**09 – FOSS Community and Collaboration**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

**Name:**

Top of FormThis topic has focused on FOSS communities. It looked at what they are, how they are organized, the principles on which they operate and the roles that the members take on. We also began to look at some of the tools and processes that these communities use to support those principles and coordinate their work. In particular we saw how git and GitHub work together to allow FOSS communities to share their work and to collaborate. We saw how forks and clones make it possible for a diverse and distributed group of contributors to work asynchronously and to contribute useful changes back to the upstream. In this and the next several activities you will use a copy of the FarmData2 project repository to gain hands-on experience using use git and GitHub. These activities will strengthen your understanding of git, GitHub and FOSS communities.

**Pre-Survey:**

0. The materials used here for learning about git and GitHub are a part of a research study on teaching open source in the computer science curriculum. As a part of that study please consider completing the following pre-survey. You will then complete a corresponding post-survey to help the researchers understand the effectiveness of these materials.

<https://drexel.qualtrics.com/jfe/form/SV_cHFOcgCVskr19L8>

Please complete the above pre-survey.

**FOSS Communities:**

1. In class we discussed some of the key principles that guide FOSS communities. These were *shared values*, *collaboration*, *transparency*, *inclusivity*, *meritocracy* and *release early & often*. Consider each of the following statements about the operation of a FOSS community and label it with the principle (or principles if multiple apply) with which it most closely aligns.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Statement** | **FOSS Community Principle(s)** |  |
|  | By building on each other’s work the community can solve problems that no one could solve alone. |  |  |
|  | Decisions and the rationale for them are available to the community. |  |  |
|  | The mission and goals of the community are more important than individual agendas. |  |  |
|  | The best ideas should win, regardless of where they come from. |  |  |
|  | Incorporating new changes and features quickly generates feedback and leads to rapid improvement. |  |  |
|  | Community members enhance and extend what others contribute in unanticipated ways. |  |  |
|  | Decision makers continually seek diverse perspectives. |  |  |
|  | All community members have access to the information necessary to do their best work. |  |  |
|  |  |  |  |

2. In class we also discussed some of the roles that tend to exist in FOSS communities. These included *Users*, *Requestors*, *Contributors*, *Maintainers* and *Leaders*. Consider each of the actions described below. For each action, label it with the role of the individual (or individuals if multiple apply) that is *most likely* to be responsible for the action.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Action** | **Role** |  |
|  | Choosing the license under which the project will be released. |  |  |
|  | Using the software in a new, unanticipated or creative way. |  |  |
|  | Asking that a useful new feature be added to the software. |  |  |
|  | Discovering a bug in the software. |  |  |
|  | Providing a code patch that fixes a bug in the software. |  |  |
|  | Submitting an improved set of installation instructions. |  |  |
|  | Documenting a bug in the issue tracker so others can fix it. |  |  |
|  | Defining the goals for the next year of work on the project. |  |  |
|  | Incorporating a contributed bug fix into the main branch. |  |  |
|  | Redesigning a critical software module in the system. |  |  |
|  |  |  |  |

**Creating a GitHub Account:**

You will be using git and GitHub for the remainder of these activities. In order to use GitHub it is necessary that you have an account.

3. Give the full URL to your GitHub profile (e.g. github.com/MyUserName) below. If you do not have a GitHub account, or would like to have a separate account just for this class you can create a GitHub account here: <https://github.com/signup>.

**Forking FarmData2:**

Recall that the first step in working on a FOSS project in GitHub is to make a *fork* of the *upstream* repository. As shown in Figure 1, creating a fork makes a full copy of the main project repository (i.e. the upstream) into your own GitHub space.

Figure - Forking the Upstream

For the purposes of this activity (and then next few) you will not be interacting with the main FarmData2 repository. Instead, you will be interacting with a copy of FarmData2 that was created just for these activities. We will treat this copy as if it is the “Main Project Repo” or the upstream. This will allow you to practice interacting with an upstream for learning purposes without affecting the main FarmData2 repository, which is under active development.

