GIT AND HEROKU SETUP

This exercise is intended to walk you through setting up accounts for Github and Heroku and to get started with Git.

Due by the date on the dropbox. Submit your modified Merge_Edits.txt and/or ASCII file to the dropbox along with links in your submission comments to

- **Your Heroku application (the page saying “welcome to your new app”) and**
- **Your forked Github repository for the ASCII section of the Git exercise.**

<u>**Exercise Contents**</u>

GIT AND HEROKU SETUP

Setting up Github

In this class you will be using version control. In Heroku, you use Git to upload files. Unlike our servers, Heroku does not have FTP. It only has Git. All files are uploaded to the app with Git.

What is Git?

Git is a distributed version control software (DVCS). In traditional version control systems, there was a central repository that held all of the code.

A *repository* is a data structure used by version control software to hold all of your files and the history of each change. This allows you to rollback to previous versions of files and see who has done what.

In traditional version control systems (like SVN), there was a central repository stored on the server that everyone connected to. This mean that if the server went down or if you lost connection, you could not sync you code. It also makes copies of the code (forks) more difficult to accomplish.

In a distributed version control system, every person's computer is a full-fledged repository. In the event of a server failure or a connection error, you can still sync your code to your local machine and to anyone else who needs it. It is a peer-to-peer system where one or more of the "peers" may be a server. This makes sharing your code with people much easier.

What is Github?

Github is a collaborative git repository web service that also contains a web UI, a task tracker, wiki pages and more. It actually works a lot like a social network in that has feeds, followers, organizations, etc. At its core though, Github is just a server system for hosting git repositories.

A Github repository is just one of the "peers" in the peer-to-peer organization of git. It may be one of the places you sync your code to. It makes it easy for other people to see your code and sync with it. In multi-developer situations, Github lets each developer sync their code back to Github so that the others can get it.

Git and Heroku Setup

Since Github is just a web service for repositories, you can use git without it. Github is also not the only service that does this. There are several others like Bitbucket.

Creating a Github Account

1. Navigate to https://education.github.com/benefits?type=student.

2. Click "Get Student Benefits", and ensure you are logged in to GitHub with the account you use for classes. If you don't have a GitHub account, create one now.

3. Follow the on-screen directions to verify your student status. You will need to upload something with a date on it that proves your student status: most likely the back of your student ID.

   Note that if for some reason it rejects your image, students have found success by activating their webcam and taking a picture of their ID using the in-browser option that GitHub provides.

4. It will take some time to process your request, but most are processed in a minute or two. If the page seems to be stuck on processing, check your email. Chances are the page just bugged out and your account was verified.

5. Once you are signed up, you should see your dashboard. You can follow people or projects, fork projects to make your own version, create new git repositories, etc.

6. Check out our class organization under **IGM-RichMedia-at-RIT**.
   https://github.com/IGM-RichMedia-at-RIT

7. Once you have created your account, email your username to your professor (or tell us in person if we are available) requesting access to the class organization. Once we add you, you will have write access one of the repositories used in this exercise. **You will need to accept the invite before you get access!**

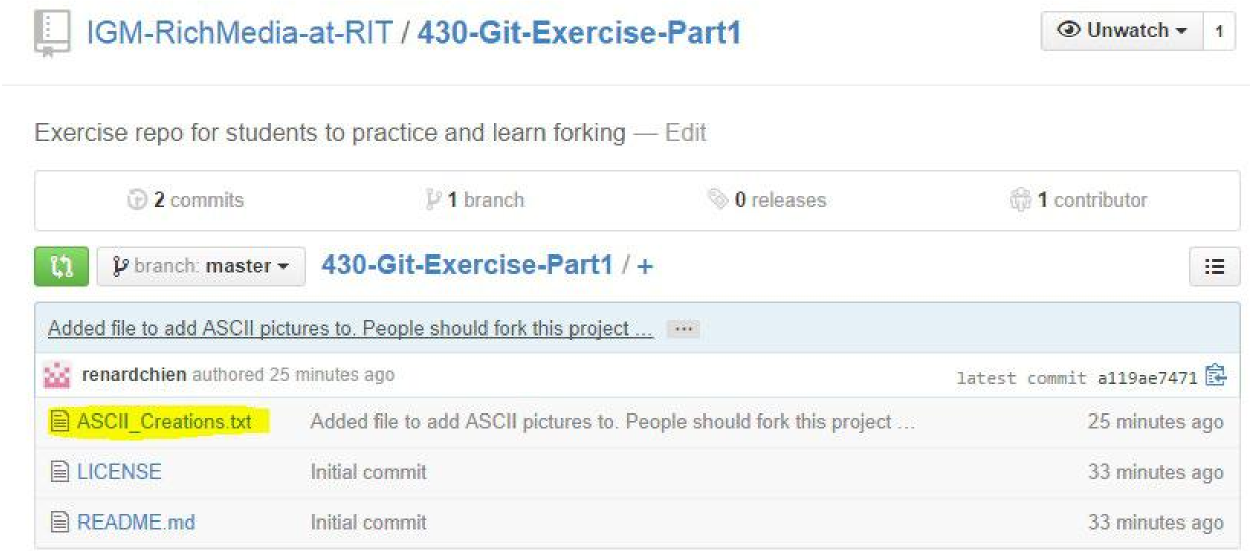
Using Git Part I

In this part of the exercise, you will *fork* one of our repositories and change the content to your own. A *fork* is a separate copy of the repository.

You **do not** need to be added to the organization to do this part. You may do this section will you are being added to the organization.

1. Go to the repository for exercise part 1. https://github.com/IGM-RichMedia-at-RIT/430-Git-Exercise-Part1

2. Click on the link for ASCII_Creations.txt

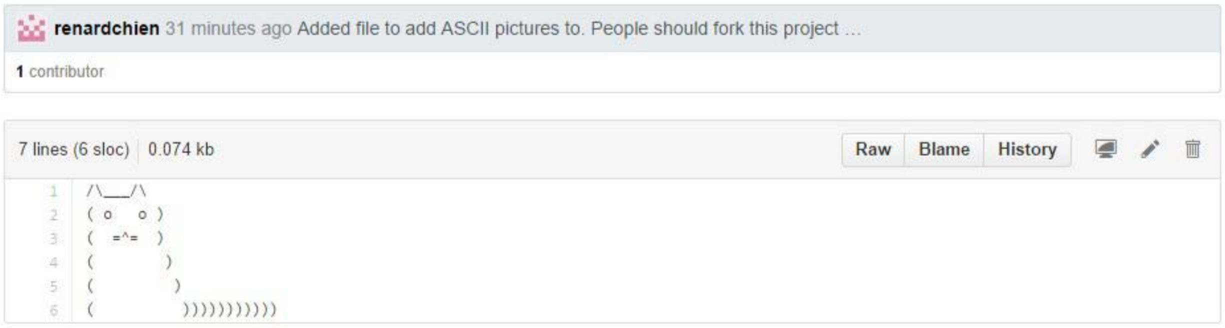

IGM-RichMedia-at-RIT / 430-Git-Exercise-Part1
Unwatch 1
Exercise repo for students to practice and learn forking — Edit
2 commits
1 branch
0 releases
1 contributor
branch: master
430-Git-Exercise-Part1 / +

