

# 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
<a href="https://github.com/Ralfs-GitHub/pdsnd_github">https://github.com/Ralfs-GitHub/pdsnd_github</a>

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/Ralfs-GitHub/pdsnd_github
B.	Move your bikeshare.py and data files into your local repository.	<b>No git command needed (you can use <code>cp</code> or a GUI)</b>
C.	Create a .gitignore file containing the name of your data file.	<b>No git command needed (you can use <code>touch</code> or a GUI)</b>
D.	List the file names associated with the data files you added to your .gitignore	<b>No git command needed (add the file names into your .gitignore file)</b>
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 the project file bikeshare.py"

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 <i>documentation</i> on your local repository.	\$ git branch documentation
B.	Switch to the <i>documentation</i> branch.	\$ git checkout documentation
C.	Update your README.md file.	<b>No git command needed (edit the text in your README.md file)</b>
D.	Stage your changes.	\$ git add README.md
E.	Commit your work with a descriptive message.	\$ git commit -m "Update of README.md"
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 <i>documentation</i> 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.	<pre>\$ git diff diff --git usage: git diff --no-index [&lt;options&gt;] &lt;path&gt; &lt;path&gt;  Diff output format options -p, --patch      generate patch -s, --no-patch   suppress diff output -u              generate patch -U, --unified[=&lt;n&gt;] generate diffs with &lt;n&gt; lines context -W, --function-context                     generate diffs with &lt;n&gt; lines context --raw           generate the diff in raw format --patch-with-raw synonym for '- p --raw' --patch-with-stat synonym for '-p --stat' --numstat       machine friendly - -stat</pre>

		<p>--shortstat      output only the last line of --stat</p> <p>-X, --dirstat[=&lt;param1,param2&gt;...]      output the distribution of relative amount of changes for each sub-directory</p> <p>--cumulative      synonym for --dirstat=cumulative</p> <p>--dirstat-by-file[=&lt;param1,param2&gt;...]      synonym for --dirstat=files,param1,param2...</p> <p>--check      warn if changes introduce conflict markers or whitespace errors</p> <p>--summary      condensed summary such as creations, renames and mode changes</p> <p>--name-only      show only names of changed files</p> <p>--name-status      show only names and status of changed files</p> <p>--stat[=&lt;width&gt;[,&lt;name-width&gt;[,&lt;count&gt;]]]      generate diffstat</p> <p>--stat-width &lt;width&gt;      generate diffstat with a given width</p> <p>--stat-name-width &lt;width&gt;      generate diffstat with a given name width</p> <p>--stat-graph-width &lt;width&gt;      generate diffstat with a given graph width</p> <p>--stat-count &lt;count&gt;      generate diffstat with limited lines</p> <p>--compact-summary      generate compact summary in diffstat</p> <p>--binary      output a binary diff that can be applied</p> <p>--full-index      show full pre- and post-image object names on the "index" lines</p>
--	--	--

		<p>--color[=&lt;when&gt;] show colored diff</p> <p>--ws-error-highlight &lt;kind&gt; highlight whitespace errors in the 'context', 'old' or 'new' lines in the diff</p> <p>-z do not munge pathnames and use NULs as output field terminators in --raw or --numstat</p> <p>--abbrev[=&lt;n&gt;] use &lt;n&gt; digits to display object names</p> <p>--src-prefix &lt;prefix&gt; show the given source prefix instead of "a/"</p> <p>--dst-prefix &lt;prefix&gt; show the given destination prefix instead of "b/"</p> <p>--line-prefix &lt;prefix&gt; prepend an additional prefix to every line of output</p> <p>--no-prefix do not show any source or destination prefix</p> <p>--inter-hunk-context &lt;n&gt; show context between diff hunks up to the specified number of lines</p> <p>--output-indicator-new &lt;char&gt; specify the character to indicate a new line instead of '+'</p> <p>--output-indicator-old &lt;char&gt; specify the character to indicate an old line instead of '-'</p> <p>--output-indicator-context &lt;char&gt; specify the character to indicate a context instead of ' '</p> <p>Diff rename options</p> <p>-B, --break-rewrites[=&lt;n&gt;[/&lt;m&gt;]] break complete rewrite changes into pairs of delete and create</p> <p>-M, --find-renames[=&lt;n&gt;]</p>
--	--	---

