Git Pre-Work Exercises

In these exercises we will introduce topics related to *version control systems*, specifically we will be using the Git version control system. These exercises will introduce some of the basic features and commands that are associated in Git. During the cohort we will review these commands but it is assumed that you have completed these exercises and understand these commands at a high level as we will expand on this during the cohort but spend more time with advanced topics such as branching, merging and rebasing.

Some Basic Terms

Version Control System - standalone application that tracks and stores the revisions of documents being stored within a repository

Git - open source version control system that is both very popular and efficient in its implementation. This system focuses on developers working in a distributed environment where offline use may be required.

GitHub - an online git host provider that allows for the creation and use of git repositories that can be shared amongst developers via an internet connection.

How this relates to you:
Your code will be checked into a repository and you will be able to save revisions of that code as you development various applications. In an enterprise or team scenario, this allows multiple developers to work on the same code at the same time creating a history of the changes that have been made to the code. In the event that something went wrong, a previous revision of the code can be retrieved.

Git Commands Covered

This is not all encompassing list of the git commands but rather a sample set of commands that we will see in the Pre-Work. A few more commands will be introduced during the cohort but these are the basic ones you are expected to have knowledge of when you begin your cohort.

clone - copies an existing repository to the local file system
add - adds changes to the repository putting them in a staged status, readying for commit
commit - creates a snapshot of changes to the repository
pull - retrieves changes from remote repository and attempts to incorporate those changes in current branch of local repository
push - saves local changes to remote repository
status - returns the current status of the repository
log - returns a report to review the change log and history of repository

Exercise 1: Getting Setup

Now that we know some of the terms and have an idea of the commands we will be working with, let's talk about setup of git and how to get started.

Installing Git

Git can be installed on Windows, Linux or Mac. Instructions for each installation can be found on the git website located at git-scm.com. The key to a successful installation is installing the Git Bash command line tools. All exercises and labs will use the git bash command line to accomplish our goals.

Setting Git up

Once you have installed Git you can set your email address and username so that this is used with each of the repositories you create locally. This will be saved in the git configuration at the machine level and used for all repositories on that machine.

Steps

1. Open a command prompt or terminal window. This varies by operating system and is assumed you know how to do this.

Note for Windows users: If you installed the explorer integration you can also right click and select git bash to open a command window with the current directory set as the current context.

2. Run the command to set your username

```
git config –global user.name "<USERNAME>"
```

3. Run the command to set you email address

```
git config –global user.email "<EMAIL>"
```

At this point you are ready to use git on your local machine.

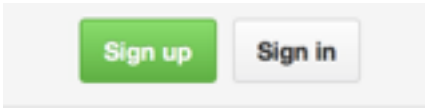
Create a GitHub Account

In order to simulate a typical development environment where you are able to share your code with others we will have you create and setup a github account. Github is an ideal site to share code with other developers and a very popular site for open source projects. Repositories residing on github will be used throughout the cohort whether they are your personal repositories or once shared for cohort purposes. Follows these steps to create the account.

Steps

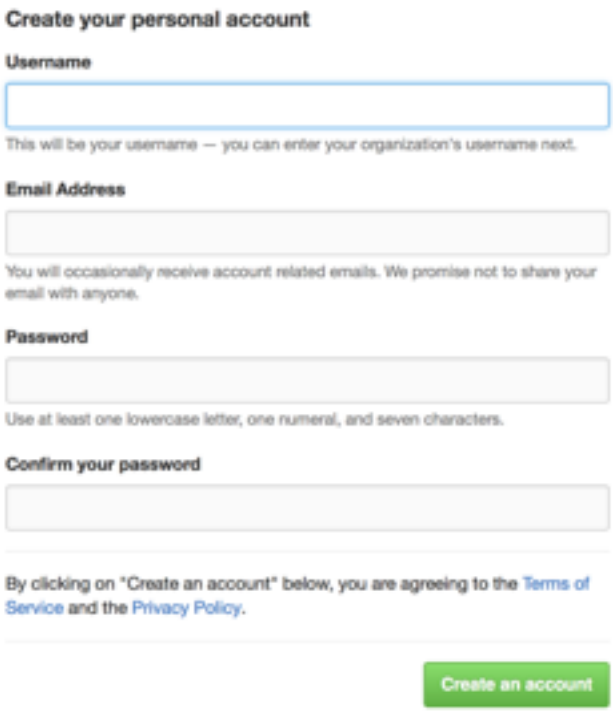
1. Open a Browser and navigate to http://github.com

2. Assuming you don't have an account, click the sign up button in the top right hand corner



Note: if you have an account you can use your existing account and skip this section

3. Enter a preferred username, your email address and password you wish to associate with your account and click" Create an account"



4. Next you can select a plan. for the purposes of the cohort, you can select the free plan. We will not require private repositories and hence no need to pay for an account at this time.