Added file to add ASCII pictures to. People should fork this project ...

renardchien authored 25 minutes ago latest commit a119ae7471

ASCII_Creations.txt	Added file to add ASCII pictures to. People should fork this project ...	25 minutes ago
LICENSE	Initial commit	33 minutes ago
README.md	Initial commit	33 minutes ago

It should open the file so you can view the “source code” - AKA, this amazing picture.


renardchien 31 minutes ago Added file to add ASCII pictures to. People should fork this project …

1 contributor

7 lines (6 sloc) | 0.074 kb

```
/\___/\
( o   o )
(  =^=  )
(        )
(         )
(          )))))))))))
```

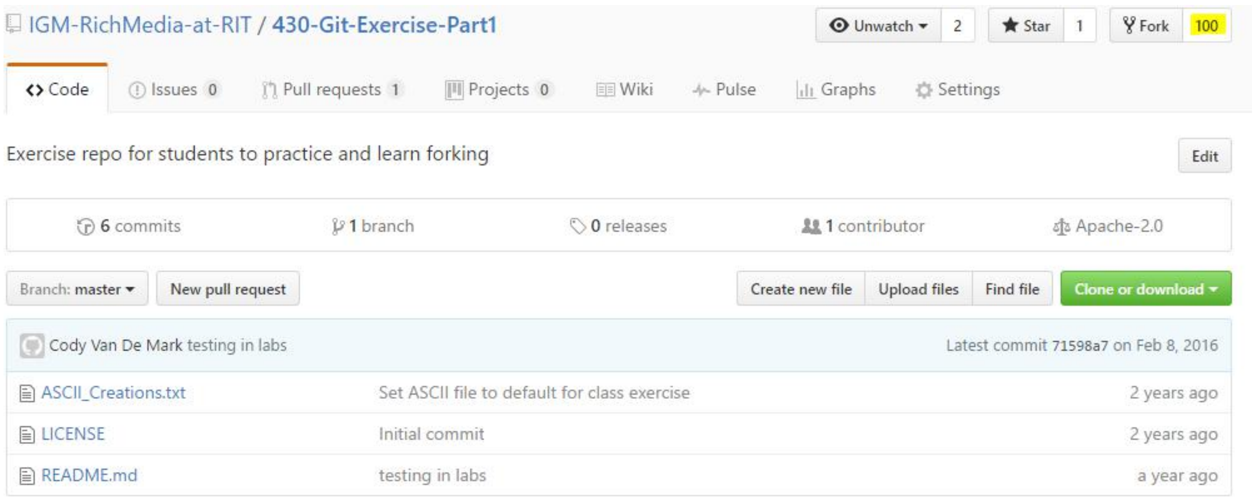
You are going to make your own fork of the repository. A fork is a copy of a repository. People make forks so that they can build on other peoples’ code. Forks are often done to fix bugs or take projects in a different direction.

A good example of this is LibreOffice from OpenOffice. Unhappy with the direction of OpenOffice, people forked the open source code from OpenOffice and made their project called LibreOffice.

Not all forks are to make a new project though. Oftentimes people will not have access to make changes to a repository. Instead they will make a fork, fix a bug or add a feature and then send a request to the original owner to pull in their changes. This is referred to as a pull request.

3. Go back to the Exercise Part 1 page and hit the fork button. This will create a new copy under your account that is owned by you. Github lets you have any number of open source repositories for free (ones you create, ones you share and ones you fork).

   If it asks, put it under your own account.



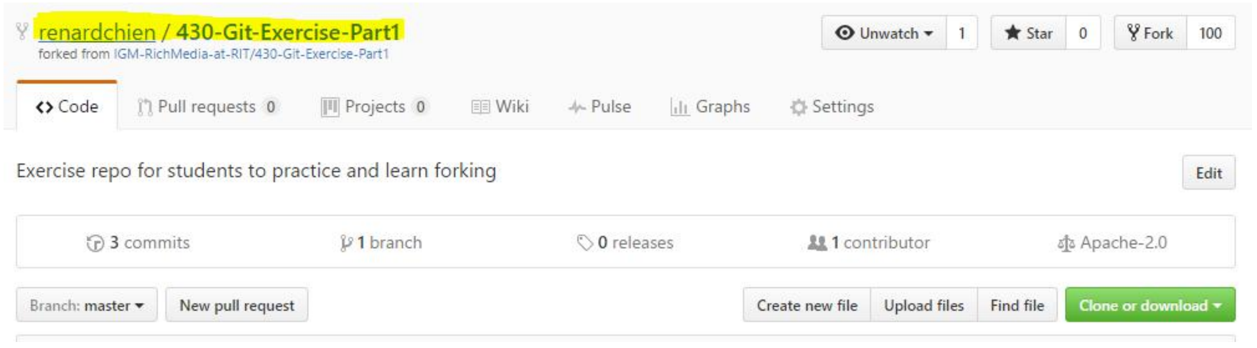
IGM-RichMedia-at-RIT / 430-Git-Exercise-Part1

Exercise repo for students to practice and learn forking

6 commits | 1 branch | 0 releases | 1 contributor | Apache-2.0

Cody Van De Mark testing in labs — Latest commit 71598a7 on Feb 8, 2016

ASCII_Creations.txt	Set ASCII file to default for class exercise	2 years ago
LICENSE	Initial commit	2 years ago
README.md	testing in labs	a year ago

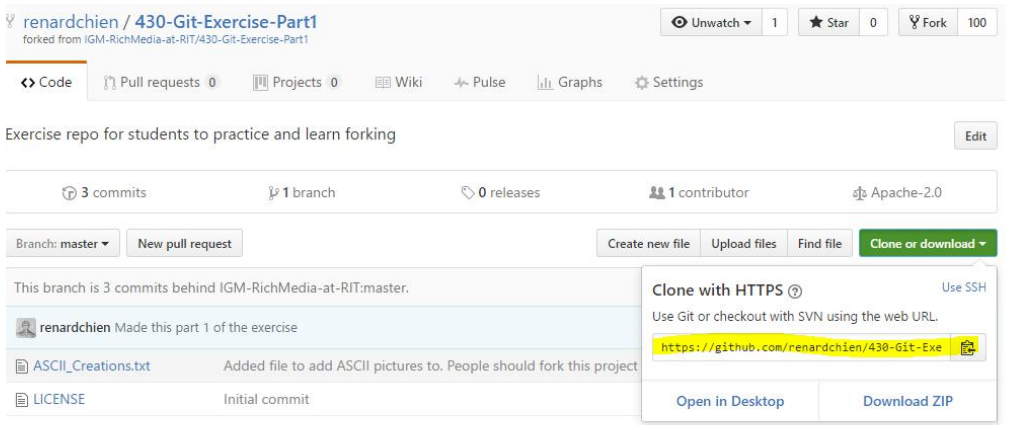


renardchien / 430-Git-Exercise-Part1
forked from IGM-RichMedia-at-RIT/430-Git-Exercise-Part1

Exercise repo for students to practice and learn forking

3 commits | 1 branch | 0 releases | 1 contributor | Apache-2.0

4. Copy the CLONE URL from **your** new repository (highlighted in yellow). This is the one under your account, not the original. If you hit the little clipboard looking button next to the URL, it will copy the entire URL to your clipboard for pasting.



