Adding New Features to `Torch`
Last updated: August 24, 2022

1 Note to the reader

Keep in mind that everything listed here is based on my (Sean C. Lewis) personal workflow and can be augmented to fit whatever your preferences are. That said, following this workflow structure will help ensure that all contributions to the `Torch` project are kept organized and introduced to the master project in an organized matter.

2 Overview

In order to effectively maintain an up-to-date collaborative project, we will need to implement some sort of version control. Luckily, a `git` repository already exists for `Torch`: https://bitbucket.org/torch-sf/torch/src/master/

`Git` version control allows multiple users to view and use the most recent version of a project while also maintaining individual feature development and bug fixes. `Git` is not inherently intuitive and this document is intended to be a basic guide to the development and implementation of a feature in `Torch`.

A feature for `Torch` could be a minor update to a routine, a bugfix, or the inclusion of a new physics module. Any changes made to the `Torch` source code should be done using a process similar to the following.

3 Making a new `Torch` feature

3.1 Bring local repository up-to-date with `master`

```
git checkout master
git fetch origin
git reset --hard origin/master
```

This series of commands will switch your local repository over to the master branch and resets the local to match the latest version of the source code.

WARNING: If you have any previous changes that you have made locally, back them up in a repository outside of the Torch git controlled repo. All changes that diverge from the master branch will be erased by these commands.

`git checkout master` moves your HEAD to your local master branch (which reflects the remote master branch at the time of your last `git pull`

`git fetch origin` 'downloads' all updates from the remote master branch but does not make the changes locally.

`git reset --hard origin/master` implements all changes staged for the master branch and disregards any local changes you have made.

3.2 Create new local branch off `master` to work in

We want to now create a local copy of the master branch that we can augment without consequence to the production source code. Here, the code snippet shows the creation and switching to a branch called `new-feature`.

```
git checkout -b new-feature
```

The `-b` flag of this command ensures the branch `new-feature` is created locally if it does not exist yet.

NAME YOUR BRANCH: As per convention, name your branch according to whatever task you intend to undertake (all lowercase with dashes in between words). We will pretend for the rest of this document that your branch is called `new-feature`

You can check if you are in fact on this new branch by doing `git branch -a` which will show you all available branches and highlight the one you are on.

Once here, you can implement any changes you like. If you had to backup some changes before going through this process, you can copy the pieces back into the appropriate directories now. All changes you are making at this point are to the local branch called `new-feature` (or whatever you choose). Therefore, no one else can see your changes, and no one can access them or begin to integrate them into the source code.

To keep track of your changes to your branch: check the status of your branch (if you've made any changes since your last commit), make a change, stage the changes, commit the changes with an explanation message.

```
git status
git add [name-of-file]
git status
git commit -m "Explain your changes here (in quotes)"
```

You will want to perform these commits frequently, at the very least issue a commit every time a major change to your code is made. You can issue as many commits as you want prior to the next step of pushing your branch and associated changes to origin (remote).

3.3 Push `new-feature` branch to remote

This command makes your feature branch visible to all other collaborators on the project (both in content and all the commits you made during development).

```
git push -u origin new-feature
```

The `-u` flag adds the remote `origin/new-feature` as a remote tracking branch. In simple terms, it ties your local branch to a remote one.In the future if you can therefore simply use `git push` to push changes to the remote branch without the need for the extra fluff seen in the command here.

Now, your `new-feature` branch exists locally and remotely, everyone in the project group can see it, comment on it, work on it themselves, but it is still separate from the `master` branch. Next is to incorporate your changes into the main project.

3.4 Issue pull request

Go online to bitbucket (or wherever the git repository is managed) and you will see an option available for you to issue a pull request. Within this, you will denote which branch you are wanting to merge to where (in this example, `new-feature` —> `master`) and write up a description of your features added, and denote users to be referees of your changes. They (and you) will then have to check if any changes you are intending to make will conflict with the code in `master`.

3.5 Updating your local repository

If changes have been made to the master script and you need to incorporate them into your `new-feature` branch, you can issue the following commands given that the repository you are working on has been pushed to origin (follow the above steps):

```
git checkout new-feature
git merge origin/master
```

This is a non destructive action and both `origin/master` and `new-feature` will remain intact with all commit histories preserved.

If someone else has created their own branch `other-feature`, pushed it to origin, and you want to take a look at it (say they asked you to run some test problems using their branch), you first need to pull in that information from origin so you can see it locally. Then you can checkout `other-feature` and run your tests:

```
git checkout origin/master
git pull
git checkout other-feature
```

4 Other useful `git` commands

`git remote -v`
shows the address of the remote repository that your local repository is associated with (cloned from, etc,).

`git branch -a`
shows all branches associated from the project and highlights the one you are currently on locally.

`git rev-parse HEAD`
shows what commit hash you are checked out on locally.

`git log --graph`
shows the commit history with comments and authorship as well as branches, merges, and pull requests.

5 Terminology

version control - a system that records the changes made to a set of files so that you can revert back to a specific previous states of the file set. This document describes how to use the `git` version control system. Version control can be as simple as you making local copies of all your files, but `git` is much more suited for projects with many users.

local branch - a `git` branch that exists on your personal computer; one you can `git checkout` to see and make changes to. Local branches can be put on the remote level via `git push`.

remote branch - a branch that exists on the online version control system (origin). remote branches can be brought to the local level via `git pull`.

master branch - represents the working full state of the project. Note: local and remote versions of this branch exist. If you simply `git checkout master` you will NOT be up to date with the latest version of the project. You must issue a `git pull` while on the master branch as well.