Git Tutorial

Deliverable:

Make a word doc. You will add screenshots to it. When you finish, save your word doc as a pdf and submit the pdf on Camino.

Instructions: Make a copy of a folder with a number of different files in it. Have files with at least 3 different extensions. We will use this copy as our playground for today's lab, so you will probably want to delete it after the lab is over.

If you are already proficient with git, look through all of the commands in this tutorial. If you are confident with all of them, you can do JUST steps 7 and 8.

If you have Windows: Launch git bash; this should have come with git.

If you have Mac: Launch a terminal

1. Configuration

Task: You'll want to start by telling git who you are using the following commands. For the first command, keep the quotes, and replace My Name with your name; first and last with a space between is typical. You'll need to use the quotes any time you're providing something with spaces in it, so git knows to treat it all as one thing; if there are no spaces you can leave them off or keep them, as you prefer. Now type the second thing but replace My_email with your email address. This information will be used to show who made what changes to the code.

```
git config --global user.name "My Name"
```

```
git config --global user.email My_email
```

On your machine instead of two dashes git bash might only want one. If that's the case, just make that change throughout.

Task: Specify the default name given to the primary copy of your code in any repository you create; we will name it main.

```
git config --global init.default branch main
```

Task: Take a screenshot of your terminal/git bash, and add your screenshot to the doc.

2. git init and git status

Task: You can use regular Linux commands in the git bash shell. Use cd to navigate to the folder that you made for this lab. Once in that folder, you will want to have git set up a repository:

```
git init
```

This will create a new subfolder with all the files git needs to track the changes you make to your code. You can't see it but it's there. :)

Task: Type `git status`

This tells you which branch you're currently working on, and files in the current folder that are and aren't being tracked by the repository. This is a great command to use regularly to keep you from getting lost.

Task: Take a screenshot of your terminal/git bash, and add your screenshot to the doc.

3. git add, git commit, git log

At this point, none of the files are being tracked.

Task: Pick one of the files in your directory and run

```
git add yourfilename
```

Then run git status. Your new file should be listed under a heading "Changes to be committed."

Task: Run

```
git commit -m "First commit"
```

This creates an initial save-point for our code. In other words, this creates a snapshot of our code at this moment in time that we can always refer back to in the future.

Task: Type the following command.

```
git log
```

This lists the past "commits" of your code.

Task: Take a screenshot of your terminal/git bash, and add your screenshot to the doc.

4. git ignore, more git add

It is more likely that you will want to add all but a small number of files (for instance, automatically generated files) to the repository. We will now see how to do that.

Task: Create a file in the same folder called `.gitignore` – Note there is nothing before the .

- Type git status. You should see a list of untracked files under the heading "Changes not staged for commit."
- Add the name one of these files a single line in the .gitignore file and save.
- Type git status again. You should see the file is no longer there listed as being untracked.
- You can ignore multiple files at one time. Pick an extension that appears among the files, such as .txt, and add *.yourextension on a line in your gitignore. Type git status . Notice that now all files ending in .yourextension a no longer listed.
- Comments start after # and run to the end of the line. Add a comment to your gitignore.

Task: Take a screenshot of your gitignore and add your screenshot to the doc.

Once you've specified which files to ignore, you can add all of the other files in the directory.

Task: Type

```
git add --all
```

And then

```
git commit -m “Add all files”
```

The -m means we want to include a message describing the commit. You should always do this, so when you’re looking back you can tell what makes this version of the code different from the previous version.

Task: Take a screenshot of your terminal/git bash, and add your screenshot to the doc.

4. git diff

Task: Make some changes to one of your files and save it. Then type

```
git diff
```

This shows us what’s different between current files and most recently committed version. You’ll see + before things that are in one file but not the other, and – before things that a missing from one file but not the other.

Task: Take a screenshot of your terminal/git bash, and add your screenshot to the doc.

5. More on git add

One thing we’ve glossed over so far is that there are always three different “places” where our files reside: working files, staging, and committed. When you modify a file, it automatically moves to working files. The command “git add “ moves a working file to staging.

By default, only files on staging will be committed by the commit command. This gives you the ability to commit some files but not others, by controlling which files are on staging. If you want to move a file from staging back to working files, you can do:

```
git restore --staged file.txt
```

Git will allow you to skip the step of moving files to staging before committing if you want to. To commit all files in working files AND staging, use the -a flag on a commit (this must have just one -):

```
git commit -a -m “description”
```

Task: Create two new files, file2.txt and file3.txt. Add both of them with git add and commit both of them. Then change both of them and make another commit with

```
git commit -a -m “description”
```

Task: Take a screenshot, and add your screenshot to the doc.

6. Reference: More commands

Git also gives you the ability to do some additional actions:

Delete a file:

```
git rm file.txt
```

to “undelete” a file:

```
git restore “file.txt”
```

To rename a file:

```
git mv “oldfilename” “newfilename”
```

If you want to roll back to a previous version, type:

```
git reset commit_hex_id
```

Where commit_hex_id is the hexadecimal id associated with the commit you want to roll back to.

For more information about a git command:

```
git help anycommand
```

7. Branches

As we’ve mentioned, if you want to make big changes to your code, like adding a new feature, it’s good to make a personal copy of the code (or branch) to work in, then merge your changes back into the main branch when you’re done. This will help especially when you are working with other people and it’s possible you might both modify some of the same files.

Task: Create a new branch with the following command:

```
git branch branchname
```

Task: Use the following command to switch the branch to your new branch

```
git switch branchname
```

Once you are on a different branch, if you commit you will not modify the main branch, only your current branch.

Task: Make some changes to a file. Once you’re done with your changes, commit them. Now if you want to add them to the main branch, you will switch back to the main branch and type:

```
git merge -m “message” branchname
```

However, merging does not always go so smoothly. If someone else has committed changes to a file on main that you modified in your branch, you can end up with a conflict. Before you commit your code, you need to resolve that conflict.

Task: While still on the main branch, make some changes to a file and commit those changes, being sure to save and close the file. Then switch to the other branch and make changes to the same file, and commit those changes on that branch, being sure to save and close the file. You should then switch to main and try to merge. You should get an error. Note that you are now in a temporary branch that exists just for you to do the merge (check to see the name of your current branch). Now open the file. You should see something like:

```
<<<<<<< HEAD

Here there will be the text that is in this spot in the main branch

=======

Here there will be the text that is in this spot in the branchname
branch

>>>>>>> branchname
```

Make changes to get the file the way you want it, making sure to get rid of the <<<<<<HEAD, ======, and >>>>>>branchname that git added. Save, and now you will commit, and that will send you back to the main branch.

Task: Once you are happy with your merge, delete the branch using the --d flag:

```
git branch --d branchname
```

Task: Take a screenshot and add it to your word doc.

8. Put your code on github

Task: Create an account on www.github.com if necessary. Log in.

Create a new repository

To put your code on the cloud, do the following; Go to https://github.com/settings/tokens Create a personal access token, check the “repo” scope, and copy it.   Replace the struck-through token below with your token.

git remote add origin ~~http://accesstoken@github.com/username/reponame.git~~

```
git push --all
```

To get changes made on Github back on your local version:

```
git pull
```

Task: Take a screenshot and add it to your word doc.

Task: Convert your word doc to a pdf and submit it on Camino.