Exercise repo for students to practice and learn forking

This branch is 3 commits behind IGM-RichMedia-at-RIT:master.

renardchien Made this part 1 of the exercise

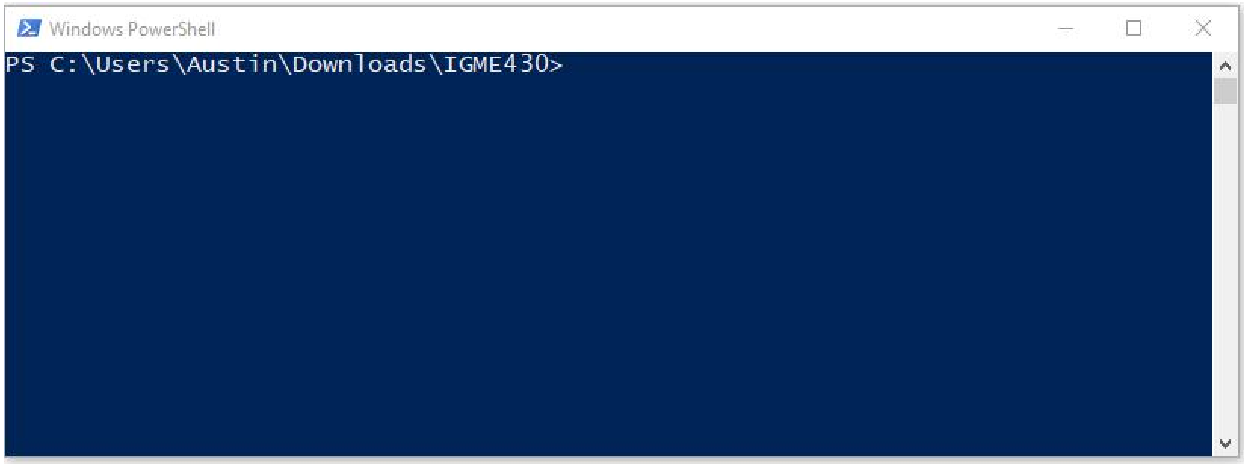
ASCII_Creations.txt	Added file to add ASCII pictures to. People should fork this project
LICENSE	Initial commit

Use Git or checkout with SVN using the web URL.

```
https://github.com/renardchien/430-Git-Exe
```

5. On your local machine, open a folder where you want to download this project to. Any folder is fine. Then in that folder open PowerShell (Windows) or Terminal (Mac/Linux).

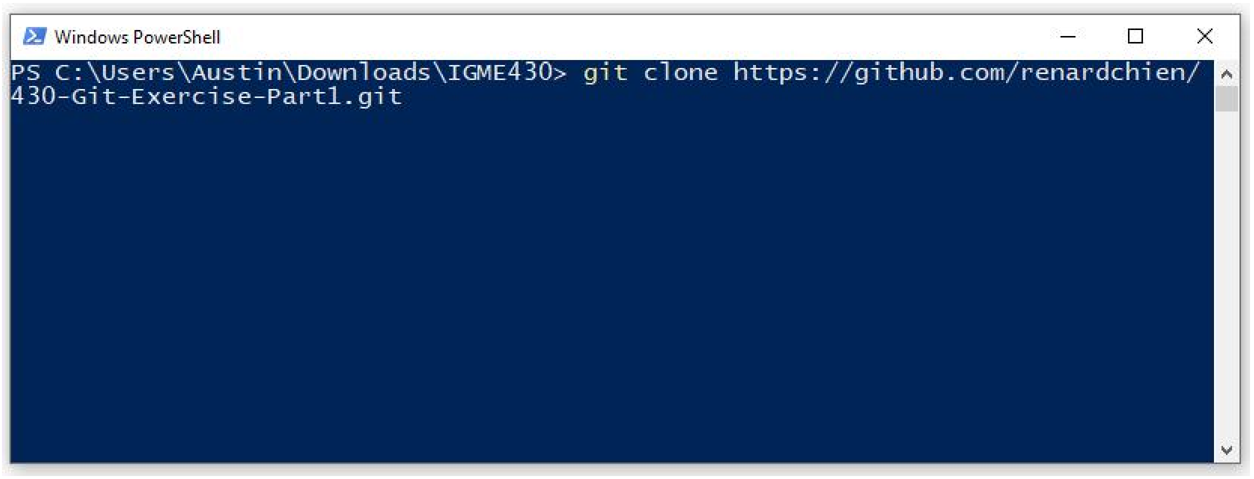
   To open PowerShell on Windows, hold down Shift, and then Right Mouse click in the folder you want to open PowerShell in. Then select “Open PowerShell window here”.



```
PS C:\Users\Austin\Downloads\IGME430>
```

6. Once you have the window open and the file path is to the folder you want, type in “git clone”, then paste in the CLONE URL you copied from Github.

   Hit enter once you have the command in and it should download the repository.



```
PS C:\Users\Austin\Downloads\IGME430> git clone https://github.com/renardchien/
430-Git-Exercise-Part1.git
```

Once it downloads, you should see a new folder of code in the folder you choose. Inside of that should be all of the code from your repository.

Git's clone command creates a full duplicate of the repository. The copy you now have on your machine will match the copy on Github.

You could now push that repository to other machines and they would all act as if they have a full copy of the repository.

7. Inside of the new folder, open the ASCII_Creations.txt file in Notepad++ or another editor.

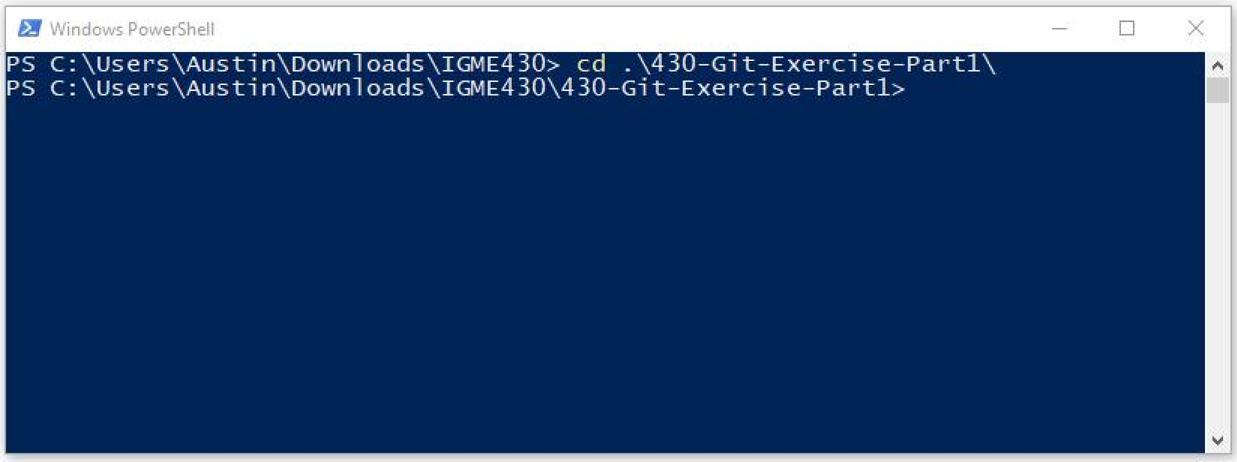
   Change or add ASCII art to this. **Do not make it**

   inappropriate. Once you are done, save the file.

