Contributing to Apache DataFu

Principles

The following principles were followed when writing this guide:

1. Commits are associated with an issue in JIRA.
2. Commits should be made in a branch and merged into master by a committer.
3. The author and committer should not be the same person.
4. Commits should retain the original author.
5. Commits committed by a committer should include "Signed-off-by" in the message.
6. Commits should include a link to the JIRA in the message.
7. Commits should include unit tests.

File a JIRA

File a JIRA here describing the change you propose to make. It's usually a good idea to file a JIRA soon so that people are aware of the issue or proposal and can respond with comments.

Setup Git

Make sure that you have your `user.name` and `user.email` configured in Git. This is so that the commits you make are attributed to you.

```
git config --global user.name
git config --global user.email
```

Both of these commands should return information. If they don't you can set each through the same commands and supplying your name and email. The name should be your full name.

Making Code Changes

If you have not already done so, get the code:

```
git clone https://git-wip-us.apache.org/repos/asf/incubator-datafu.git
cd incubator-datafu
```

Make sure you're working with an up to date copy of the master branch, in case you cloned previously:

```
git checkout master
git pull
```

Create a branch for your change. **This is very important,** as it makes it easier to submit your patch later, especially if you end up creating multiple commits. Pick a name for the branch that will help you identify it. Naming the branch after the issue name in JIRA is one good option. For example, if you created issue DATAFU-123, you could name the branch DATAFU-123 or DATAFU-123-some-issue-description.

```
git checkout -b DATAFU-123
```

Make your code changes. You will be committing to your branch. It's fine to complete the code changes in multiple commits. These can be squashed together into a single commit when the patch is created.

All changes should also be unit tested. If you are fixing an issue then you should add tests that demonstrate the issue (i.e. test should fail). Your fix should make the tests pass. If you are submitting new code then you should write tests that fully test the functionality.

Preparing Code For Review

Once you think your code is ready for review you can create a patch. To do this, use the `format-patch` command. This bundles together all the commits in your branch that are not in master into a single patch file.

Before creating the patch though, note whether you have multiple commits or not. If you do have multiple commits you should squash them into one so that the patch only consists of one commit. If you only have one commit then you can skip this step. We do this for the sake of having a clean commit history. Each issue should correspond to one commit. We can do this with a temporary throwaway branch. For example, if branch DATAFU-123 has our fix, we can create a branch DATAFU-123-squash and merge the changes into there.

```
git checkout master
git checkout -b DATAFU-123-squashed
git merge --squash DATAFU-123
git commit -m "This is a description of the change"
```

Now create the patch. For issue DATAFU-123 we would name the patch DATAFU-123.patch:

```
git format-patch master --stdout > DATAFU-123.patch
```

Attach the patch to the JIRA you created (if you prefer, you can open a GitHub pull request to this branch in your fork of the DataFu repository).

If you created a squashed branch, such as DATAFU-123-squashed, you can delete it with this command:

```
git branch -d DATAFU-123-squashed
```

Open a review board request here with a title such as "DATAFU-123 Some issue description". Upload your patch to the review board. Add a link for the review board you created in the JIRA issue. Be sure to add the group DataFu to the review board. This will cause the review request to be emailed to the DataFu dev mailing list.

Someone should respond to your request soon.

If you need to make additional changes (in response to feedback for example), you should append a version to any new patches you upload. For example, if a second patch was created for DATAFU-123 it could be named DATAFU-123-v2.patch. Attach the new patch to the JIRA and also upload it to Review Board.

How Code Gets Committed (by committers)

Once committers have signed off on your change, someone will commit the code to the master branch. Usually this will be one of the people who have reviewed the code and given their +1.

To commit the code, a committer would first start by creating a branch for the change. **This is very important.** You must create a branch before applying a patch. Otherwise the changes would be made directly to master.

For example, if the change was for issue DATAFU-123:

```
git checkout master
git pull
git checkout -b DATAFU-123
```

It's usually a good idea to check whether the patch will apply cleanly to the branch before proceeding:

```
git apply --check DATAFU-123.patch
```

Now the patch is committed to the branch. Note that we add the `-signoff` flag so that a sign off message is added to the commit. **This is important** as it keeps a record of who acutally committed the change, as opposed to the author. The patch retains the original author.

```
git am --signoff DATAFU-123.patch
```

The patch should apply cleanly if it was generated based on an up to date copy of master. If there are conflicts then try to resolve them, if possible. If resolving the conflicts is too complicated then raise this with the author and have them update their branch to master.

Run the unit tests. At the very least run the unit tests associated with the change.

To merge in the changes, simply merge the branch back into master and push. For example, for branch DATAFU-123:

```
git checkout master
git merge DATAFU-123
git push origin master
```

Now delete the branch:

```
git branch -d DATAFU-123
```