Introduction to Git and GitHub

What is Git?

It is an open-source and source control system.

Git is:

1. Free
2. Open source
3. Superfast
4. Scalable

<*>Version Control System: This is a system that records changes to a file or files so that a specific version can be recalled later.

With a version control System, we can track our history and work together.

a. Local Version Control System: Files are copied onto a directory. It is error-prone.

b. Centralized/ Remote Version Control System: A single server contains all the versioned files. And all team members are connected to the central server

to get the latest copy of the code and to share the changes with others [Some examples are subversion and Microsoft team foundation].

The problem with the centralized architecture is the single point of failure. If the server goes offline, we cannot collaborate or save snapshots of

our project, so we have to wait until the server comes back online This is the standard for version control.

c. Distributed Version Control System: Here, every team member has a copy of the project with its history on their machine, so we can save snapshots of our project locally on our machine.

If the central server is offline, we can synchronize our work directly with others [Some examples are git and mercurial]

Operations like branching and merging are slow and painful in other version control systems like subversion, but they are very fast in git.

<*> Checksum: It is impossible to change the content of a file or directory without Github knowing about it. The same applies to losing a file or getting a corrupt file. GitHub will be able to detect it. The mechanism GitHub uses for checksumming is ***SHA-1 hash***. It is a 40-character string (0-9 and a-f).

<*> Git stores everything in its database not by file name but by the hash

value of its contents.

<*> When you do actions in Git, nearly all of them only add data to the Git database. So we can experiment without the danger of severely screwing things up.

<*> The three states:

a. Modified state: Changing files but not yet committed to your database.

b. Staged state: Marking a modified file in the current version to go into your next commit snapshot.

c. Committed state: Saving files safely in your database.

<*> Repository: A storage location for files such as files in a source control system.

What is GitHub?

It is an open-source cloud-based platform.

Why do we need Git and GitHub?

Git is a tool that is used to manage multiple versions of source code edits that are then transferred to files in a Git repository.

GitHub serves as a location for uploading copies of a Git repository.

Since Git and GitHub are primarily about cloud storage, we can access the files/ folders/ documents we push into Git Hub, on any device as long as we have internet connectivity.

Roadmap To Mastering Git

1. On the command line, type the following:
   (*) git config --global user.name “Faisal Alidu”
   (*) git config --global user.email faisalidu08@gmail.com
   (*) git config –global init.defaultBranch main
   The instruction above helped configure the user’s name, email and branch name in this case called main.

   (*) git config --local user.name “Faisal Alidu”
   (*) git config --local user.email faisalidu08@gmail.com
   To use a different name and email, replace global with local.

   (*) git config --list
   For checking the name, email and branch name you currently have.

   (*) git --version
   For checking the git version.

   (*) git help config
   (*) git add -h
   (*) git config -h
   To get help.

To clear everything on the terminal use: CLS or cls

In Git, when a name is more than one word, we enclose it in a double quotation mark, but when it’s a singular word, we don’t enclose the name in a double quotation mark.

Example with a quotation mark

“High tempo.txt” / “Faisal Alidu” / “First commit”

Example without a quotation mark

Index.html / new.txt / leaveMe.pdf / language.docx / savings.xls

Getting A Repository !!!

1. First you need a directory.
   You can open the command line using the directory of the file you wish to track. Else, use cd, to change the directory to your preferred file directory.
   Example:
   cd C:\Users\user\OneDrive - Ashesi University\Desktop\ALL IN ONE\OTHERS\GIT AND GITHUB

2. git init
   At this point, you have initialized an empty git repository.

   i. Go back to the folder where you have saved the file(s).
   ii. Go to view.
   iii. Go to show.
   iv. Check or tick hidden items
   A folder named .git would appear.

Note that we haven't tracked our file(s) so far.
To check whether or not the file(s) is being tracked:

(*) git status

To track a file:

(*) git add fileBeingTracked.txt

Confirm fileBeingTracked.txt is being tracked:

(*) git status

All the files highlighted green are being tracked while those highlighted red are not being tracked.

To track all files:

(*) git add --all

(*) git add .

The last thing we do is save the file by means of commit,
or save changes made to the file.
The commit state comes after the staging state.
The staging state is where we use git add .

To commit a file(s):

(*) git commit -m “first commit: committing all files to the repository”

This is now a page in the history books of Git database.

We can untrack fileBeingTracked.txt:

(*) git rm –cached fileBeingTracked.txt

(*) git status

How To Ignore Some Files

- Go to File Explorer.
- Go to the folder where you saved the file.