8. Now go back to PowerShell and go into the folder from the command line. Use unix's change directory command (cd) followed by the name of the folder.

   Type "cd" (no quotes) followed by the folder name you want to go into.

   You can type the beginning of the folder name (likely "430") and then hit tab twice. Tab is the autocomplete key. You can also type "cd" followed by a space and then just hit tab. It will try to autocomplete what you have typed, or cycle through all the directories in the folder if you didn't type anything.



```
PS C:\Users\Austin\Downloads\IGME430> cd .\430-Git-Exercise-Part1\
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

"cd" is a Unix command, not a Git command. That means all Unix systems have this built-in regardless of whether or not Git is installed. If you opened terminal on Mac/Linux, you would be able to use this command without Git installed.

Once you have changed directory to the new folder, you should see the file path has changed in the terminal (as shown above).

9. Type in the following two commands. Replace the name and email address with your appropriate name/username and email.

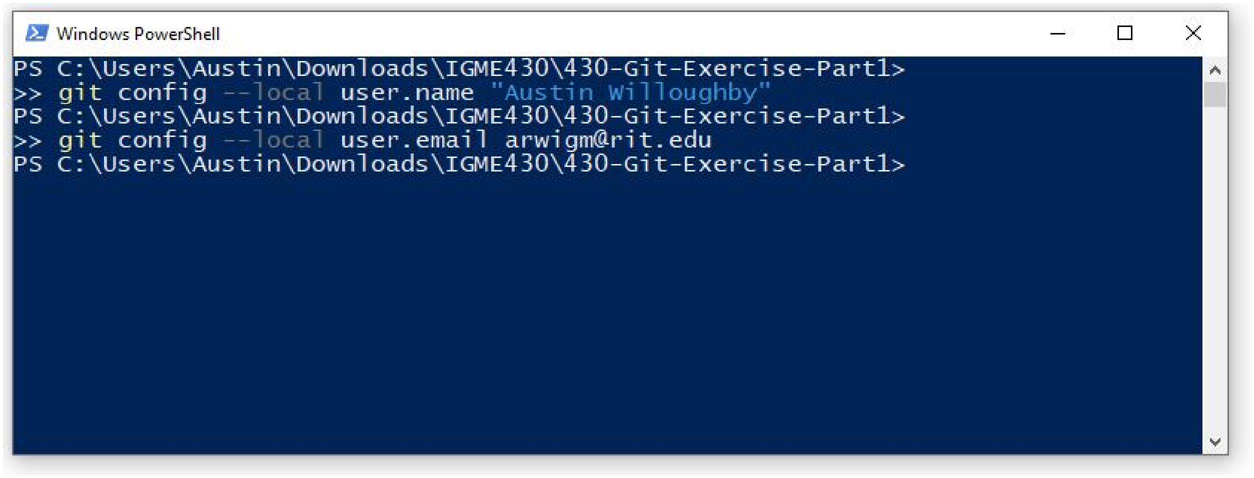
   Note: If you are using your own machine, then you may use --global which will set your name and email for all of your git projects.

   "git config --local user.name '*your name or username or nickname*' "

   "git config --local user.email *your_email_address* "

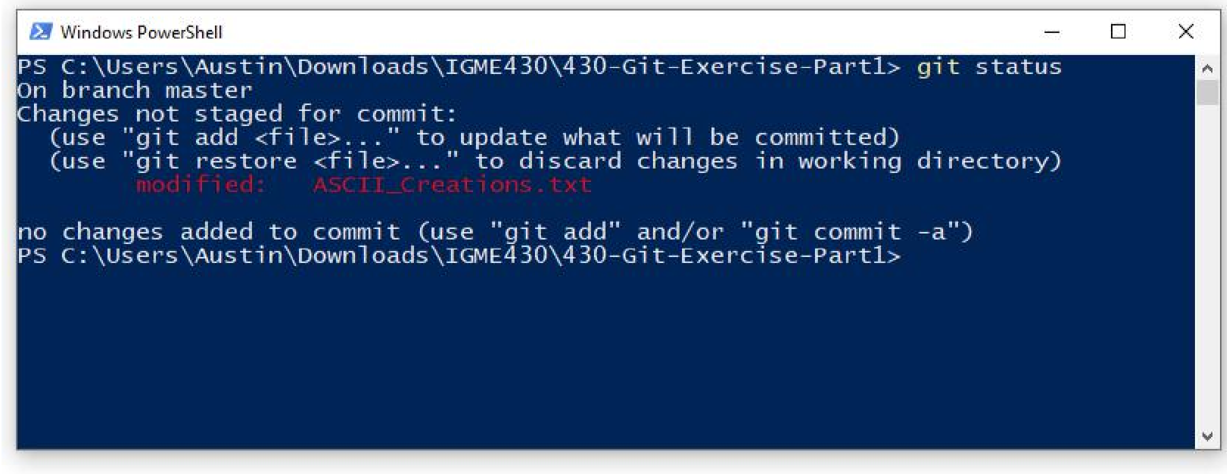
   That will tell this repository to make all commits using this name and email

Example:



```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
>> git config --local user.name "Austin Willoughby"
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
>> git config --local user.email arwigm@rit.edu
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

10. Type in “git status”. It should tell you that you have modified the file. Git status tells you the state of the files in this repository.



