**3 – Staying Synchronized with the Upstream**

**Activities**

**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 are 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. 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 upstreaming a change. You will again find an issue, fix it and make a pull request (PR) to the upstream.

Remember to work within the KitClient when working locally.

**Merged into Upstream:**

If the upstream maintainers decide that your changes 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 were added to the upstream main since the contributor created the feature branch that is shown?

b. Which of the commits from part a were merged from the developer’s feature branch?

c. Explain where the pink commit to the upstream main may have come from?

2. GitHub will inform you when the active branch in your origin repo is out of synch with the upstream. Visit your origin repo using your browser 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*. Recall that the 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 from the upstream and pushing to your origin synchs the main branches.

Figure 2 - Synch with Upstream

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

git remote add <name> <Repository 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. Hint: You can find the URL by using the “Code” button in the upstream repo, it ends in .git.

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

e. To which orange dotted line in Figure 2 do the new lines of output in part d 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. Ensure that main the active branch in your local repo. What command(s) did you use? What output was produced?

b. The command git pull --ff-only <remote repo name> <branch> command 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. What output does your command produce?

c. Did your command from part b succeed or fail? How can you tell?

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> <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. What output does your command generate?

b. There should be 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. The last two lines of the output should however have some recognizable information. What do these lines of output tell you? (Don’t just copy them here, explain what you think they mean.)

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 shows the state that will result after deleting the feature branch.

Figure 3 - Deleting Feature Branches

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

a. Give a command that lists all of the branches in your local repository. Ensure that you are on the main branch and use your command. What output does it generate?

b. The command git branch -D <branch> will delete a branch from your local repository. Give a command that deletes your feature branch. What output does it generate when you use it?

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. Did this also delete that branch from 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. What output is generated when you use the command?

c. Confirm that your feature branch is deleted from GitHub. If it is not, try again.

**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 topic!

9. Visit the issue tracker in the upstream repository.

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 of these issues for everyone to have their own like 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. Try to spread out across the issues so that there are multiple comments 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 will be closed automatically if the pull request is merged.

a. Create a new feature branch.

b. Add changes to the stage

c. Commit changes to the feature branch

d. Push the feature branch to your origin

12. Give the following information about your pull request.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | |  |
|  | Base Repository URL | |  |  |
|  | Head Repository URL | |  |  |
|  | Base branch | |  |  |
|  | Compare branch | |  |  |
|  | The line number of the README file that you changed. Hint: use the diff | |  |  |
|  | The full URL to your Pull Request | |  |  |
|  |  |  | |  |

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. What types of problems might this 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)>