		<p>detect renames</p> <p>-D, --irreversible-delete omit the preimage for deletes</p> <p>-C, --find-copies[=&lt;n&gt;] detect copies</p> <p>--find-copies-harder use unmodified files as source to find copies</p> <p>--no-renames disable rename detection</p> <p>--rename-empty use empty blobs as rename source</p> <p>--follow continue listing the history of a file beyond renames</p> <p>-l &lt;n&gt; prevent rename/copy detection if the number of rename/copy targets exceeds given limit</p> <p>Diff algorithm options</p> <p>--minimal produce the smallest possible diff</p> <p>-w, --ignore-all-space ignore whitespace when comparing lines</p> <p>-b, --ignore-space-change ignore changes in amount of whitespace</p> <p>--ignore-space-at-eol ignore changes in whitespace at EOL</p> <p>--ignore-cr-at-eol ignore carrier- return at the end of line</p> <p>--ignore-blank-lines ignore changes whose lines are all blank</p> <p>-I, --ignore-matching-lines &lt;regex&gt; ignore changes whose all lines match &lt;regex&gt;</p> <p>--indent-heuristic heuristic to shift diff hunk boundaries for easy reading</p>
--	--	--

		<p> --patience      generate diff using the "patience diff" algorithm  --histogram      generate diff using the "histogram diff" algorithm  --diff-algorithm &lt;algorithm&gt;      choose a diff algorithm  --anchored &lt;text&gt;      generate diff using the "anchored diff" algorithm  --word-diff[=&lt;mode&gt;]      show word diff, using &lt;mode&gt; to delimit changed words  --word-diff-regex &lt;regex&gt;      use &lt;regex&gt; to decide what a word is  --color-words[=&lt;regex&gt;]      equivalent to --word-diff=color --word-diff-regex=&lt;regex&gt;  --color-moved[=&lt;mode&gt;]      moved lines of code are colored differently  --color-moved-ws &lt;mode&gt;      how white spaces are ignored in --color-moved </p> <p> Other diff options  --relative[=&lt;prefix&gt;]      when run from subdir, exclude changes outside and show relative paths  -a, --text      treat all files as text  -R      swap two inputs, reverse the diff  --exit-code      exit with 1 if there were differences, 0 otherwise  --quiet      disable all output of the program  --ext-diff      allow an external diff helper to be executed  --textconv      run external text conversion filters when comparing binary files  --ignore-submodules[=&lt;when&gt;] </p>
--	--	--



		<p>ignore changes to submodules in the diff generation</p> <p>--submodule[=&lt;format&gt;] specify how differences in submodules are shown</p> <p>--ita-invisible-in-index hide 'git add -N' entries from the index</p> <p>--ita-visible-in-index treat 'git add -N' entries as real in the index</p> <p>-S &lt;string&gt; look for differences that change the number of occurrences of the specified string</p> <p>-G &lt;regex&gt; look for differences that change the number of occurrences of the specified regex</p> <p>--pickaxe-all show all changes in the changeset with -S or -G</p> <p>--pickaxe-regex treat &lt;string&gt; in -S as extended POSIX regular expression</p> <p>-O &lt;file&gt; control the order in which files appear in the output</p> <p>--rotate-to &lt;path&gt; show the change in the specified path first</p> <p>--skip-to &lt;path&gt; skip the output to the specified path</p> <p>--find-object &lt;object-id&gt; look for differences that change the number of occurrences of the specified object</p> <p>--diff-filter [(A C D M R T U X B)...[*]] select files by diff type</p> <p>--output &lt;file&gt; Output to a specific file</p>
C.	After each change, stage and commit your changes. When you commit your work, you should use a descriptive message of the	<pre>\$ git add <b>README.md</b> \$ git commit -m "<b>Updated the Headings in README.md</b>"</pre>

	changes made. Your changes should be small and aligned with your commit message.	\$ git add <b>README.md</b> \$ git commit -m " <b>Added new content in README.md</b> "
D.	Push your changes to the remote repository branch.	\$ git push origin documentation
E.	Switch back to the <i>master</i> branch.	\$ git checkout master
F.	Check the local repository log to see how <i>all the branches</i> 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.	<b>No git command needed</b>

