

Github Project: Git Commands Documentation

Programming for Data Science Nanodegree Program

1. Set Up Your Repository

The following are the steps you will take to create your git repository, add your python code, and post your files on GitHub.

Step 1. Create a GitHub profile (if you don't already have one).

Step 2. Fork a repository from Udacity's [GitHub Project repository](#) and provide a link to your forked GitHub repository here:

GitHub Repository Link

https://github.com/rohitkaundal19/pdsnd_github

Step 3. Complete the tasks outlined in the table below and copy and paste your git commands into the "Git Commands" column. The first git command is partially filled out for you.

	Tasks	Git Commands
A.	Clone the GitHub repository to your local repository.	<code>git clone https://github.com/rohitkaundal19/pdsnd_github</code>
B.	Move your bikeshare.py and data files into your local repository.	No git command needed (you can use cp or a GUI)
C.	Create a .gitignore file containing the name of your data file.	No git command needed (you can use touch or a GUI)
D.	List the file names associated with the data files you added to your .gitignore	No git command needed (add the file names into your .gitignore file)
E.	Check the status of your files to make sure your files are not being tracked	<code>git status</code>
F.	Stage your changes.	<code>git add .</code>
G.	Commit your changes with a descriptive message.	<code>git commit -m "Adding only rohit_bikeshare.py file"</code>

H.	Push your commit to your remote repository.	<code>git push origin master</code>

2. Improve Documentation

Now you will be working in your local repository, on the BikeShare python file and the README.md file. You should repeat steps C through E three times to make at least three commits as you work on your documentation improvements.

	Tasks	Git Commands
A.	Create a branch named documentation on your local repository.	<code>git branch documentation</code>
B.	Switch to the documentation branch.	<code>git checkout documentation</code>
C.	Update your README.md file.	No git command needed (edit the text in your README.md file)
D.	Stage your changes.	<code>git add README.md</code>
E.	Commit your work with a descriptive message.	<code>git commit -m "The Documentaion branch updated the README.md"</code>
F.	Push your commit to your remote repository branch.	<code>git push origin documentation</code>
G.	Switch back to the master branch.	<code>git checkout master</code>

3. Additional Changes to Documentation

In a real world situation, you or other members of your team would likely be making other changes to documentation on the documentation branch. To simulate this follow the tasks below.

	Tasks	Git Commands
A.	Switch to the documentation branch.	<code>git checkout documentation</code>
B.	Make at least 2 additional changes to the documentation - this might be additional changes to the README or changes to the document strings and line comments of the bikeshare file.	<pre>git diff diff --git a/README.md b/README.md index ad0238a..a1f245b 100644 --- a/README.md +++ b/README.md @@ -4,6 +4,9 @@ #### Date created The Date of creation at Github is 15 May 21. +#### Date Updated +The documentation was updated on 15 May 21 at 1400 hrs + #### Project Title Exploring US Bikeshare Data. @@ -29,3 +32,4 @@ numpy #### Credits Udacity +Atom diff --git a/rohit_bikeshare.py b/rohit_bikeshare.py index 73cb7f4..71142ca 100644 --- a/rohit_bikeshare.py</pre>

		<pre> +++ b/rohit_bikeshare.py @@ -1,4 +1,5 @@ -# The project is coded by Rohit +# The code was written by Rohit Kaundal on Atom Text Editor + # Important points to consider # time.sleep() function is used to provide better readability during program execution #Various spaces and next line have been includes to have better readability </pre>
C.	After each change, stage and commit your changes. When you commit your work, you should use a descriptive message of the changes made. Your changes should be small and aligned with your commit message.	<pre> git add README.md git status git commit -m "Updated Documentation README.md file by Documentation Branch" git add rohit_bikeshare.py git status git commit -m "Updated rohit_bikeshare.py file with a starting line" </pre>
D.	Push your changes to the remote repository branch.	git push origin documentation
E.	Switch back to the master branch.	git checkout master
F.	Check the local repository log to see how all the branches have changed.	git log --oneline --graph --all
G.	Go to Github. Notice that you now have two branches available for your project, and when you change branches the README changes.	No git command needed

4. Refactor Code

Now you will be working in your local repository, on the code in your BikeShare python file to make improvements to its efficiency and readability. You should repeat steps C through E three times to make at least three commits as you refactor.

	Tasks	Git Commands
A.	Create a branch named refactoring on your local repository.	<code>git branch refactoring</code>
B.	Switch to the refactoring branch.	<code>git checkout refactoring</code>
C.	Similar to the process you used in making the documentation changes, make 2 or more changes in refactoring your code.	No git command needed (edit the code in your python file)
D.	For each change, stage and commit your work with a descriptive message of the changes made.	<code>git commit -m "Updated README.md by Refactoring"</code> <code>git commit -m "Updated rohit_bikeshare.py by refactoring branch"</code>
E.	Push your commits to your remote repository branch.	<code>git push origin refactoring</code>
F.	Switch back to the master branch.	<code>git checkout master</code>
G.	Check the local repository log to see how all the branches have changed.	<code>git log --graph --all --oneline</code>
H.	Go to GitHub. Notice that you now have 3 branches. Notice how the files change as you move through the branches.	No git command needed

5. Merge Branches

	Tasks	Git Commands
A.	Switch to the master branch.	<code>git checkout master</code>
B.	Pull the changes you and your coworkers might have made in the passing days (in this case, you won't have any updates, but pulling changes is often the first thing you do each day).	<code>git pull origin</code>
C.	Since your changes are all ready to go, merge all the branches into the master. Address any merge conflicts. If you split up your work among your branches correctly, you should have no merge conflicts.	<code>git merge refactoring</code> <code>git merge documentation</code>
D.	You should see a message that shows the changes to the files, insertions, and deletions.	No git command needed
E.	Push the repository to your remote repository.	<code>git push origin</code>
F.	Go to GitHub. Notice that your master branch has all of the changes.	No git command needed