwrite the result file

```
cd itests/qtest
mvn test -Dtest=TestContribCliDriver -Dqfile=alter1.q -Dtest.output.overwrite=true
```

How do I remote debug a qtest?

```
cd itests/qtest
mvn -Dmaven.surefire.debug="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000 -Xnoagent -
Djava.compiler=NONE" test -Dtest=TestCliDriver -Dqfile=<test>.q
```

How do I modify the init script when testing?

The option to skip the init script or supply a custom init script was added in Hive 2.0 (see HIVE-11538).

To skip initialization:

```
mvn test -Dtest=TestCliDriver -Phadoop-2 -Dqfile=test_to_run.q  -DinitScript=
```

To supply a custom script:

```
mvn test -Dtest=TestCliDriver -Phadoop-2 -Dtest.output.overwrite=true -Dqfile=test_to_run.q  -
DinitScript=custom_script.sql
```

How do I update the output of a CliDriver testcase?

```
cd itests/qtest
mvn test -Dtest=TestCliDriver -Dqfile=alter1.q -Dtest.output.overwrite=true
```

How do I update the results of many test cases?

Assume that you have a file like below which you'd like to re-generate output files for. Such a file could be created by copying the output from the precommit tests.

```
head -2 /tmp/failed-TestCliDriver-file-tests
org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver_allcolref_in_udf
org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver_annotate_stats_join
```

You can re-generate all those output files in batches of 20 with the command below

```
egrep 'TestCliDriver' /tmp/failed-TestCliDriver-file-tests | perl -pe 's@.*testCliDriver_@@g' | awk '{print $1
".q"}' | xargs -n 30 | perl -pe 's@ @,@g' | xargs -I{} mvn test -Dtest=TestCliDriver -Dtest.output.
overwrite=true -Dqfile={}
```

To do the same from the output of a precommit result, with multiple drivers, you can do

```
import re
from itertools import groupby
s = """
org.apache.hadoop.hive.cli.TestBeeLineDriver.testCliDriver[drop_with_concurrency] (batchId=231)
org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[comments] (batchId=35)
org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[vector_if_expr] (batchId=141)
"""
PAT = re.compile("org.apache.hadoop.hive.cli.([^\.]*).*\[([^\]]*).*")
l = [PAT.match(x.strip()) for x in s.split("\n") if x.strip()]
for driver,q in groupby(sorted([a.groups() for a in l if a]), key=lambda a:a[0]):
        print """mvn clean test -Dtest=%s '-Dqfile=%s' -Dtest.output.overwrite=true""" % (driver, ",".join(["%s.
q" % a[1] for a in q]))
```

Where is the log output of a test?

Logs are put in a couple locations:

- `From the root of the source tree: find . -name hive.log`
- `/tmp/$USER/ (Linux) or $TMPDIR/$USER/ (MacOS)`

See Hive Logging for details about log files, including alternative configurations.

How do I add a test case?

First, add the test case to the qfile test suite:

- Copy the test to a new file under `ql/src/test/queries/clientpositive/<filename>.q` (or `/clientnegative` if it is a negative test).
  - If the new test creates any table, view, function, etc., make sure that the name is unique across tests. For instance, name a table in the test file `foo.q`, `foo_t1` instead of simply `t1`. This will help reduce flakiness in the test runs, since Jenkins will run tests and batches, and currently it does not restore to former state after running each of the q files.
  - If there is any interaction with file system, use unique folders for the test to avoid any collision with other tests.
- Add the `<filename.q>` to `itests/src/test/resources/testconfiguration.properties` to the appropriate variable (ex. `minimr.query.files`).

Next, generate the golden (output) file the first time you run the test case:

- Compile the Hive source from the top level Hive directory:

  ```
  mvn clean install -DskipTests
  ```

- Compile the itests:

```
cd itests
mvn clean install -DskipTests
```

- Run the test and generate the output file using the appropriate `-Dtest` (ex. `TestCliDriver`; see `itests/qtest/pom.xml` for the names of other test suites):

```
cd qtest
mvn test -Dtest=TestCliDriver -Dqfile=<filename>.q -Dtest.output.overwrite=true
```

- Add your output file to `ql/src/test/results/clientpositive/<filename>.q.out` (or `/clientnegative` if it is a negative test).

With the above steps, you can create a patch which has a `.java` file, a `.q` file and a `.q.out` file.

Why isn't the itests pom connected to the root pom?

It would be great to have it connected, but it would make it harder to use **mvn test** locally. The best option would be to utilize the failsafe plugin for integration testing; but it needs a bit different setup, and it's harder to use for now.... If you'd like to give that a try, by all means, go ahead.

