Git Workflow for Heliotrope

This document outlines a workflow for creating Git branches for specific GitHub issues and bug fixes while collaborating with Data Curation Experts and other members of the Heliotrope development team. The local team generally follows a Clone and Branch method, with branches related to specific GitHub Issue(s). Once the work is ready to be merged with the project, you'll issue a Pull Request (PR) to the remote repository. You need to be set as a contributor to the repository in order to be assigned tickets or issue PRs. Non-local members can follow a Fork and Branch method, and ask for their work to be merged to the main repository.

Heliotrope on GitHub
Heliotrope Waffle Board

Locally

Create a `new-feature` branch from `master`. Try and base the name of the branch off the issue you're working on so it is clear to the rest of the team. (In this example I'm just generically calling it `new-feature`).

```
git pull origin master
git checkout -b new-feature master
```

Do work, add commits to the `new-feature` branch, edit, stage, commit, like this:

```
git status
git add directory/to/some-file
git commit -m "log message that will be recorded goes here"
```

NOTE: *You cannot switch branches without making commits to your* `new-feature` *branch first. Otherwise, your changes will be merged into the branch you are switching to. Nooooo! If you'd rather not commit, there are ways around this, like* `git stash`.

If changes have been made to `master` that are essential to your `new-feature` branch, you may want to pull those changes into your branch. If so, run

```
git checkout master
git pull
git checkout new-feature
git rebase master
```

Before you merge your changes to `master`, you may want to see a list of what will be merged.

Do the following while on your `new-feature` branch

```
git diff --stat --cached origin/master
```

When you're done with your work, push it to GitHub!

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

If you're ready to issue a Pull Request (PR), first make sure you've remembered to run your tests (`rake ci` from project root). Then head over to the Heliotrope repository on GitHub or open GitHub Desktop for Mac and issue a Pull Request (PR) for your branch to be merged with `master`. Be sure to include a complete description of your contributions and the related issue number (ie, #53). If it closes a specific issue number, be sure to include either "Fixes #53" or "Closes #53" or "Resolves #53" in the comment – GitHub will automatically close the issue upon successful merge of the branch to `master`.

You should never merge your own PR to `master`.

After you've received confirmation that `new-feature` has merged to `master`, delete the `new-feature` branch if the merger did not.

```
git branch -d new-feature
```

Updating Your Branch Mid-Pull Request

There may be something wrong with your code or questions about the way you've done something, resulting in requested changes to your branch. Instead of closing the PR, make the requested changes locally to your branch and then stage all modified files and commit:

```
git add directory/to/some-modified-file directory/another-file
git commit --amend
```

Enter a log message describing the amended fix. `--amend` will add any staged changes to the previous commit you made prior to your PR.

You'll then want to get your branch to the remote repository by force pushing your branch. Do:

```
git push origin new-feature -f
```

The `-f` flag will force push your branch. If this flag isn't included, Git will be annoying. Use it. But never force push to `master`.

Deleting Your Branch Without Issuing a Pull Request

You may also decide that you don't want to merge the work you did on the branch and would rather just delete it and start over. To do a hard delete without issuing a PR do:

```
git branch -D new-feature
```

Reverting to Previous Commits

In case you totally screw something up and commit something that breaks everything and need to jump backwards – you can revert back to a previous commit quite easily. First look at which version you'd like to move back to:

```
git log
```

Find the state (the first seven numbers of the commit) you want to revert to and do

```
git revert commit 123abc1
```

Hotfixes for Production During a Sprint

Oh no! A bug has been deployed to production and needs to be fixed outside of a regular sprint release cycle. Well, we only deploy tagged releases to production (not the `master` branch) so you'll want to first make sure you have the latest tagged release. Do

```
git fetch
```

Make a branch from the latest release. Let's say the latest release is 1.1.0. Do

```
git checkout 1.1.0
```

Then make your branch from that point:

```
git checkout -b hotfix/name-of-bug 1.1.0
```

Make your changes, test, and commit per usual. When you're ready, push your branch to the remote repository

```
git push origin hotfix/name-of-bug
```

Once tests pass, using the GitHub GUI, tag your `hotfix/name-of-bug` branch with a new release version. In this case, it would be tagged `1.1.1`

Now you're ready to deploy to staging, preview, and production (in that order).

```
nectar: be cap staging deploy
```

As the deploy scripts run, you'll be asked what branch or version you'll want to deploy – you'll want to enter the tagged release number, in this case `1.1.1`

Don't forget to merge your `hotfix/name-of-bug` branch into `master`. Using the GitHub GUI, create a PR like you would for a feature. This ensures that when master gets tagged and deployed on the normal release schedule, the hotfix is included.

On Tang/Sunny-D for Deployment to Nectar

Working from the U-M Library development servers, you'll want to clone and run commands a bit differently than if you were developing locally.

Clone the repo from your github account

```
git clone https://github.com/curationexperts/heliotrope.git
```

In the project/application directory, set the ruby version to 2.3.0

```
rbenv local 2.3.0
```

Install gems locally to the application

```
bundle install --path=.bundle
```

Follow directions from above for the Git Workflow.