5. Finalize your account and welcome to github!

Additional Resources

The Git Website: http://git-scm.com

Git Documentation: http://git-scm.com/doc

Git Pro Online Book: http://git-scm.com/book/en/v2

Exercise 2: Using Git

So, you're setup and ready to go. Where do you start?… In this section we will walk through a few of the basic commands and get you setup for the rest of the Pre-Work. This section can also be used for reference later as several of the commands will be demonstrated without any action required by you. Your action with them is to review them as you complete other pre-work assignments as you will be asked to submit screenshots and certificates to the repository created within.

Creating a Repository for Pre-Work

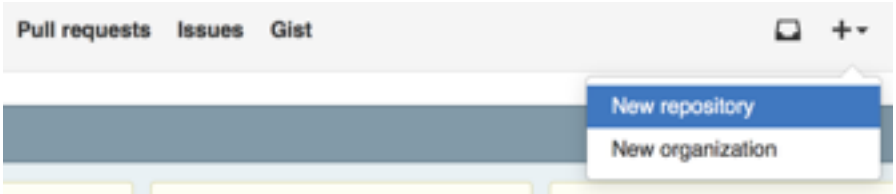
In order to share the results of your Pre-Work with your instructor, we have decided to have you save everything to a git repository. This will both show you that not just code files can go into a repository and also give you some valuable practice with the tool and commands. This repository will use your github account and you will need to share the URL of this repository with you instructor via the Moodle system you have already logged into. (if you have not seen Moodle, please review pre-work materials.)

Steps

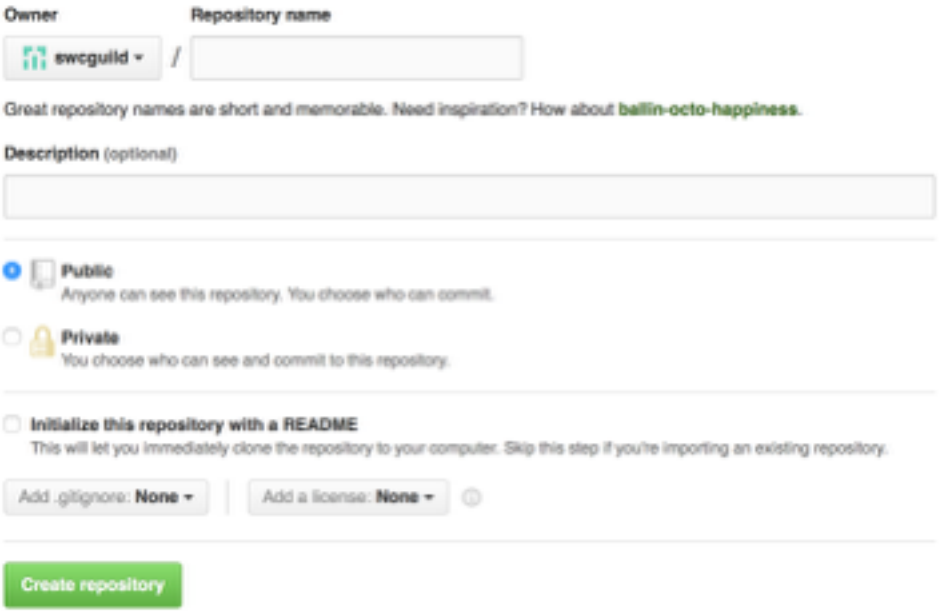
1. Open a browser and navigate to https://github.com

2. Login to github using the account you created in 1st exercise

3. Upon logging in click the plus "+" sign in the top right hand side of the screen and select "New Repository" from the menu



4. On the next screen set the "Repository name" = "swcguild", provide a "Description" and make sure you have it set for a "Public" repository



5. Click “Create repository”

6. Once you have created your repository you will be presented with the repository page. From here you can get to the URL of your repository. This page will remain in this format until you have added content to the repository.