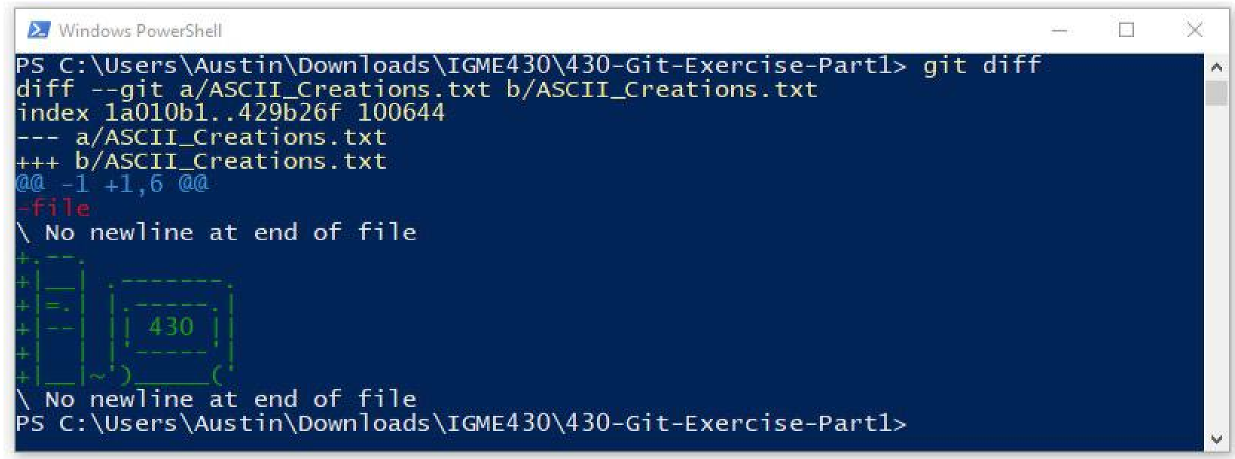
```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ASCII_Creations.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

In this case, it should show you that you have modified the file (shown in red). Notice it says “Changes not staged for Commit”. Git knows the file has changed, but it has not added it yet.

11. Now type in “git diff”. This shows you all of the lines that have been changed. Lines with a + in the front mean they have been added. Lines with a – in the front means they have been removed. Lines with nothing in front means they are unchanged. These lines are usually only shown to give you context.

Your ASCII art will be different that mine.

Hit “q” to close gif diff

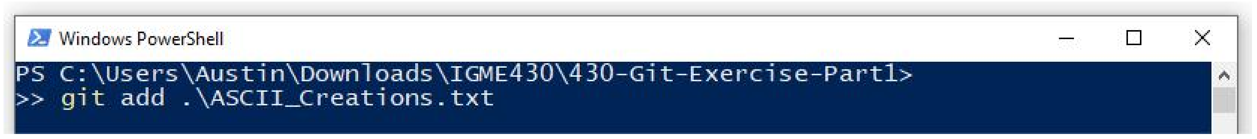


```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1> git diff
diff --git a/ASCII_Creations.txt b/ASCII_Creations.txt
index 1a010b1..429b26f 100644
--- a/ASCII_Creations.txt
+++ b/ASCII_Creations.txt
@@ -1 +1,6 @@
-file
\ No newline at end of file
+.---.
+|___| .-------.
+|=. | |.-----.|
+|-- | || 430 ||
+|   | |'-----'|
+|___|~')_____('
\ No newline at end of file
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

12. These changes have been made in the files, but they have not changed in the local repository. The local repository is the data structure on your machine controlling the history and changes of the code. We need to add your changes to the repository using Git’s “add” command.

    Git’s “add” command adds a file to the **queue** of files to be added to the repository. It does not actually add them, just stages them for review and to be added to the repository later.

    Type “git add” and the filename(s) to add. In our case ASCII_Creations.txt



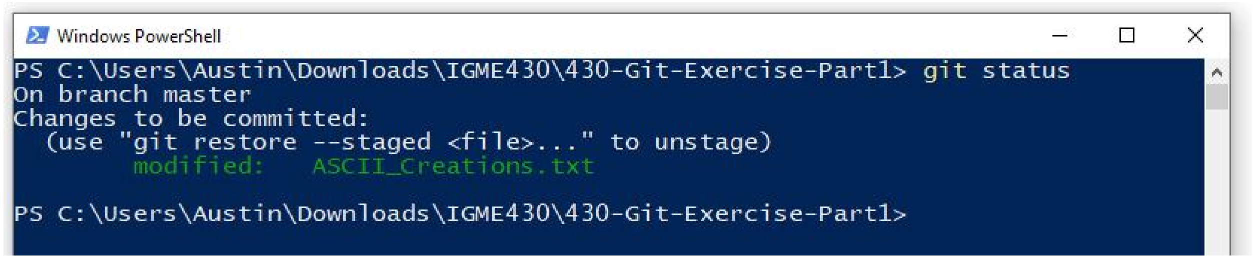
```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
>> git add .\ASCII_Creations.txt
```

    Remember that you can also make use of the “tab” key to autocomplete things in Powershell / Terminal. If you want to simply stage **all** files that have been edited. You can use “git add .”

13. Type git status again and it should have updated to “Changes to be committed”. The modified text should now be green instead of red, indicating that it is in the queue of files to be added.

    The reason you do this is, is that when making a new feature you probably altered multiple files. If adding a file directly updated the repository, there would be a chance someone else grabbed updates from the repository before you finished adding in all of the files and the code would break for them.

Instead git waits until you have queued all of the files you want to add and then it adds them in all at once. When someone syncs their code with the repository, they get all of the files from your change so that there are no missing files (unless you actually forgot to add them).

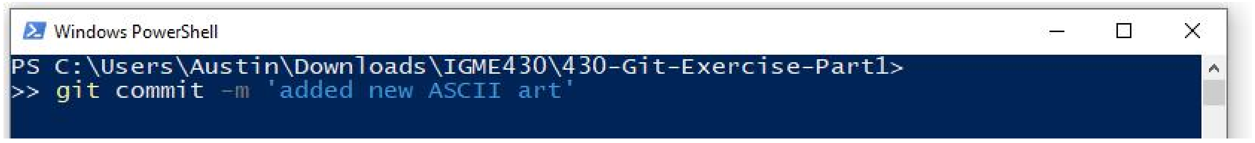

Windows PowerShell

```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   ASCII_Creations.txt

PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

14. Once you have added the modified file, we need to “commit” it to the repository. Committing means that you are adding all of the changes in the queue to the repository along with a message of what you did.

    Type in “git commit -m ‘some message text’ “. The “-m” flag indicates that you are typing a string message about what you did.

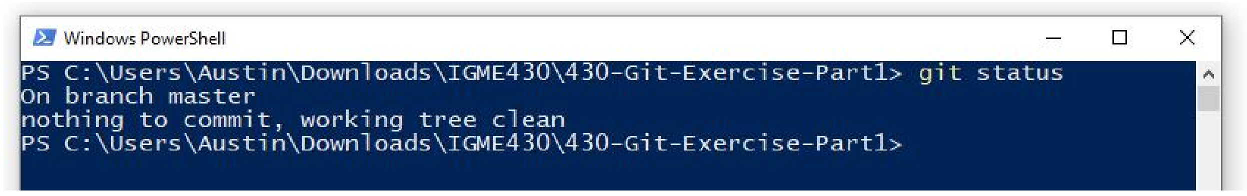
    Remember: Make these messages useful. Explain briefly what you did. One line is fine, but it should be a clarification of what you changed, not just ‘did stuff’.


Windows PowerShell

```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
>> git commit -m 'added new ASCII art'
```

15. Now type “git status”. It should tell you there is nothing to commit. Your local files are synced with your local repository. Now something to keep in mind is that your local repository is tracking all of your changes, but it is not synced to any other repository.

    You either need to manually push or pull changes to sync with another repository.