The URL for the copy of FarmData2 that we will use as the upstream for these activities is:

[**https://github.com/dickinson-comp190/GitKit-FarmData2**](https://github.com/dickinson-comp190/GitKit-FarmData2)

4. Using your web browser of choice go to the upstream repository given just above.

a. Find the “Fork” button ( Text

Description automatically generated ) on the upstream repository page.

b. Click the “Fork” button to create a new fork of the upstream repository. Make sure that you create the fork into the GitHub account that you listed above in question #3.

c. Give the full URL of your fork. Recall that creating a fork copies the repository into your own GitHub space. Thus, the URL for your fork will include your GitHub username. If you do not see your GitHub username in the URL, you have not found your fork. Visit your own GitHub space and see if you can find the fork there. If not, go back to part a and try again.

5. In Figure 1 there is a dotted line from the fork back to the upstream showing that the fork knows about its upstream repository.

a. Visit the GitHub page for your fork. There is some text and a link on that page that indicates the upstream repo from which it was forked. Give a screenshot of that text and link here.

b. Click the link in the text you found for part a to go to the upstream repo. Use the back and forward buttons of your browser to compare the list of files contained in the upstream repo to the list of files contained in your fork. Is your fork an exact copy of the upstream repo? If you notice any differences describe them here.

c. In class we discussed a *workflow* for making contributions to FOSS projects. For that workflow to operate it is essential that your fork know the upstream repository from which it was forked. Which of the four operations discussed in class would require this information?

**Running the KitClient Linux Environment:**

Because your individual computers can vary greatly, you will be working within the *KitClient*. The KitClient is a Linux environment running inside of a Docker container. The KitClient that has been created specifically for these activities and includes a few extra features that are designed to help with these activities. Using the KitClient also ensures that everyone is working in the same environment, simplifying the instructions, and reducing the chances that things will go wrong.

6. What is the KitClient and why is it being used?

7. Running and interacting with the KitClient requires that you install (or have already installed) some dependencies. If you already have Docker and TigerVNC Viewer installed, skip to question #8. Otherwise, complete the following steps:

a. Download, install and run *Docker Desktop*, which will be used to run the KitClient:

* + <https://www.docker.com/>

b. Install the *TigerVNC Viewer*, which you will use to interact with the KitClient:

* + Windows: <https://sourceforge.net/projects/tigervnc/files/stable/1.12.0/vncviewer64-1.12.0.exe/download>
  + Mac: <https://sourceforge.net/projects/tigervnc/files/stable/1.12.0/TigerVNC-1.12.0.dmg/download>

8. Now let's create and run the KitClient and connect to it.

a. Open a Terminal Window or Command Prompt on your computer and then copy and paste the command below to run the KitClient:

docker create --name KitClient --mount source=gitkitvol,target=/home/student -p 6901:6901 -p 5901:5901 registry.gitlab.com/hfossedu/kits/kitclient

It may take a little while for the KitClient to download to your machine. Fortunately, you will only have to use this command once.

When the download completes, the final line of output from the above command should be a long string of letters and numbers. Copy and paste that string here. If you do not see that string, check the output for error messages and try the command above again.

b. Now that you have the KitClient image you can start it running using the following command:

docker start KitClient

While this command doesn’t appear to do much, it has in fact started a server on your machine that contains the KitClient. If this worked successfully, the command will simply output KitClient.

There is nothing required here. But you need to be sure that the docker start command has outputted KitClient to indicated that the KitClient is running. If not, revisit #7 and #8 to ensure that you have everything installed correctly.

c. With the KitClient running you can connect to it using the TigerVNC viewer.

* + Launch the TigerVNC viewer.
  + Put the address: localhost:5901 into the “VNC server” text field.
  + Click the “Connect” button.

When you click connect a window should open with the KitClient’s Linux environment running inside. Take a screenshot of that desktop and paste it here.

Note: If you have been unable to install the Tiger VNC viewer you can also access the KitClient using a web browser by visiting the URL: http://localhost:6901

If you access the KitClient via your browser:

* + Open the “noVNC” menu (the little tab on the left) and then use the settings (the the gear) to set the “Scaling Mode” to “Remote Resizing.” This will resize the KitClient desktop when you resize your browser window.
  + Copy and pasting between your OS and the KitClient requires that you use the clipboard in the “noVNC” menu. This is a little inconvenient, but it is functional.

