Git Intro 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.

Assign the following roles to the members of your team. If you are in a team of 2, assign the recorder and navigator roles to the same person.

Roles:

- Driver: Creates and maintains a local git repository.
  Olivia Karney
- Navigator: Reads instructions and records answers.
  Sydney Holland

3rd person role (if you have a 3-person team):

- Quality Assurance Person: Ensures instructions are being carried out correctly, answers are clear, and looks for ways for the team to work together more effectively.

You will be rotating roles. Your instructor will let you know when to switch roles. Switching roles requires switching computers. So, plan accordingly.

B. Setup
---

1. Create and share a shared document (Google Doc, Etherpad, etc.) for your team.
2. Copy and paste this document into your shared document.

C. Download and install Git

Download and install Git for your operating system:

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

Starting a terminal:

- **Windows**:

    - git-bash.exe (Linux style commands)

    - git-cmd.exe (Windows style commands)

- **Mac OSX:** Finder -> Applications -> Utilities ->
  Terminal.app

- **Linux:** will vary depending on your window manager

D. Getting help

Run the following commands.

    git help
    git help -ag
    git help init

1. What does `git help` do?

    “git help” gives a list of common commands and explains what each command does.

    ```

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

-ag causes git help to list all the available git subcommands.

    ```

E. Identify yourself

Run the following commands, replacing BOGUS NAME and BOGUS@EMAIL with
your name and email.

    git config --global user.name 'BOGUS NAME'
    git config --global user.email 'BOGUS@EMAIL'

WARNING: The name and email you give will be listed on each commit you make.
If this repository is ever published, your name and email on every commit

will be too. Also, if you are working on a shared computer, you should consider changing this configuration before you walk away.

1. What are these commands doing?

   They assign your name and email to every commit you make in the repository.

   ```

2. What is the purpose of `--global`?

   ```
–global means that it is assigning the values entered into the terminal to the variables to be used across the repository.

   ```

F. Create a repository
---

From the command line, run the following commands.

```
mkdir first_project
cd first_project
```

1. By default any file that starts with `.` is hidden. How do you display a hidden file?

   ```
A file that is hidden can be displayed using the -a flag with the ls command.

   ```

2. Run this command to show the hidden files in the current directory. Are there any?

   ```

Yes. There is a ./ and a ../ file

```

3. Now run the following command.

```
git init
```

4. Check for hidden files again.  What was created by `git init`?

```
Git init creates an empty repository.
```

```

3.  What do you think would happen if you delete `.git`?

```
Deleting .git would delete the empty repository created by git -init.
```

```

4.  Using your observations to the previous questions, answer the following.
    You find an old project on your hard drive. You do not remember if
    it is a under version control. What could you look for to determine
    if the project is being managed using Git?

```
You could use git status to see if the project is being managed using git.
```

G. Basic commands
---

Use a plain text editor to create `names.txt` inside the `first_project`
folder. 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?

```
Git status tells you how many commits there have been to the repository and which files are being tracked.
```

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

```
Git add names.txt initializes the commit to the repository
```

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

```
This commits the file with the committal note “Add our names” describing the change made to the repository.
```

Use a plain text editor to create the following files:

- `birthdays.txt` - Put your birthdays in this file.

- `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?

```
Git add . adds all the files that have not been committed yet.
```

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

```
git commit (without -m) allows us to write multi-line commit messages more easily.
```

6. If you want to write a more detailed commit message (which is good practice) what command would you use?

```
git commit (without -m)
```

7. What does `git log` do?

```
Git log tells you all the commit messages added to the repository.
```

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. Categorize the state of each file by writing each file name under the appropriate state below. 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***)?

```
Only the names.txt file will be committed. The other files will not be updated in the remote repository.
```

```

2. What command do you run to stage changes?

```
Git add

```

3. What command do you run to unstage changes?

```

Git restore –staged

```

Run the following commands:

git diff
git diff --cached

1. What does `git diff` display?

```
Git diff displays the changes made to an unstaged file.

```

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

```
Git diff – cached displays the version history of all the commits to a staged file.

```

3. Formulate a sequence of commands to unstage changes to `names.txt`, and stage the changes to `movies.txt`. Execute your commands and
```

confirm they worked.

```
git restore 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 git status informs me that movies.txt has been modified but has not been committed. This is because we have just modified the movies.txt file to change one of the names of the movies but we haven't committed it to the repository.
```

```

5. Delete `names.txt`. Then run `git status`. What do you observe? Explain what you think is going on.

```
The git status informs us that names.txt has been deleted under changes not staged for commit. This is because we have just deleted names.txt but we have not yet committed the change to the repository.
```

```

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

```
The git status explains that movies.txt has been deleted under changes not staged for commit and that last-movies.txt is an untracked file. This is because while we have renamed movies.txt, because we have not committed the change to the repository, the git status reads this change as movies.txt being deleted to explain its name absence in the repository and a new file last-movies.txt as being untracked.
```

```

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 –all
Git commit
```

```

8. In Git vernacular, `index`, `cache`, and `stage` all refer to the same thing. What does it hold?

```
These terms hold the changes made to the current repository that have not been committed yet.
```

```

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

```
This is to hold the changes made to the repository in the event that they are not committed so that the changes are not lost.
```

```

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?
```

```

Git reset –soft “HEAD^” resets the current branch to the last commit but doesn’t touch the working tree.

```

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?

```

Git reset –hard “HEAD^” resets the current branch and working tree to the last commit.

```

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

```

–hard involves the working tree while –soft does not.

```

3. What do you think `HEAD` means?

```

HEAD means your current working directory.

```

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

```

HEAD^ means everything under your current working directory.
```

```

J. Helpful resources
---

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

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

- <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

K. Copyright and Licensing
---
```