```
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\Austin\Downloads\IGME430\430-Git-Exercise-Part1>
```

16. You can manually push your changes to another repository with "git push" Git's push command takes the name of a remote repository and the name of a branch.

    We won't worry about those for now. You can manually add remote addresses and give them names, but when you clone a repository a remote repository is automatically added called 'origin' that points to the server it came from.

    Similarly, repositories can hold multiple sets of the code simultaneously. I don't mean the history (it does that anyway), but it can hold entirely different sets of same code with different histories and changes. These are referred to as "branches". By default, one branch is always created called "master".

    Run the command "git push origin master". That will say take the local master set of code and push it to the master set of code at the origin (which is Github). When it asks, provide your github username and password.

17. Now go back to your fork of the repository on Github. You just synced your local repository to the one on Github. You should see all of the changes you just made are now on the Github web page for your repository.

18. Now go back to the original organization's version of the repository. Notice that is has not been changed. This is because your "fork" of the repository is entirely separate. You don't have access to modify my repository; instead you can do anything you want with your forked copy.

19. Since your fork on Github has been updated (**and you have confirmed that!**), you can now delete the code on your local machine (if you want to). You will always be able to clone the latest from your fork on Github onto any machine with git installed.

GIT AND HEROKU SETUP

Using Git Part II

To do this section, you must have been added to the organization account. If that has not been done, please remind us or wait a little bit longer if you know we are working on it.

1. Check out exercise part II on our organization account. If you have been added to the repository, then you will have write access to this repository. This is an exercise in working together! https://github.com/IGM-RichMedia-at-RIT/430-Git-Exercise-Part2

2. Instead of making a fork, grab the CLONE URL of this repository. We will all write content back to this directory.

3. Find a folder on your local machine and clone this project using git clone. By the time you do this, some people may have already started modifying the code on Github. That is okay!

4. “cd” into the new folder that was created from git clone.

5. Set your name and email for this repository using git config. If you are using your own machine and you used --global in the previous section, you can skip this step.

6. Open the Merge_Edits.txt file from the folder you cloned. You should see a lot of text.

7. Inside of Merge_Edits.txt, go ahead and modify the existing text. Also add your own. **Do not make it inappropriate.** It does not need to be a lot or meaningful, but please make a couple small changes (of both modification and addition) and save the file.

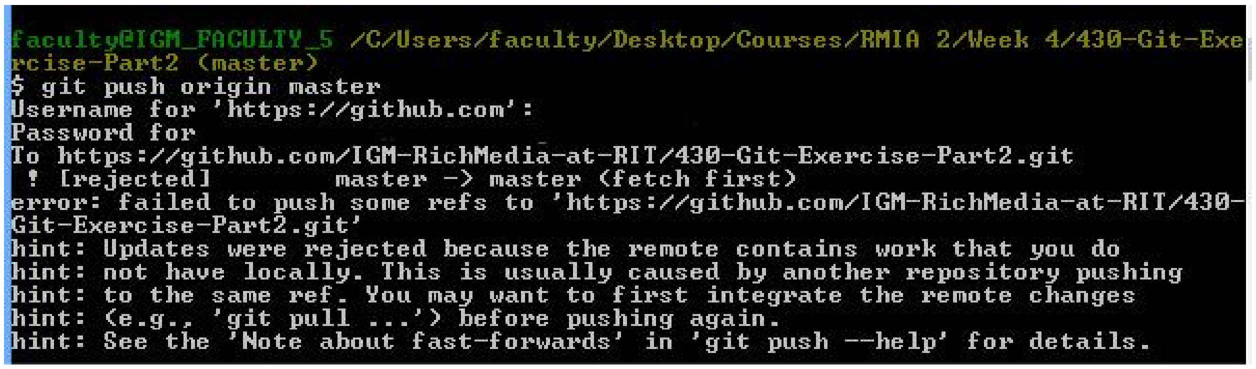
8. Once you have finished, add the changed file to the queue using "git add". Remember you can track what has changed with "git status"

9. Now add the file to your local repository using "git commit -m *'message'*". Please add an appropriate comment.

10. Use "git status" to make sure all of your changes have been committed.

11. Now sync your changes back to the organization's Github repository using "git push origin master"

If someone else has modified the repository since last time you pulled or cloned, then your changes will get rejected. That looks like this. Note that this screenshot is from Git Bash, not PowerShell. The text output should be similar. We tend to avoid Git Bash in this class because it can conflict with how Node works.



```
faculty@IGM_FACULTY_5 /C/Users/faculty/Desktop/Courses/RMIA 2/Week 4/430-Git-Exe
rcise-Part2 (master)
$ git push origin master
Username for 'https://github.com':
Password for
To https://github.com/IGM-RichMedia-at-RIT/430-Git-Exercise-Part2.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/IGM-RichMedia-at-RIT/430-
Git-Exercise-Part2.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.
```

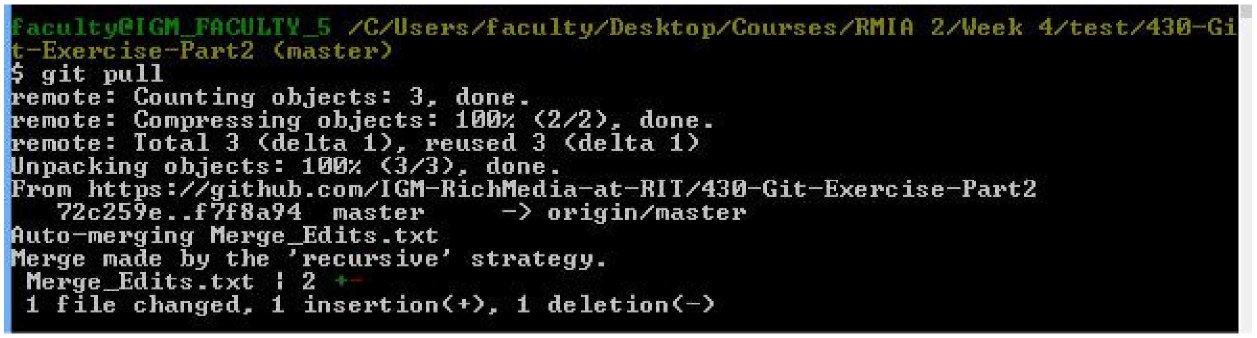
If no one has made changes, then it should push successfully and you can see your changes in the Organization's repository along with everyone else's changes. If so, jump to the submission step at the end of the exercise.

If it did not push successfully, then move onto the next step ☺

12. If you did not push successfully, then you need to grab the updates. You do this with the command "git pull". Git pull can take a location to check for updates from, but by default it will check for updates from the origin, which is Github.

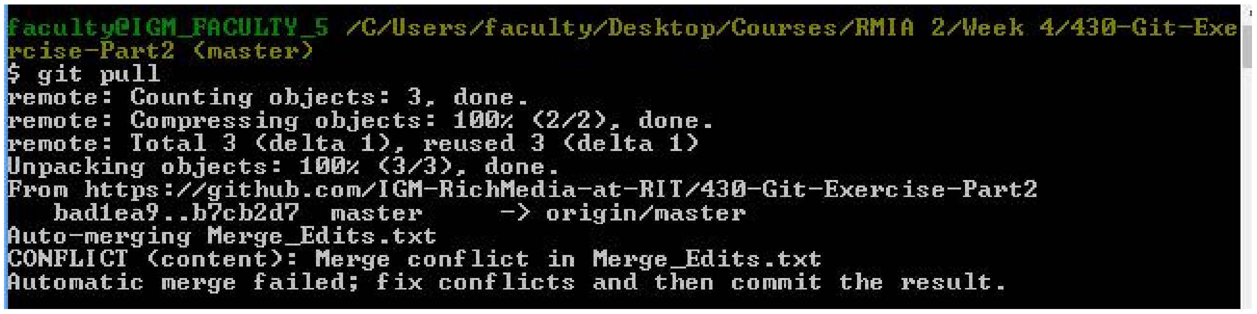
Go ahead and run “git pull”. Git pull automatically pulls down updates and merges them into the code.

If it auto-merges and there are no conflicts, go ahead and move onto step 14. It will look like similar to this if successful.