9. At some point you will complete this assignment or want to take a break and do other things. When you do so, you should stop the KitClient and then restart it when you are ready to work again.

a. Stop the KitClient by opening a Terminal Window or a command prompt on your computer (not in the KitClient) use the command:

docker stop KitClient

Take a screenshot of your command and its output and paste it here.

b. Use the steps in #8b and #8c to restart the KitClient and reconnect to it with TigerVNC (or noVNC in your browser). You should notice that it starts much more quickly this time.

Nothing is required here. But be sure you have the KitClient running and open in a VNC Viewer window or in a browser. **The remainder of this activity (and the next several) will assume that you are working within the KitClient.**

**Cloning Your FarmData2 Fork:**

In order to work with the files in your forked repository you will need to create a *clone* of it onto your local machine as shown in Figure 2.

Figure - Cloning your Origin

9. To make a clone of your fork you will first need a *Git client*. There are lots of different Git clients and most Integrated Development Environments (IDEs) will provide a graphical interface to Git. For these exercises however, you will use the command line interface (CLI) to Git. This is the most general way to interact with Git and will provide a foundation for using any of the graphical interfaces that you might encounter.

Open a terminal window (inside the KitClient) and enter the command git. Use the output that you see to determine how to display the version of git that is installed.

Paste a screenshot of the command you used to display the git version and the output that it generated here.

10. Recall that Git repositories maintain a complete history of all changes. To do this, git needs to know a little bit of information about you. This question will have you do a little configuration to provide git with the information that it needs.

Enter the following commands one by one on the command line replacing the <…> with the appropriate values:

git config --global user.name <your GitHub username>

git config --global user.email <your e-mail>

Now run the following command:

git config --global --list

Give a screenshot of the commands you used and their output here. Note: You should see the information that you entered in the above commands. If not, try those commands again.

11. Now with git configured you can use it to create a local clone as was illustrated in Figure 2.

a. On the GitHub page for your fork, find the “Code” ( Graphical user interface

Description automatically generated with low confidence ) button. When you click this button, you will be shown a URL that can be used to clone the repository. That URL should include your GitHub username. If it does not, then you are not on the page for your fork (Go to your GitHub profile and find your fork). Also that URL should end with .git. If it does not, then you have not copied the correct URL (Check under the Code button again). Copy and paste the URL you found here.

b. Open a Terminal (in the KitClient) and use the URL from part a in the following command:

git clone <URL>

Give a screenshot of the command you used and its output here.

c. In the output above from part b, there should be a line that starts “Cloning into”. This line tells you the name of the directory into which the repository has been cloned. If you do not see this line, check the output from part b for errors and try again.

What directory was your repository cloned into?

d. Examine the remaining output from your command in part a. What is the “Kit-tty”?

Be sure to keep your eyes open for messages from the Kit-tty, they will help you to stay on track with these activities and may save you a lot of time by preventing you from going down an incorrect path.

e. Examine the files that are in the directory you found in part c and compare them to the files that are in your fork on GitHub. Remember that you can use ls -a to see all files including hidden files. Is your clone an exact copy of your fork? If you notice any differences describe them here.

12. In Figure 2, there is a dotted line from your clone to your fork indicating that your clone knows about the remote repository from which it was cloned. The Git CLI will allow us to see and confirm this connection.

a. Using a Terminal, ensure that your working directory is the directory containing your cloned repository and use the following command:

git remote -v

This command lists all of the remote repositories that your clone knows about. In your case there should be two lines of output (one for “push” and one for “fetch”).

Give a screenshot of your command and its output here.

b. In class we saw that the name *origin* is used to refer to your fork on GitHub. You should also see this name in the output from part a. This indicates that your clone knows the URL of your origin. How is the information about the *origin* remote represented in Figure 2?

c. In class we discussed a *workflow* for making contributions to FOSS projects. For that workflow to operate it is essential that your clone (i.e. local copy) know about the origin repository from which it was cloned. Which of the four operations discussed in class would require this information?

**The Issue Tracker:**

You have now forked and cloned the our FarmData2 repository and are in the state shown in Figure 3. From here you are now ready to do some work.

Figure - Ready to Work

As described in class, FOSS projects often use an *issue tracker* to maintain a list of bugs to be fixed, features to be added and other tasks to be completed. Thus, the issue tracker is a good place to go in order to find something to work on in a FOSS project.

