Github Project: Git Commands Documentation Template

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/mokgonyanetb/pdsnd_github_bikeshare

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/mokgonyanetb/pdsnd_github_bikeshare.git
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 "Adding bikeshare script"
H.	Push your commit to your remote repository.	git push

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 checkout -b documentation
B.	Switch to the *documentation* branch.	git checkout -b 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 "docs:Removing the project instructions from the Readme file"
F.	Push your commit to your remote repository branch.	git push --set-upstream 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 documention
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 status git add .\bikeshare_2.py git commit -m "docs: Updating the Docstrings for the disdisplay_raw_data function"
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 status git add .\README.md git commit -m "docs: Adding a new section (An Interactive Experience) to the README file"
D.	Push your changes to the remote repository branch.	git push
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 --graph --oneline --all --decorate
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 checkout -b refactoring
B.	Switch to the *refactoring* branch.	git checkout -b 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 status git add .\bikeshare_2.py git commit -m "feat: Added a new function for visualizations" git status git add .\bikeshare_2.py git commit -m "refactor: Using mode() to find most common values in time_stats method for faster execution" git status git add .\bikeshare_2.py git commit -m "refactor: Replacing dashes with asteriks (*) to separate output"
E.	Push your commits to your remote repository branch.	git push --set-upstream origin documentation
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 --graph --oneline --all --decorate
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 master
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 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
F.	Go to GitHub. Notice that your master branch has all of the changes.	**No git command needed**