Note: blank spaces in the screen shot would be filled with your user name

Cloning a GitHub Repository

Anytime you want to work with a github repository locally you will need to clone your repository.

Steps

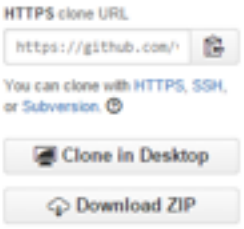
1. Open a browser and navigate to https://github.com

2. Navigate to your repository that you wish to clone (ex. “swcguild”)

3. Copy the URL of the repository

The URL will either be available on the main page if no content exists in the repository yet. This will be the case with any new repository just created.

Or it will be located in a box on the right hand side of the page part way down the page.



4. Open a command prompt \ terminal window

5. Type the git clone command for the repository

```
git clone https://github.com/<user>/swcguild.git
```

The URL starting with “https” can be pasted from step 3.

Also note the command window will either need to be on the directory where you wish to clone the repository to or you will have to use commands to navigate to such a directory before executing this command.

Ex. Navigate the command window to C:_repos and then execute git clone to place the repository within this directory.

Checking the Status of a Repository

As you work with a repository more you may find that you lose track of which files you have added (or staged) to the repository, what files have not yet been committed and which files you haven’t even added to the repository and are still untracked. To alleviate some of this burden git has a command that you can run at any point to get the current status of the repository.

```
git status
```

It is not uncommon to run this command frequently. In fact, as you are learning it would be beneficial for you to run this command frequently so that you may better understand the topics and commands by seeing how they change the repository as it is happening.

Adding Files to the Repository

Anytime a file is changed or a new file is added to the project, the changes will have to be staged within the repository. In order to do this we can run the following command.

```
git add *
```

or

```
git add <filename>.<ext>
```

In the first case the wildcard character of '*' serves to include all files that need to be staged into the repository in a single command. The second demonstrates how to add a specific file to be staged in the repository.

Committing Files to the Repository

As the number of changes made to a repository increase you may want to record those changes to the repository creating a single snapshot of all the changes staged at a given point in time. This is known as a commit. In order to commit those changes you can use the following command.

```
git commit -a -m "added file <filename>.<ext> to the repo"
```

As with all the commands, git commit has a few command line options. The example above demonstrates two of those command line options.

-a : automatically stages any modified or deleted files to the commit. This option will not add new files but takes care of staging anything that was already part of the repository.

-m "<message>" : adds a commit message to the history. This allows for good description of what changes have been stored in this snapshot of the repository. It is highly encouraged to use this option every time a commit occurs.

Pushing Files to GitHub

All the changes you make to a repository are local to your machine. This means if you were to check github after making a change you would not see any of the changes. In fact, github remains unchanged unless you specifically update github and copy the changes. In order to

copy these changes to github you will need to push them there. The git push command allows you to push changes to any remote repository.

```
git push origin master
```

In the case of this example we will be pushing the master branch to the origin remote repository. Upon executing this command we will need to authenticate to the remote repository, assuming it requires such (note: github requires you to authenticate). So in the case of your repository for the pre-work the remote repository is github, and chances are it will be defined as origin.

Accessing the Git Log

Where Git Status focuses on the current state of the repository, Git Log allows you to review the history of the repository. This command has many, many options and arguments. Feel free to review them in your own time. For these exercises, if you want to track what is happening in the repository there are two main options demonstrated below.

```
git log –graph –oneline
```

These options are defined as the following.

—graph : which shows the dot graph of the commits

—online : which limits the notes of the commit to a single line. This makes it very readable but does limit the information shared.

here is a sample output from the advanced exercise that shows the history.

```
$ git log --graph --oneline
*   57ce1e6 fixed the merge issue
|\
| * f500e88 added new line to the file
* | b51564e second line from Ward added
|/
* b6864ab initial commit - adding file1
* e6c49c6 Initial commit
```

Exercise 3: Basic Walkthrough

In this exercise we will walk through the basics of how to add a file to a repository, save the file as part of the repository and push the change to github. This walkthrough can be used for many of the pre-work exercises as they require you to add a file to your repository and push them to github.

