Github Project: Git Commands Documentation Template

Programming for Data Science Nanodegree Program

You will use this template to copy and paste the git commands you used to complete all tasks on your local and remote git repository for this project. This file will serve as your submission for the GitHub project.

Instructions:

1. Make a copy of this Git Commands Documentation template on your Google Drive.
2. Complete the four sections in this document with the appropriate git commands.
3. Download this document as a PDF file.
4. Submit this on the Project Submission page within the Udacity Classroom.

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/suv96/Udacity-Programming-for-Data-Science-With-Python

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.	$ git clone https://github.com/suv96/Udacity-Programming-for-Data-Science-With-Python
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	$ git status
F.	Stage your changes.	$ git add .
G.	Commit your changes with a descriptive message.	$ git commit -m "initial commit"

H.	Push your commit to your remote repository.	$ git push origin master

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.	$ git branch documentation
B.	Switch to the *documentation* branch.	$ git checkout documentation
C.	Update your README.md file.	**No git command needed (edit the text in your README.md file)**
D.	Stage your changes.	$ git add README.md
E.	Commit your work with a descriptive message.	$ git commit -m “Updated info about bikeshare.py file”
F.	Push your commit to your remote repository branch.	$ git push origin documentation
G.	Switch back to the master branch.	$ git checkout master

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.	$ git checkout documentation

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.	$ git diff
		diff --git a/bikeshare.py b/bikeshare.py
		index e38ae84..0000f09 100644
		--- a/bikeshare.py
		+++ b/bikeshare.py
		@@ -142,7 +142,7 @@ def station_stats(df):
		  print("\nThis took %s seconds." % (time.time() - start_time))
		  print('-'*40)
		-
		+# Trip duration stats will display the total time of travel as well as mean travel time
		 def trip_duration_stats(df):
		  """Displays statistics on the total and average trip duration."""
		
		$ git diff
		diff --git a/bikeshare.py b/bikeshare.py
		index b30cc82..e38ae84 100644
		--- a/bikeshare.py
		+++ b/bikeshare.py
		@@ -6,7 +6,7 @@ from statistics import mode
		 CITY_DATA = { 'chicago': 'chicago.csv',
		  'new york city': 'new_york_city.csv',
		  'washington': 'washington.csv' }

		- +# choice class is defined to include the data from the user in the form of choices to laid him towards required data def choice(prompt, choices=('y', 'n')): """Return a valid input from the user given an array of possible answers. """

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.	$ git add .
		$ git commit -m “Added info about trip duration”
		$ git add.
		$ git commit -m ”Added description about the choice class”

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.	$ git branch refactoring
B.	Switch to the *refactoring* branch.	$ git checkout refactoring
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.	$ git commit -m “Add print statement regarding station_stat”
		
		$ git commit -m “Add redirecting message to raw data block”

E.	Push your commits to your remote repository branch.	$ git push origin master
F.	Switch back to the *master* branch.	$ git checkout master
G.	Check the local repository log to see how *all the branches* have changed.	**$ git log --oneline --graph --all**
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.	$ git checkout master
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).	$ git pull origin

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.	$ git merge refactoring
		
		$ git merge documentation

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.	$ git push origin master
F.	Go to GitHub. Notice that your master branch has all of the changes.	**No git command needed**

Submission:

This concludes the project.

- Please review this document to make sure you entered all the required response fields in all four sections.
- Download this document as a PDF file.

- Submit the PDF file on the Project Submission page within the Udacity Classroom.