13. The issue tracker for the FarmData2 repository that you forked contains a list of issues that we will use for practice. This question explores that issue tracker.

a. Every issue has a *number* and a *title*. What is the title of issue #10?

b. Each issue can have *tags* that can be used to sort and categorize issues. What tags are associated with issue #8?

c. To which file in the repo does the typo described in issue #11 apply?

d. In what different orders can you ask the GitHub issue tracker to sort the issues?

e. Why might it be useful to sort issues by oldest or least commented or least recently updated?

14. Most projects will have a process by which contributors can *claim an issue* to work on or have an issue that they want to work on *assigned* to them.

a. Examine the issue tracker for the FarmData2 upstream that we are using for this activity (see the URL just above question #4).

Which issue has been assigned to someone? To whom has it been assigned?

b. In the version of FarmData2 that you are working on you can claim an issue in the issue tracker by commenting on the issue with the **exact** **message**:

I would like to work on this please!

The first person who comments on an issue with this message will receive a response from the project maintainers that the issue has been assigned to them and the issue tracker will update to show this.

Visit the upstream repository **and comment on an issue that is tagged Round1 to claim it**.

Reload the issue tracker to double check to be sure you receive a response and have been assigned the issue before going on. If you are not assigned the issue you may have mistyped the message above or someone else may have slipped a comment in just ahead of you and was assigned the issue. If this happens, just try again.

What is the title and number of the issue that has been assigned to you?

c. Some projects intentionally do not allow issues to be claimed or assigned to individuals. What do you think some advantages and disadvantages of claiming or assigning issues might be?

15. Do not attempt to fix your issue. That is, **please do not make any changes to the files in your local repo at this time.** We will be learning about the proper way to work with a repo in the next activity. If you accidentally made changes, please delete your clone and recreate it (see #11 b). Type “I Understand” in the following box to confirm that your local repo has not been changed.

16. When you forked the upstream, GitHub made a copy of the repository into your account. Did it also make a copy of the issue tracker and its issues into your GitHub space? What reasons can you think of for why the designers of GitHub decided that a fork should not also have a copy of the issue tracker?

If you are curious, you can explore the issue tracker for the main FarmData2 repository here:

* <https://github.com/DickinsonCollege/FarmData2/issues>

**Review & Reflection:**

17. Give a short description, in your own words, of what each of the following Git CLI commands does. Your description should do more than simply restate the command. I.e. “Clones the repository” is not a sufficient description for the git clone command.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Command** | **Description** |  |
|  | git config |  |  |
|  | git remote -v |  |  |
|  | git clone |  |  |
|  |  |  |  |

**Extra Practice - Forking and Cloning Another Project:**

18. There are literally millions of repositories on GitHub that you can fork and clone and work on. For a little extra practice with forking and cloning, choose one of the following projects that sounds interesting to you and then complete the questions below.

|  |  |  |
| --- | --- | --- |
| **Project** | **Community Mission** | **Repository URL** |
| OpenMRS | Aims to “Improve healthcare delivery in resource-constrained environments by coordinating a global community that creates and sustains a robust, scalable, user-driven and open-source medical record platform.” | https://github.com/openmrs/openmrs-distro-referenceapplication |
| Fineract | Aims to provide “a reliable, robust, and affordable core banking solution for financial institutions offering services to the world’s 3 billion underbanked and unbanked.” | https://github.com/apache/fineract |
| Sahana Eden | Aims to provide an “Open Source Humanitarian Platform which can be used to provide solutions for Disaster Management, Development, and Environmental Management sectors.” | https://github.com/sahana/eden |
| Oppia | Aims to provide “A free, online learning platform to make quality education accessible for all.” | https://github.com/oppia/oppia |

a. Fork the repository for the project into your GitHub space. Give the URL of your fork.

b. Clone your fork into your home directory in the KitClient.

Give a screenshot of the commands you used and their output here.

c. Use the git command that displays the URL’s of the remote repository from which your local repo was cloned.

Give a screenshot of the commands you used and their output here.

**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:_Cloning>
* <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)>

Some materials, questions and resources have been adapted from opensource.com under the :

* <https://opensource.com/open-source-way> (question #1)
* <https://www.theopensourceway.org/the_open_source_way-guidebook-2.0.html>