This exercises will use the swcguild repository. If you wish to follow this walkthrough with another repository, this exercise assumes the repository is created on github and is cloned to the local file system. If you have not completed these steps please refer to exercise 2 on how to create the repository and also how to clone the repository locally.

Now to get started…

Steps:

1. Create the file in the desired directory. This directory should be inside the directory you created for the repository during the git clone command. In many of the pre-work exercises you will be asked to create a specific directory structure. Do so directly in the repository directory on your local machine.

Ex. ..\swcguild\Pre-Work\Certificates\HTML\Codeacademy

In the above “swcguild” is the repository directory and each additional folder was created within that directory. The file can then be placed in the desired directory.

Running a git status at this point will show that the directory is added and untracked.

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Pre-Work/

nothing added to commit but untracked files present (use "git add" to track)
```

2. Now that the file is in the directory we need to tell git to track the file and stage the change. This can be accomplished using our git add command.

```
$ git add *
```

Running git status shows the file being tracked but not yet committed. Now we see the actual file added to the repository and not just the directory.

```
$ git status
On branch master
```

```
Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Pre-Work/Certificates/HTML/Codeacademy/certificate.jpg
```

3. Now we can commit the change using git commit. Remember to add the -m “<message>” for good measure and better description in the history.

```
$ git commit -m "saved my HTML CodeAcademy certificate"
[master (root-commit) 869cb64] saved my HTML CodeAcademy certificate
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Pre-Work/Certificates/HTML/Codeacademy/certificate.jpg
```

Now git status will show there are no changes to be committed but it does hint that we may need to push the changes to the remote repository.

```
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working directory clean
```

4. All that is left is to push the changes to github using the git push command.

```
$ git push origin master
Username for 'https://github.com': <USER>
Password for 'https://<USER>@github.com':
Counting objects: 7, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (7/7), 2.82 KiB | 0 bytes/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/<USER>/swcguild.git
 * [new branch]      master -> master
```

And now git status has nothing to say except we are up to date.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
```

This process can be followed for all of the pre-work. Just make sure you start with the swcguild repository and add the files to the correct directory. You do not need to run git status with every command but certainly can to get exposure to what is happening at each step. Git status was used here to help illustrate the steps.

Exercise 4: Team Development

In this exercise we will walk through the development process used by two developers. For the sake of keeping things clear let's call those developers Wise and Ward. In this example Ward will start a new repository and begin modifying documents. Early in the process Ward will ask Wise to join the development and make his contributions. As the two developers work together you will see the conflict this causes in the code and the resulting strain on their ability to check files in. The goal of this exercise is to introduce the idea of encountering a conflict committing files and how to merge your changes once you have encounters such a situation. This is a common issue in scenarios where more than one developer will be working in the same solution at the same time.

Note: This example is a precursor to the paired programming you are likely to see in the cohort. We will keep the process simple using only a single branch and instead of code files we will be modify text files. Regardless this is good practice and the topics here will be expanded on in the cohort.

So, let's begin…

Task 1:
Ward is ready to begin development and is going to create a new repository on git hub. We will create a new repository to keep this separate from the rest of the pre-work exercises.

Steps:

1. Create a repository on github to store the files that are used in this practice exercise. Follow the steps from Exercise 2 where we created the repository for the Pre-Work only instead of swcguild we will call this repository "GitPractice".

2. Next we want to clone the repository to our local machine. Since you are going to be working on this exercise alone you will be creating 2 separate repositories to simulate the 2 developers. Therefore you can follow the steps in Exercise 2 for cloning the repository but we are going to add a command line parameter to the git clone command.

```
git clone https://github.com/<user>/GitPractice.git Ward
```

As shown above you can include a directory name for the repository. In this case since we are starting with Ward performing the development we will call the directory Ward. This would then be created in the C:_repos directory or where ever your choose to store your repositories locally on your machine.

Task 2:
With the repository created and cloned, Ward is ready to create the first file and check in the initial file to the repository and push those changes to github.

Steps:

1. Create a text file in the *Ward* directory and name it file1.txt

2. Add some text to the file using your favorite text editor (perhaps Notepad or Notepad++ for you Windows users, while my Linux friends can use gEdit).