```
faculty@IGM_FACULTY_5 /C/Users/faculty/Desktop/Courses/RMIA 2/Week 4/test/430-Gi
t-Exercise-Part2 (master)
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/IGM-RichMedia-at-RIT/430-Git-Exercise-Part2
   72c259e..f7f8a94  master     -> origin/master
Auto-merging Merge_Edits.txt
Merge made by the 'recursive' strategy.
 Merge_Edits.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

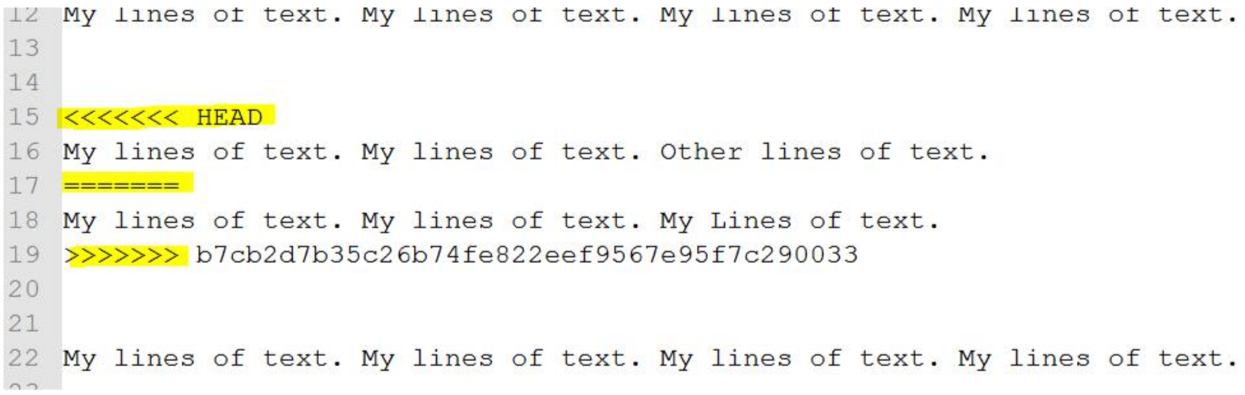
If you get a conflict error, continue on with this step on the next page. In the event that someone modified the same lines as you, then you will see a CONFLICT error. All of the CONFLICTS will be marked in the file that had conflicts.



```
faculty@IGM_FACULTY_5 /C/Users/faculty/Desktop/Courses/RMIA 2/Week 4/430-Git-Exe
rcise-Part2 (master)
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/IGM-RichMedia-at-RIT/430-Git-Exercise-Part2
   bad1ea9..b7cb2d7  master     -> origin/master
Auto-merging Merge_Edits.txt
CONFLICT (content): Merge conflict in Merge_Edits.txt
Automatic merge failed; fix conflicts and then commit the result.
```

If you have conflicts, open the Merge_Edits.txt file and find the conflicts. Anywhere there is a conflict, Git will mark the lines for you.

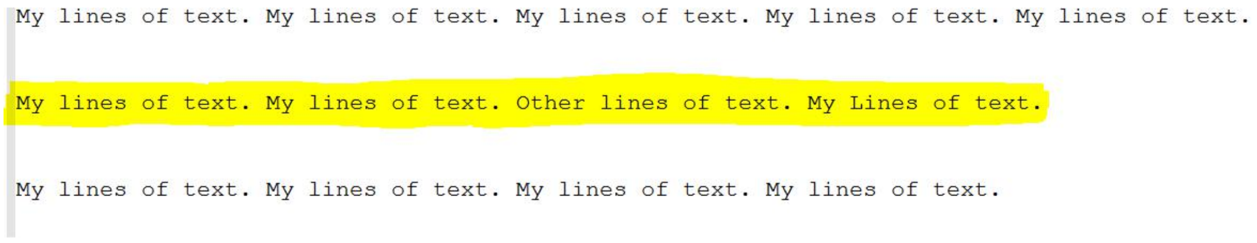
- The arrows left with the word 'HEAD' is above the lines you changed.
- The equals signs show the end of your changes following by the conflicting changes that occurred.
- The right arrows with the hash code show the end of the conflicted changes.



```
My lines of text. My lines of text. My lines of text. My lines of text.


<<<<<<< HEAD
My lines of text. My lines of text. Other lines of text.
=======
My lines of text. My lines of text. My Lines of text.
>>>>>>> b7cb2d7b35c26b74fe822eef9567e95f7c290033


My lines of text. My lines of text. My lines of text. My lines of text.
```

Resolve all of the conflicts by choosing the changes you want and deleting the rest of the text that git generated.

In the above example, I just merged the two conflicts into one line and deleted the rest or git's conflict text, like so. The lines in the above example got merged into the single line below and I saved it.



```
My lines of text. My lines of text. My lines of text. My lines of text. My lines of text.

My lines of text. My lines of text. Other lines of text. My Lines of text.

My lines of text. My lines of text. My lines of text. My lines of text.
```

Resolve all of the conflicts by replacing the conflicts with the changes you want to make and save the file.

Once you have fixed merge conflicts, go ahead and add the file with "git add" again and commit your changes locally

13. Once you have retrieved updates push the updates back to the server. If you are lucky, no one else made changes during that time :D.

    Otherwise, you will need to repeat steps 12 and 13 until successful.

    If this worked correctly, you should see your changes on the Organization's repository page.

Setting up Heroku

In this section, you will set up an account on Heroku. You MUST have GitHub Student setup before you can setup Heroku.

Note: You will need to use a Credit Card with Heroku in order to verify your identity. You will **not** be charged unless you do not properly setup GitHub Student as shown above, so make sure you have done that correctly.

What is Heroku?

Heroku is a *Platform as a Service (PaaS).* Basically, Heroku is a platform of cloud computing instances that can just be used like a service. You control your instances through the Heroku CLI and the Web Dashboard.

Each instance is like a small computer that can run your code. Just choose the add-ons you want from Heroku and they get installed into whichever instances you need them on.

Need PHP with MySQL? You just need to choose PHP and add the MySQL add-on.

Need Ruby with MongoDB, SMS messages and a search engine? Again, just add-ons.

