Example Workflow for Teams

Courtesy Patrick Williams -
https://medium.com/bigcommerce-developer-blog/version-control-for-teams-a186bd74ba7e

Our project's repository will have two main branches: master and development. The master branch will contain working, production ready code. Pushing code to the master branch will essentially be a release of our project. The development branch will contain all our completed features that are not yet released.

As we discussed before, we will be creating a branch for each feature we are implementing within our project. We'll create this branch when we begin working on the feature, making sure we have the latest stable working code from the development branch as our starting point. Once we're done building the feature, we will merge our feature branch back to the development branch.

If you're following along with this example, you will only have a *Master* branch in your repository. To create the *Development* branch, simply run the following in the root path of your project to create and checkout the new branch.

```
git checkout -b development
```

Let's go through an example workflow for implementing *featureX*.

First, we'll want to make sure we're in the development branch. Running `git status` will give you information around which branch you’re currently in and any staged or unstaged file changes. You should see “On branch development nothing to commit”

Now we’ll use the same command to create our *featurex* branch

```
git checkout -b featurex
```

We are now ready to implement the feature. As a best practice, you will want to regularly commit your code to your branch with good messages that make it clear what work was done. For this example, create a new file, add it to the index, and commit the change.

```
cat > newfile
```

```
git add newfile
```

```
git commit -m “added blank newfile”
```

Once the feature is completed and tested, we will want to merge the feature branch into the development branch. Before we do this, we’ll want to run `git fetch` to ensure we have references to any changes made to the development branch in-between the creation of our feature branch and now. To merge, we’ll switch to the development branch and then use the merge command.

```
git fetch origin
```

```
git checkout development
```

```
git merge featurex
```

Since other changes may have been merged into the development branch since we checked out our branch, we'll need to run a command to ensure we have the latest changes from the current remote development branch. While git fetch updates the index, git pull will actually pull in the changes and update your files.

```
git pull origin development
```

Our changes made in the featurex branch have now been merged into the development branch. To push these to our remote setup at GitHub, we need to tell git to push the development branch to the remote repository "origin".

```
git push origin development
```

We can now safely delete our featurex branch since the work for that feature is complete and merged back into the development branch.

```
git branch -d featurex
```

Once we have enough features ready in our development branch, we will follow a similar path for creating a release and pushing to master. Since we are now working on the default branch (master) many of the commands we ran in previous steps require less arguments.

```
git checkout development
```

```
git fetch
```

```
git checkout master
```

```
git merge development
```

```
git pull
```

```
git push
```