Example Text: Ward: This is a text file

3. Save the file.

4. Next Run the git status command. You may need to change the directory within in git bash if it is not already on the Ward directory.

```
$ git status
```

The output of git status shows that the file is untracked and not a part of our repository.

```
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1.txt

nothing added to commit but untracked files present (use "git add" to track)
```

5. Use the git add command to add the file to the repository and stage the changes for commit.

```
$ git add *
```

Running git status again, we can see that the file is tracked and ready to be committed.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   file1.txt
```

6. Use git commit command to save the change to the repository.

```
$ git commit -m "initial commit - adding file1"
[master b6864ab] initial commit - adding file1
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

In this instance, git status shows that we have nothing to commit but encourages us to push our changes to the origin/master we have made modifications that have not been pushed to our remote repository (github in this case).

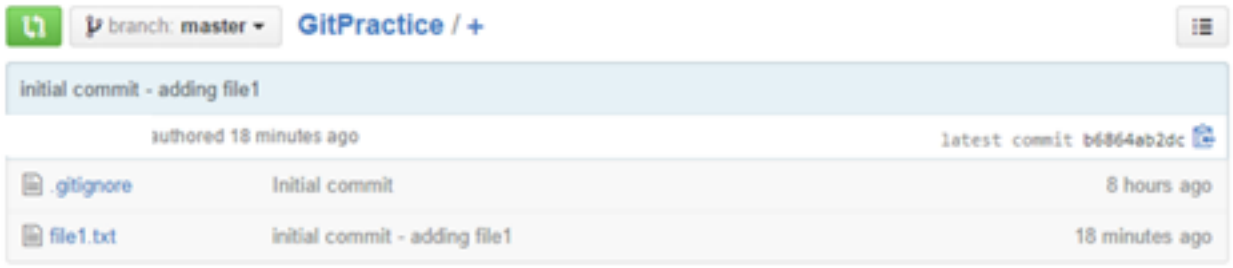
```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

```
nothing to commit, working directory clean
```

7. To push the changes to github we can use the git push command and authenticate to github using our username and password that was used to create the account earlier.

```
$ git push origin master
Username for 'https://github.com': <USER>
Password for 'https://<USER>@github.com':
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/<USER>/GitPractice.git
   e6c49c6..b6864ab  master -> master
```

Our changes will not be visible in github from the website.



Task 3:
Now Wise is now going to join the development effort. We will simulate this by creating a separate directory for his repository and modify the file already in the directory. Once modified we will jump to Ward's file and modify that. Let's see what happens when he tries to push the file.

Steps:

1. Following the instructions for cloning a repository we are going to run the git clone command again. Only this time make sure you also set a directory name and we will call the new directory "Wise".

(Note: make sure the directory is set to the _repos directory)

```
git clone https://github.com/<user>/GitPractice.git Wise
```

2. Open the file1.txt file and modify the file and save.

Example Text: Wise: Yes, this is a text file

Once the changes are saved the git status command will show the uncommitted file again.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
```

```
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file1.txt
```

3. Use the commit command to commit the change to the file.

```
$ git commit -a -m "added new line to the file"
[master f500e88] added new line to the file
 1 file changed, 1 insertion(+)
```

Notice the “-a” on the commit and the lack of an add command call. This is because “-a” takes the place of add. Adding all the files that should be staged to the commit.

4. Push the changes to github

```
$ git push origin master
Username for 'https://github.com': <USER>
Password for 'https://<USER>@github.com':
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/<USER>/GitPractice.git
   b6864ab..f500e88  master -> master
```

JUMP OVER TO WARD’S FILE NOW

Make sure you now have Ward’s file open. The first thing to notice is that Ward’s file doesn't have the line we added to Wise’s. That is ok. Let’s proceed anyway…

5. Modify file1.txt and save the edit as we have done in the past.

Example Text: Ward: adding a new line of text

git status works the same as it has and identifies the file as being changed.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

6. Use the commit command again and again we will add the -a to ensure the change is first staged and then committed.

```
$ git commit -a -m "second line from Ward added"
[master b51564e] second line from Ward added
 1 file changed, 1 insertion(+)
```

