**11 – Staying Synchronized with the Upstream**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

**Name:**

Top of FormThis topic focused on how to keep your local and origin repositories synchronized with the project’s upstream repo. In class you saw how changes contributed to the upstream via pull requests can be merged into the main branch by a project maintainer. When your changes, and those of your classmates, were merged it left your local and origin repos out of synch with the upstream. That is, the upstream main branch contains commits that your copies of the main branch do not.

The activities below will have you *synchronize with the upstream*. Synchronizing with the upstream ensures that your local and origin copies of the main branch contain all of the same commits as the upstream main branch. This is something that you will want to do regularly as it ensures that when you create a new feature branch your changes are built on the most up to date version of the project. Once you have synchronized your repositories you will repeat the process of fixing an issue and upstreaming your changes via a pull request. You will find an issue in the issue tracker, create a feature branch, fix the issue and make a pull request (PR) to the upstream for your changes.

**Getting Started Again:**

Like the last activity, you will be working within the *KitClient* for this activity. Revisit one of the prior activity sheets to find the docker run command that you used to start the KitClient.

Nothing is required here, but you must have the KitClient running to work with your local copy of the FarmData2 repository.

**Merged into Upstream:**

If the upstream maintainers decide that the changes in your PR are good for the project, they will merge them into the upstream main branch. That way they become a part of the main project, and everyone can benefit from your work. In class your instructor played the role of an upstream maintainer and demonstrated how the changes contained in pull requests are merged.

Figure 1 – Changes Merged into Upstream

Figure 1 illustrates the state after several commits have been merged into upstream main.

1. Answer the following questions using Figure 1.

a. Which commits (give the colors) were added to the upstream main since the contributor created the feature branch that is shown?

b. Which of the commit from part a did the managers merge from the developer’s pull request?

c. Briefly explain how the pink commit could have gotten into the upstream main.

2. GitHub will inform you when the active branch in your origin repo is out of synch with the upstream. Use your browser to visit your origin repo on GitHub and ensure that the main branch is selected.

a. How many commits behind the upstream main branch is your origin’s main branch?

b. Briefly explain how your origin’s main branch got behind the upstream main.

**Synch with Upstream:**

As changes from pull requests (yours and others) are merged into upstream main, the main branches in your local and origin repos will get *out of synch with the upstream*.

Figure 2 - Synch with Upstream

Now, recall that upstream main should be the starting point for all new work. So, before you can work on something new, you’ll want to make your main branches look like the upstream main (i.e. you will synchronize them with the upstream). This is a two-step process. You will *pull* the upstream main into your local repository and then you will *push* it to your origin. Figure 2 shows how pulling main from the upstream and then pushing it to your origin synchs the main branches.

*Setting Upstream Remotes:*

In order to pull changes from the upstream you will need to tell your local repository where the upstream is. In earlier figures, dotted orange lines were used to indicate that your local repository knew about the origin and that the origin knew about the upstream. But there was no orange dotted line from your local repository to the upstream. That is because, by default, your local repository isn’t aware of the upstream.

3. In order to pull from the upstream you will first need to make your local repository aware of the upstream. This is done by *setting a remote* for it. This question will walk you through that process.

a. The git remote -v command lists the names and URLs of all of the remote repositories (or just *remotes* for short) that your local repo knows about. Run this command in the directory that contains your local repository. Give the output that it generates.

b. Which of the orange dotted lines in Figure 2 is represented by the two lines of output in part a?

c. To create a new remote that tells your local repo about the upstream you will need the URL of the upstream repo. Use your browser to navigate to the upstream FarmData2 repo that you are using for this course and give its URL here. Hint: You can find the upstream using the link from your origin. Once you find the upstream, you can get the URL by using the “Code” button in the upstream repo, it ends in .git.

d. The git remote command can also be used to add information about a remote repo to your local repo. The format for this command is:

git remote add <remote repo name> <remote repo URL>

Give a command that will create a remote named upstream that points to the FarmData2 upstream repository that you are using for this activity.

e. What new lines appear in output of git remote -v after you use your command from part d?

e. To which orange dotted line in Figure 2 do the new lines of output in part e correspond?

Note that you will only need to set the upstream remote once for a repository. Once the upstream remote is set you will be able to pull from the upstream repo as often as is necessary.

Pulling from Upstream:

4. The next step is to pull the changes from the main branch from the upstream repo to your local repo. This question walks you through that process.

a. To pull the changes from the upstream main branch into your local repo you first need to ensure that main is the active branch in your local repo. Make sure that main is the active branch in your local repo. Give the command(s) that you used and their output.

b. The command git pull --ff-only <remote repo name> <branch> will pull any new commits from the specified branch in the remote repo and add them onto the end of the active branch of your local repo. Give a command that will pull and add the commits from the main branch of the upstream repo to your main branch.

c. Give the output that was produced from your command in part b.

d. Examine the output you gave in part c. Did your command succeed or fail? How can you tell? If your command failed, return to part b and try again.

Push to Origin:

5. Your local main branch is now in synch with the upstream main branch. What’s left is to also synch the main branch in your origin repo.

a. Recall that you previously used the git push <remote repo name> <branch> command to push your feature branch from your local repo to your origin repo. Give a command that will push the main branch of your local repo to your remote origin repo.

b. What output was generated by your command in part a. You should see a lot of information in your output about *deltas* and *objects*. Those are terms related to the inner workings of git that we will not worry about. If you don’t see this information about deltas and objects but see an error message instead, revisit part a and try again.

c. The last two lines of the output in part b should contain some recognizable information. What do you see in these lines? What do you think it means?