## 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 <i>refactoring</i> on your local repository.	\$ git checkout -b refactoring
B.	Switch to the <i>refactoring</i> 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.	<b>No git command needed (edit the code in your python file)</b>
D.	For each change, stage and commit your work with a descriptive message of the changes made.	\$ git add bikeshare.py \$ git commit -m " <b>Adding of a comment at the beginning of the code</b> " \$ git add bikeshare.py \$ git commit -m " <b>Adding of another comment at the end of the code</b> "
E.	Push your commits to your remote repository branch.	\$ git push origin refactoring
F.	Switch back to the <i>master</i> branch.	\$ git checkout master
G.	Check the local repository log to see how <i>all the branches</i> 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.	<b>No git command needed</b>
----	--	------------------------------

## 5. Merge Branches

	Tasks	Git Commands
A.	Switch to the <i>master</i> 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.	<b>No git command needed</b>
E.	Push the repository to your remote repository.	\$ git push origin

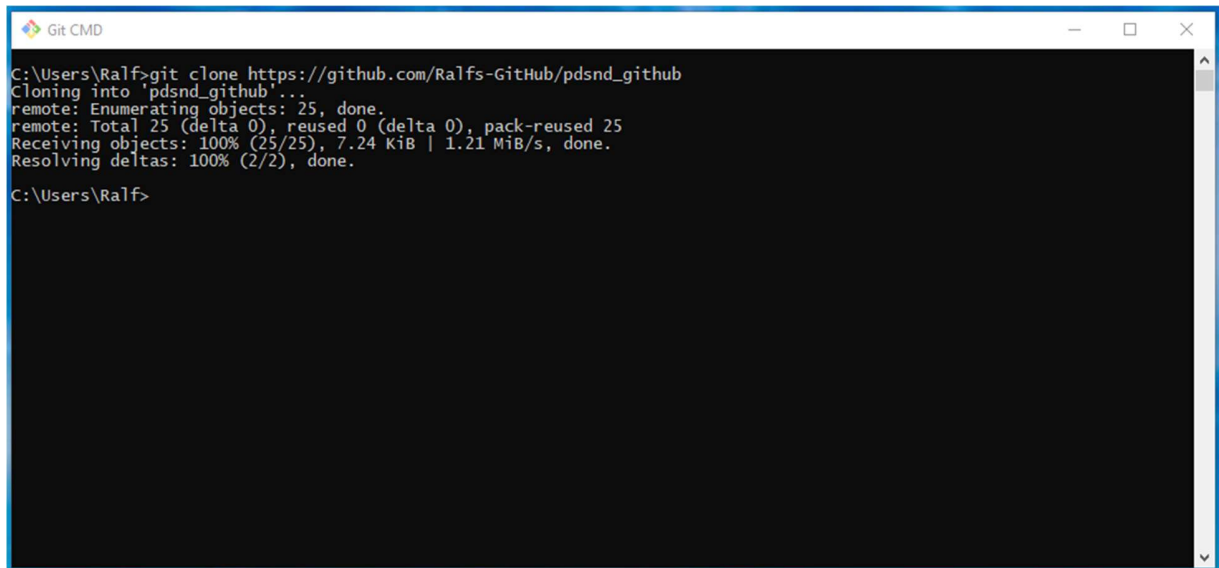
F.	Go to GitHub. Notice that your master branch has all of the changes.	<b>No git command needed</b>
----	--	------------------------------

## 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.

Git clone:



```
Git CMD
C:\Users\Ralf>git clone https://github.com/Ralfs-GitHub/pdsnd_github
Cloning into 'pdsnd_github'...
remote: Enumerating objects: 25, done.
remote: Total 25 (delta 0), reused 0 (delta 0), pack-reused 25
Receiving objects: 100% (25/25), 7.24 KiB | 1.21 MiB/s, done.
Resolving deltas: 100% (2/2), done.
C:\Users\Ralf>
```