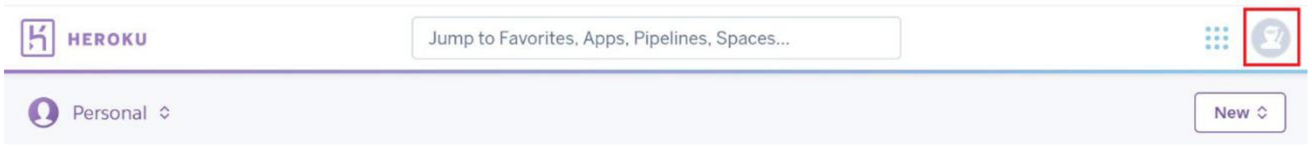
GIT AND HEROKU SETUP

Setting up an Account

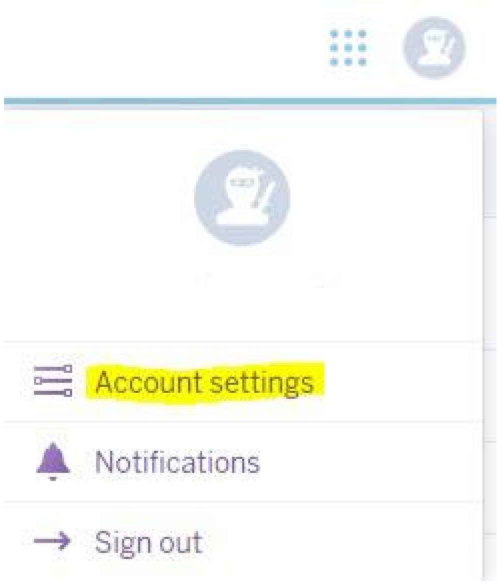
1. Go to https://www.heroku.com/

2. Create a free account. Make sure you use a secure password and do not share it with anyone under any circumstances. You will need a valid email to sign up. You do not have to use your RIT email if you do not want to.

3. Once you have signed up, go into your Heroku Dashboard



4. At your dashboard, go to your account in the top right and choose "Account Settings".



5. In the "Account" section, set up Multi-factor Authentication. Heroku will require MFA as of February 2022. I recommend using an authenticator app like Authy, but any of the options work.

6. At the “Billing” section, add a credit card provided I assure you nothing we do in this class will incur any charge provided you follow the instructions in this document.

7. We now need to link your Heroku account to your GitHub Student account to give you access to the $156 credit that the Student Developer Pack provides to Heroku users. Navigate to http://www.heroku.com/github-students/signup. Be aware that this is the sketchiest looking legit webpage you have ever seen.

8. Once logged in, follow the on-screen directions to link your Heroku and GitHub Student accounts.

9. Once the two are linked, it may take up to 24 hours to process your credit. Once the credit has been applied to your account, you should be able to see it by navigating to the Account Settings in the top right corner of https://dashboard.heroku.com/ and going to the Billing tab. There you should see Platform Credits equal to $156.00.

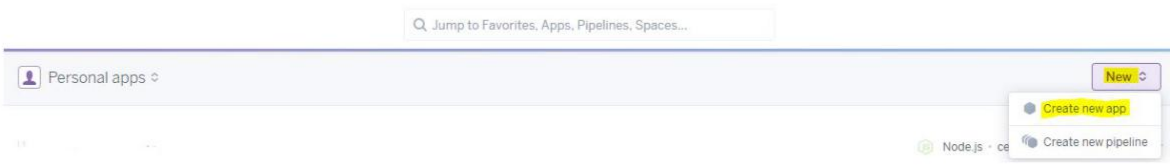
10. Now that you have these platform credits, any expenses charged to your Heroku account will consume those funds instead of charging your credit card. We will be using this to sign up for Heroku’s Eco Plan. Scroll down on the Billing page (where you see the Platform Credits). Click the button to subscribe to the Eco Dynos Plan. The $5 a month charge will be billed to your platform credits every month.

    Note: The Eco Dyno plan allows you to make as many apps as you want using Eco Dynos (as long as they don’t use over 1000 hours a month collectively). Eco Dynos are simple applications that run on Heroku that turn off after 30 minutes.

    There are “better” dynos, like Basic, Standard, etc. however these cost money **per application**. If we were to use anything over than Eco Dynos, we would blow through the $156 in a matter of months. By default, Heroku will always make your apps using Eco Dynos, so as long as you don’t change any settings you will not need to worry.

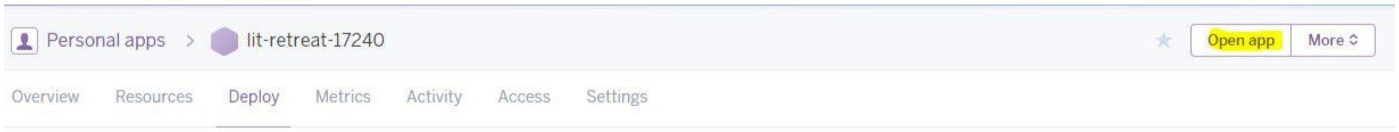
Creating a Heroku App

1. Now return to your Personal Apps page and create a new app using the new button. You may choose to give your test app a name (otherwise a random one will be generated for you). If you choose to name it, the name should be appropriate. All names need to be unique. Create it in the United States region.

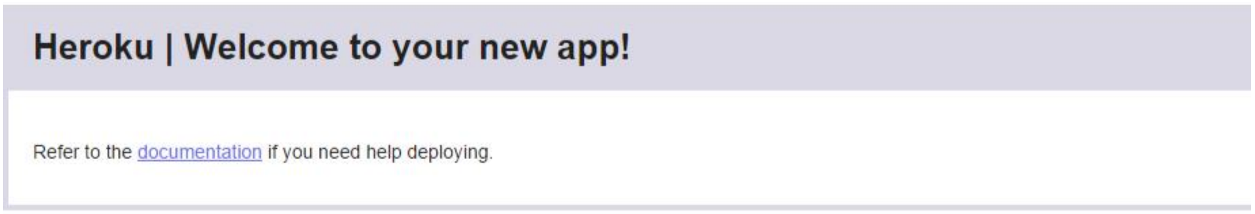


Once you create an app, you should see a new app page. On this page you will see instructions on getting started. All of these instructions have been customized and generated for your app. If you notice, the instructions use your email, your app name, etc.

2. On the app page, test your new app by choosing "open app"



It should open a tab with this server instance running. Apps may have 1 or more server instances running on them. Each server instance on Heroku is called a Dyno. The free account only allows 1 dyno per app, but it allows 100 free apps per account (if you have entered your credit card info). That's 100 different servers for you. If you need a second server of your app, you cannot get another dyno without paying, but you can choose an option to create a duplicate of this app (second server of it, but only one dyno each)


Heroku | Welcome to your new app!

Refer to the documentation if you need help deploying.

Congrats, you have your own personal web server ☺

<u>**Submission**</u>

Submit your modified Merge_Edits.txt and/or ASCII file to the dropbox along with links in your submission comments to

- **Your Heroku server (the one saying "welcome to your new app") and**
- **Your forked Github repository for the ASCII section of the Git exercise.**

The submissions just tell us who is completed with the exercise. That way we don't have to poll each person randomly to see if they have submitted the assignment yet.

What we will really be looking for is the link in the submission comments.