As with most things related to Git and GitHub, there are a number of different ways to synchronize. For example, you can also fetch the changes from the upstream main to your origin and then pull them from there to your local repository. If you are curious, you can check out this GitHub link:

* <https://docs.github.com/en/github/collaborating-with-pull-requests/working-with-forks/syncing-a-fork>

**Deleting a Feature Branch:**

Once your changes have been merged into the upstream main, and you have synchronized there is no need for you to retain your feature branch. There is no harm in keeping it. But most developers will delete them to avoid having their repos become cluttered with old feature branches.

Figure 3 - Deleting Feature Branches

Figure 3 shows the state that will result after deleting the feature branch from your local and origin repositories.

6. The first step is to delete the feature branch from your local repository. This activity will walk you through that process.

a. It is not possible to delete the active branch. So, if you have not already, switch from your feature branch to the main branch. Then give a command that lists all of the branches in your local repository and show its output.

b. The command git branch -D <branch> will delete a branch from your local repository. Give a command that deletes your feature branch and show the output that it generates.

c. Use your command from part a again. What output does it generate now? What changed?

7. In the previous question you deleted your feature branch from your local repository.

a. Does this feature branch still exist in your origin repo on GitHub? How did you check?

b. You can delete your feature branch from your origin repo using the git CLI. The command git push -d <remote repo name> <branch> will delete the specified branch from the specified remote repository. Give a git command that will delete your feature branch from your origin repo and show the output that it produces.

c. Use your browser as you did in part a to confirm that your feature branch is deleted from your origin repo on GitHub. If it has not been deleted, revisit part b and try again.

Nothing is required here, however you should ensure that your feature branch has been deleted from your origin on GitHub.

**Command Summary:**

It will take a while for this process and all of the commands to become a natural part of the way you work. Until that happens it is often useful to have a short concise cheat sheet of the command for accomplishing each task.

8. Complete the table below by filling in the right-hand column with the commands that accomplish the task listed on the left. Use the < > notation appropriately to indicate parameters that need to customized for each use. Note that the tasks listed are in approximately the same order as they appear in this activity.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task to Complete** | **Git Commands** |  |
|  | List your remote repos |  |  |
|  | Add a new remote repo |  |  |
|  | Synch a branch of your local repo with the upstream. |  |  |
|  | Synch a branch of your origin repo with your local repo. |  |  |
|  | Delete a feature branch from your local repo. |  |  |
|  | Delete a feature branch from your origin. |  |  |
|  |  |  |  |

**Let’s Do it All Again:**

You’ve learned a lot about Git and GitHub. But, to become proficient with these tools takes a lot of repetition and practice. This section asks you to go through the process of finding an issue, fixing it and creating a pull request again. This is good practice but what you will be asked to do has also been carefully planned to set us up for the next activities too!

9. Visit the issue tracker in the upstream FarmData2 repository being used for these activites.

a. Use the *label dropdown* in the issue tracker to filter for issues labeled with the “Round2” tag. How many issues are tagged with this label?

b. Clearly there are not enough “Round2” issues for everyone to have their own as was the case with the “Round1” issues. So, this time there will be multiple people working on the same issue. Pick one of the “Round2” issues to work on and make a comment on it to indicate your interest. Note it is not necessary that the issue be assigned to you this time. Try to spread out across the issues so that multiple people have commented that they are working on each one. Give the Issue # and title that you chose.

10. Now you will go through the process of fixing the issue and making a pull request. You will use the same process that you used for your “Round1” issue earlier. That process had a number of steps. They are listed below in a jumbled-up order. Reorder these steps into an order that will complete the task.

Commit changes to the feature branch

Switch to the feature branch

Make a pull request to the upstream

Add changes to the stage

Push the feature branch to your origin

Create a new feature branch

Edit the local files

11. Use the process that you outlined in question #10 to fix your “Round2” issue and create a pull request to the upstream for your changes. As you do, give the exact Git commands that you use for the steps indicated below. The command reference that you created in the previous activity may also come in handy here. Also be sure to:

* Use a descriptive name for your branch.
* Use a meaningful message when committing your changes to the feature branch.
* Provide a useful title and description when creating your pull request.
* Include a “Closes” or “Fixes” line in your pull request description so that the associated issue number will be closed automatically if your pull request is merged.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task** | **Git Command** |  |
|  | Create feature branch |  |  |
|  | Switch to feature branch |  |  |
|  | Add changes to stage |  |  |
|  | Commit changes |  |  |
|  | Push feature branch |  |  |
|  |  |  |  |

12. Give the following information about your pull request. Hint: You can revisit the prior activity to refresh your memory on the terms Base Repository, Head Repository, Base Branch and Compare Branch.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | |  |
|  | The number of your PR | |  |  |
|  | Full URL of your PR | |  |  |
|  | Base Repository URL | |  |  |
|  | Head Repository URL | |  |  |
|  | Base Branch | |  |  |
|  | Compare Branch | |  |  |
|  | The line number of the README.md file that you changed. Hint: use the diff you saw in the prior activity. | |  |  |
|  |  |  | |  |

13. Because there were a limited number of “Round2” issues, other students will have been simultaneously making changes to fix the same issue that you did. Thus, there will be multiple pull requests for each change. Describe in a few sentences the types of problems might this eventually create.

**Optional:** To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.

**Acknowledgements:**

Some materials, questions and resources have been adapted from activities posted on foss2serve.org:

* <http://foss2serve.org/index.php/Git:_Git_Intro_Activity>
* <http://foss2serve.org/index.php/Git:_GitHub_Issues_and_Pull_Requests>
* <http://foss2serve.org/index.php/Git:_GitHub_Workflow_Activity>
* <http://foss2serve.org/index.php/Intro_to_GitHub_(Activity)>
* <http://foss2serve.org/index.php/Version_Control_(Activity)>
* <http://foss2serve.org/index.php/Work_Locally_with_Git_from_the_Command_Line_(Activity)>