How to Contribute

Where to start.

You need the source code tree to work with. You can clone the repo either from

- official Apache Bigtop repo

Clone the repo

```
git clone https://gitbox.apache.org/repos/asf/bigtop.git
```

- or from Github mirror (see https://github.com/apache/bigtop)

How to generate patches by formatting patch files

We use git as our version control system. To streamline the process of giving proper credit to the contributors when committing patches, we encourage contributors to submit patches generated using `git format-patch`. This has many benefits:

- Committers can't forget to attribute proper credit to the contributor
- The contributors name and email address shows up in git log
- When viewing Bigtop's source code on github.com/apache/bigtop, the commits from the contributor are linked to their github.com account if it's linked to the same email address they used when generating the `git format-patch`

Long story short, it makes both the contributors' and committers' lives easier, so please generate your patches using `git format-patch`.

Here are some instructions on how to generate patches:

1. Ensure that you have all of your change as 1 commit which has the correct commit message - something like `BIGTOP-1031: README has outdated/ambiguous information`
2. `Run mvn apache-rat:check` to make sure that newly added files do not have any licensing issues. When in doubt refer to https://www.apache.org/licenses/
3. Then run a command like: git format-patch HEAD^..HEAD --stdout > BIGTOP-1031.patch
4. Upload the BIGTOP-1031.1.patch file to this JIRA

The naming of the patch file is up to you. The preferred way however is to just name the file after the JIRA ticket e.g. BIGTOP-1031.patch. In the latter case, If case you need to upload another version of the patch, you should keep the file name the same and JIRA will sort them according to date/time if multiple files have the same name. This feature is also useful to traceback the history of a patch and roll-back to an earlier version if needed.

How to generate patches by creating pull requests

1. Folk Bigtop from https://github.com/apache/bigtop
2. Develop your patch in your own branch
3. Ensure that you have all of your change as 1 commit which has the correct commit message - something like `BIGTOP-1031: README has outdated/ambiguous information`
4. Create a Pull Request against Bigtop master branch on Github with the PR title `BIGTOP-1031: README has outdated/ambiguous information`

REVIEWING PATCHES (reviewer guidelines)

The steady pace of the contributions relies on how effectively we can review, give feedback, and commit new patches. JIRA provides a nice workflow that allows to indicate which patches are ready for a review. Once you decided that your patch is good for others' comments change the JIRA status to Patch Available. That will be the indication to anyone that a comments and review are requested. For your convenience, here's the filter that shows all JIRAs in PA state

Contributors and Committers: Go over this checklist for your patches

(Hopefully more of this will be automated by https://issues.apache.org/jira/browse/BIGTOP-1249)

- Commit message should be part of the patch, see the recipe above using "git format-patch" for a way to do that.
- Commit message should be 1 line long. If you have multiple commits , squash them and rename them as in the next bullet:
- Commit message should be of the format "JIRA #. JIRA synopsis", for example "BIGTOP-1234. A patch that makes Bigtop awesome".
- Commits should be +1's by one committer (not the submitter) before committing.
- Commits which modify the way we **run** Bigtop should modify corresponding README files as well.
- Reviews from non-committers are highly encouraged as it helps you to learn more about the project and helps to catch more issues.
- Follow Apache Hadoop formatting guidelines:
    - No trailing whitespace on lines
    - Code must be formatted according to Sun's conventions, with one exception:
    - Indent two spaces per level, not four.
- CODE vs Docs
    - DOCs commits (README files, README.md files, and .txt files) can be commited by simply stating in the JIRA ("DOC")

For Committers: how to commit patches

While `git am` is typically used for applying patches generated using `git format-patch`, we recommend that committers use the `--signoff` flag when using `git am`. This way the commit, even though is attributed to the contributor, it shows the committer's name in the log message as "Signed-off-by: <Committer name>" which can be useful.
Consequently, to commit a patch, do the following:

```
git am --signoff < BIGTOP-1031.patch
```

Once the patch is committed, please update the JIRA ticket:

- make sure that **Affects Version/s:** and **Fix Version/s:** are correctly set against the release the issue is found in and is fixed against
- make sure that **Component/s:** field has correct values
- make sure the ticket is assigned to the person who contributed to code. If such person isn't yet marked as a contributor and the ticket can not be assigned to her/him - please ping anyone who has JIRA project admin. rights - pretty much any PMC member. Please remember that correct attribution is an important part of the contribution.
- make sure the status of the ticket is set to **Resolved**

For Committers: how to commit a patch in both master and release branch

This section is to talk about how Bigtop deal with the situation that a patch needs to be committed in both master and release branch. This often happened when a bug is discovered at the time we already created the release branch. To handle such scenario, Bigtop uses feature branch model. (To know more about git branching model, see http://nvie.com/posts/a-successful-git-branching-model/)

For instance, if patch BIGTOP-1886.patch is getting in both master and branch-1.0, then we should create a feature branch from branch-1.0 named BIGTOP-1886, and add BIGTOP-1886.patch on top of BIGTOP-1886 branch, then merge BIGTOP-1886 branch into master and branch-1.0, respectively. This will give same commit sha1 of patch BIGTOP-1886 in both branch-1.0 and master.

Here's a concrete example for what to do when we'd like to add fix 1 and fix 2 in both branch-1.0 and master. The example was contributed by Olaf Flebbe.

```
#inital setup of repo
git init
# Development step on master
echo step1 >file
git add file
git commit -m step1

# Now we branch the release candidate
git checkout -b branch-1.0
# and doing the release
echo release >changelog
git add changelog
git commit -m "release-1.0 branch finished"
git tag release-1.0.0
# development on master continues
git checkout master
echo development > development
git add development
git commit -m "development on master"

# Oops two fixes are needed
# create a working branch for these
git checkout branch-1.0
git checkout -b fixes-1.0
echo "fix 1 1.0" to  >fix10
echo "fix 2 1.0" to  >fix20
git add fix10
git commit -m "fix 1 on 1.0.0" fix10
git add fix20
git commit -m "fix 2 on 1.0.0" fix20
# merge fix branch in release branch0
git checkout branch-1.0
git merge -m "merge fixes-1.0" fixes-1.0
# new release
echo release 1.0.1 >> changelog
git add changelog
git commit -m "release-1.0.1"
git tag release-1.0.1
# Merge fixes on development, too
git checkout master
git merge -m "merge fixes-1.0" fixes-1.0
# continue development
echo devel >>development
git commit -a -m "development2"
```