Git Intro 2 Activity

==================

A. Form teams

-------------

Form a 2-person team. Try to find someone who uses the same platform as you (e.g., Windows, Linux, etc.). If you can't, that's fine. If you are the odd-person-out, join a team of 2 with permission of your TA.

Make a repo on GitHub.

Complete the exercises below, and fill in your answers in the spaces provided below each question.

Once you have completed all the exercises and filled in this file with your answers,

add this file to your working directory (which you will create in section G) and push it to your team repo.

This file is the deliverable for this lab, so make sure you do not forget to push it at the end!

1. List your team members below.

Carson Hake

RJ Bowen

D. Getting help

---------------

Run the following commands.

git help

git help -ag

git help init

1. What does `git help` do?

Lists all the common commands to be used in various situations

```

2. What does `-ag` cause `git help` to do?

List available subcommands and some concept guides

G. Basic commands

-----------------

Open the terminal. Navigate to your team’s repository directory.

Create a directory inside of your team repository which will act as your submission directory for this lab.

The directory should be named according to the last names of your team members following the format below.

<member1LastName>\_<member2LastName>\_Lab6

Use a plain text editor to create `names.txt` inside the directory you just created.

Put the names of your team in the file. Save and exit.

Run `git status` before and after each of these commands.

git add names.txt

git commit –m “Add our names.”

git log

1. What kind of information does `git status` report?

Information about the working directories that reports staged, unstaged, and untracked files. Also indicates if its ahead or behind the remote branch.

2. What does `git add names.txt` do?

Adds the file “names.txt” to the staging area

3. What does `git commit -m "Add our names."` do?

Adds the staged files into the local repository with the commit message “add our names”.

Use a plain text editor to create the following files:

- `birthdays.txt` - Put your birthdays in this file. (you are not obligated to use real information here)

- `movies.txt` - Put the last movie each of you watched in alphabetical order.

Run `git status` before and after each of these commands.

git add .

git commit Note: Commit will open the vim editor; write a multi-line commit

message, save and quit (press esc and then type :wq).

git log

4. What does `git add .` do? What do you think `.` means?

Adds all untracked and unstaged files to the stage

```

5. What does `git commit` (without -m) do?

Opens up the multiline editor

6. If you want to write a more detailed commit message (which is

good practice) what command would you use?

-m

7. What does `git log do`?

Shows the commit history

```

H. Stage/Cache/Index

--------------------

Do the following:

- Modify `names.txt` so that names are listed in \*Last, First\* format,

one per line.

- Modify `movies.txt` so they are in reverse alphabetical order

by title.

- Create a new file `foods.txt` that contains your favorite foods (one

for each team member).

Run the following commands:

git add names.txt

git status

1. Below write each file name under the state that its changes are

currently in. Compose a definition for each state.

\*\*Staged\*\*

Names.txt

```

\*\*Unstaged\*\*

Movies.txt

```

\*\*Untracked\*\*

Foods.txt

```

1. If you run `git commit` what changes will be committed (\*\*\*DON’T DO IT\*\*\*)?

The names.txt file will be updated to the new order

```

2. What command do you run to stage changes?

Git add

```

3. What command do you run to unstage changes?

Git reset

```

Run the following commands:

git diff

git diff --cached

1. What does `git diff` display?

Shows the differences between the current working directory file and the staged file

```

2. What does `git diff --cached` display?

Shows the differences between the staged file and the commit

```

3. Formulate a sequence of commands to unstage changes to `names.txt`,

and stage the changes to `movies.txt`. Write them here. Execute your commands and

confirm they worked.

Git reset “names.txt”

Git add “movies.txt”

```

4. Edit `movies.txt`, change any one of the movies, and save it. Then

run `git status`. What do you observe? Explain what you think is

going on.

The previously saved “movies.txt” is in the staged area while the newest saved “movies.txt” is in the unstaged part of the process.

```

5. Delete `names.txt`. Then run `git status`. What do you observe?

Explain what you think is going on.

The file is shown in the unstaged deleted section of the status.

6. Rename `movies.txt` to `last-movies`. Run `git status`. Observe

and explain.

the file on the working directory “last-movies.txt” is untracked, the file “movies.txt” is noted as deleted in the not-staged area and the file “movies.txt” on the staged area is shows as modified.

```

7. Formulate a sequence of commands to stage all changes including the

untracked file and commit (with any reasonable message you like).

Execute them.

Git add .

Git Commit -m “updated the repo”

```

8. In git vernacular, `index`, `cache`, and `stage` all refer to the

same thing. What does it hold?

The staging areas holding all of the meta-data for the git repository such as time-stamps, file names, and SHAs of files

```

9. Why have a `stage`? Why not just commit all changes since the last

commit?

Because it allows for a selection of which changes we want to commit. This allows for the continuation of work without effecting the repository.

```

I. Undo

-------

Run the following commands:

git log

git status

git reset --soft "HEAD^"

git log

git status

1. What does `git reset --soft ``"HEAD^" `do?

Undoes the last commit to the repository

Run the following commands:

git commit –m "Redo."

git log

git status

git reset --hard "HEAD^"

git log

git status

1. What does `git reset --hard ``"HEAD^"`` `do?

Make the current branch back to the previous commit

Makes the files in the working tree and index the same as in the previous commit

```

2. What is the difference between `--hard` and `--soft`?

Soft allows for the head to change without changing any of the staged or commited files. Hard forces the entire directory, index, and repo back to a certain commit.

3. What do you think `HEAD` means?

A reference to the currently checked out commit

```

4. What do you think `HEAD^` means?

A reference to the preciously checked out commit

```

5. On Canvas, put two things: 1) a link to your repo (as a comment on the assignment), and 2) submit this document.

6. Push this document, completed, to your repo.

J. Helpful resources

--------------------

- <https://git-scm.com/doc>

- <https://www.atlassian.com/git/tutorials/>

- github-git-cheat-sheet.pdf on class website

K. Copyright and Licensing

--------------------------

Adapted from:

Copyright 2016, Darci Burdge and Stoney Jackson SOME RIGHTS RESERVED

This work is licensed under the Creative Commons Attribution-ShareAlike

4.0 International License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by-sa/4.0/> .