7. Now let's push the change up to github

```
$ git push origin master
Username for 'https://github.com': <USER>
Password for 'https://<USER>@github.com':
To https://github.com/<USER>/GitPractice.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/<USER>/GitPractice.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

UH OH! It was rejected!!!

Task 4:
Remember when Wise made modifications, committed them and then pushed them up to github. Well, Ward didn't pull those changes and now the file is out of sync with the remote repository and cannot be committed until Ward pulls these changes and merges them with the changes he wishes to make. To do this we need to pull those changes that are in the remote repository on github. To get those changes we will start by pulling them down, merging changes and then recommitting and pushing to github.

8. Use the pull command to pull down the changes from github. This will actually try and overwrite your files to what is currently on github. You may be thinking, "but, we already made changes… won't we lose those?". Watch what happens…

```
$ git pull origin
remote: Counting objects: 3, done.
Unpacking objects: 100% (3/3), done./3)
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
From https://github.com/<USER>/GitPractice
   b6864ab..f500e88  master     -> origin/master
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

CONFLICT! things just keep getting worse, right? WRONG… This is ok and a normal part of the development cycle. Don't worry and watch how to resolve this conflict.

9. Open the file1.txt again and the file will look something like this:

Ward: this is a text file
<<<<<<< HEAD
Ward: adding a new line of text
=======
Wise: Yes, this is a text file
>>>>>>> f500e88b4cbbb35ec3f522938b3382deb125e58b

10. modify the file to be something more like the below and save the file.

Ward: this is a text file
Wise: Yes, this is a text file
Ward: adding a new line of text

After saving the file you can go to git bash and run git status

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

11. Now you can run commit using the “-a” to stage the changes again.

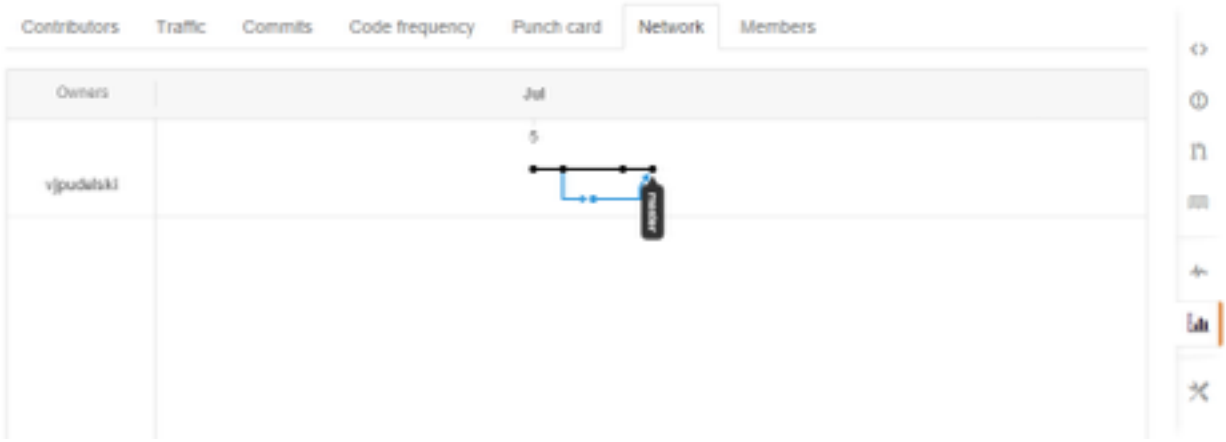
```
$ git commit -a -m "fixed the merge issue"
[master 57ce1e6] fixed the merge issue
```

12. Now use git push again and push those changes up.

```
$ git push origin master
Username for 'https://github.com': <USER>
Password for 'https://<USER>@github.com':
Counting objects: 10, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 672 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/<USER>/GitPractice.git
   f500e88..57ce1e6  master -> master
```

This exercise was an example of doing a merge. You can practice by making modifications and jumping between the two directories on your machine to get the hang of how this works. You will notice that your repository does create a fork and then merges together. This is because the merge creates an extra commit to bring the two branches (master and origin/master) in sync.

To view this fork you can use the git log command as mentioned above. You can also review the repository in github and aee this change. On the repository page go to Graphs on the right hand side and select the Network graph. It should appear like the following.



During the cohort we will look at other options to resolve these types of issues including commands such as rebase and what the differences are between rebase and merge.