There is an option to attach all the itest subprojects to the main project by enabling this with **-Pitests** (HIVE-13490).

There are some good and bad sides using this, it's introduced as a profile to clearly communicate that the integration tests are attached to the main project.

The good side is that you may freely use **-Pitests** to run integration tests from the root project without the need of **mvn install**.

```
mvn test -q -Pitests -Dtest=TestCliDriver -Dqtest=sample2.q
```

The bad side is that a simple **mvn test -Pitests** will start executing all integration tests.

Why does a test fail with a NullPointerException in MiniDFSCluster?

If any test fails with the error below it means you have an inappropriate umask setting. It should be set to 0022.

```
java.lang.NullPointerException: null
    at org.apache.hadoop.hdfs.MiniDFSCluster.startDataNodes(MiniDFSCluster.java:426)
    at org.apache.hadoop.hdfs.MiniDFSCluster.<init>(MiniDFSCluster.java:284)
    at org.apache.hadoop.hdfs.MiniDFSCluster.<init>(MiniDFSCluster.java:124)
```

Why do Spark unit tests fail with a SecurityException?

If you get the following errors in the unit tests:

```
2016-01-07T09:51:49,365 ERROR [Driver[]]: spark.SparkContext (Logging.scala:logError(96)) - Error initializing
SparkContext.
java.lang.SecurityException: class "javax.servlet.FilterRegistration"'s signer information does not match signer
information of other classes in the same package
```

It happens because there are two conflicting versions of the same classes "javax.servlet:servlet-api" and "org.eclipse.jetty.orbit:javax-servlet". Spark requires the eclipse version, but most tools including Hadoop and Jetty depend on the javax one. To avoid this problem, we need to exclude the javax version everywhere it comes up. Fortunately, maven has a tool to do that with:

```
mvn dependency:tree -Dverbose
```

which prints out the dependency tree. Go to each directory with the failing unit tests and search the dependency tree for "javax.servlet:servlet-api" and use exclusions in the pom.xml to remove it. See HIVE-12783 for an example.

Debugging

How do I debug into a single test in Eclipse?

You can debug into a single JUnit test in Eclipse by first making sure you've built the Eclipse files and imported the project into Eclipse as described here. Then set one or more breakpoints, highlight the method name of the JUnit test method you want to debug into, and do `Run->Debug`.

Another useful method to debug these tests is to attach a remote debugger. When you run the test, enable the debug mode for surefire by passing in "`-Dmaven.surefire.debug`". Additional details on how to turning on debugging for surefire can be found here. Now when you run the tests, it will wait with a message similar to

```
Listening for transport dt_socket at address: 5005
```

Note that this assumes that you are still using the default port 5005 for surefire. Otherwise you might see a different port. Once you see this message, in Eclipse right click on the project you want to debug, go to "Debug As -> Debug Configurations -> Remote Java Application" and hit the "+" sign on far left top. This should bring up a dialog box. Make sure that the host is "localhost" and the port is "5005". Before you start debugging, make sure that you have set appropriate debug breakpoints in the code. Once ready, hit "Debug". Now if you go back to the terminal, you should see the tests running and they will stop at the breakpoints that you set for debugging.

How do I debug my queries in Hive?

You can also interactively debug your queries in Hive by attaching a remote debugger. To do so, start Beeline with the "`--debug`" option.

```
$ beeline --debug
Listening for transport dt_socket at address: 8000
```

Once you see this message, in Eclipse right click on the project you want to debug, go to "Debug As -> Debug Configurations -> Remote Java Application" and hit the "+" sign on far left top. This should bring up a dialog box. Make sure that the host is the host on which the Beeline CLI is running and the port is "8000". Before you start debugging, make sure that you have set appropriate debug breakpoints in the code. Once ready, hit "Debug". The remote debugger should attach to Beeline and proceed.

```
$ beeline --debug
Listening for transport dt_socket at address: 8000
Beeline version 1.2.0 by Apache Hive
beeline>
```

At this point, run your queries as normal and it should stop at the breakpoints that you set so that you can start debugging.

This method should work great if your queries are simple fetch queries that do not kick off MapReduce jobs. If a query runs in a distributed mode, it becomes very hard to debug. Therefore, it is advisable to run in a "local" mode for debugging. In order to run Hive in local mode, do the following:

MRv1:

```
SET mapred.job.tracker=local
```

MRv2 (YARN):

```
SET mapreduce.framework.name=local
```

At this point, attach the remote debugger as mentioned before to start debugging your queries.