In the folder, create a new file and SAVE AS .gitignore

- Right-click an empty space in the folder.
- Select new.
- Go to Text Document.
- Change the extension to .gitignore

With .ignore, we can ignore all files with the same extension. Also, we can ignore files with a common extension but make exceptions for some files with the same extension.

1. **Ignoring all files with the same extension.**
   (*) Open .gitignore in Notepad

   (*) Type the following:

   # ignore all .pdf files.
   *** .pdf**
   In this case, we are ignoring .pdf files, we can do the same for docx, html, txt etc.

   (*) SAVE .gitignore

2. **Ignoring all files with the same extension but making an exception.**
   (*) Open .gitignore in Notepad

   (*) Type the following:

   # ignore all .pdf files but not leaveMe.pdf
   *** .pdf**
   ! leaveMe.pdf
   Again, we are ignoring .pdf files but we leaveMe.pdf will not be ignored, it's the only exception here. It is important to note the order (* .pdf) comes first followed by (! leaveMe.pdf).

   (*) **SAVE** .gitignore

Making Changes to a file.

- Go to the folder where the file is saved.
- Open the file.
- Make the necessary changes.
- Save your work.

On the command line, type:

- git status
  A display message/ output confirms that a file has been modified. This message is highlighted red

- git diff
  The output will display the old version of the file in red text and display the current version in green.

On the command line, we usually use Q or q to exit a view which is not a directory.

After making the changes, the file falls back to the modified state.
But we need to keep the changes, hence updating the file.
Now the file needs to fall back to the staging state.

To stage:

- ✓ git add new.txt
  Let's suppose new.txt is the file we have made changes to.

Assuming we made changes to one file or more, we can stage all at once:

- ✓ git add .
  Note the period (.) that comes after add, most people miss it. Also, note that there is one letter space between add and the period (.)

Or:

- ✓ git add –all

After staging, we can confirm if staging was successful:

- ✓ git status
  The output will display in green text; modified: new.txt

We can remove the stage file from staging:

- ✓ git restore –staged new.txt

After unstaging:

- ✓ git restore new.txt
  This helps to remove the changes we made to new.txt and we go back to the previous version.

Committing a file directly after making changes.
Here, unstaging and restoring is not involved hence, we keep the changes made:

- git commit -a -m “Making subtle changes to a text”
  The string we have enclosed in the double quote is just a description of what happened to the text. We use it to identify what we did with the file before committing it.

Deleting a file:

- git rm savings.xls

Restoring a deleted file:

- git restore savings.xls

Renaming a file, let's say from anthem.txt to pledge.txt:

- git mv anthem.txt pledge.txt

After changing the file name, remember to commit:

- git commit -m “Changed file name from anthem.txt to pledge .txt”

Inspecting Activities Done in Git Database/ Repository

To show all the commits done to a repository:

- git log

To change the recent name or description used when committing at a particular time:

- git log --oneline
  A list of all names used during all the commits done to a particular repository is displayed.

Let's amend or change the description we used when we were committing after changing the file name from anthem.txt to pledge.txt :

- git commit -m "New name for anthem.txt is pledge.txt" --amend

- Press the PgUp key (upward direction key) on your keyboard.

- When you see git log –oneline, press enter
  A new commit log will be displayed. This time, we will see the description used during the amendment, displayed.

To see more details on the changes and commits done to a repository:

- git log -p
  Press Q or q to exit this view.

To learn more on git log:

- ✓ git help log
  The output displays a Manual Page for git log.

Let's use reset:

- git reset 176b6e9
  176b6e9 is a unique number that we see attached to the name or description for each commit. To see this unique code for each commit, use (git log –oneline) to view the commit log, there you can see the code on your **LEFT-HAND SIDE**.

Reset takes the file back to unstaging. Now, let's stage the file:

- git add .

Now, let's commit:

- git commit -m "Updating my file after resetting."

To modify what appears in the history log/ Git database:

- git rebase -I –root
  The output displays an editor and options to modify the way things appear.
  Click, : + X + ENTER, to exit this view.

  Other ways to use git rebase:
  - git rebase –continue
  - git rebase --skip
  - git rebase --abort

Creating A Sub Branch

Let's suppose the current branch is called **main**. We wish to create a sub-branch called **minor**.

This is how we do it:

- git branch minor

To check how many branches there are:

- git branch
  The output displayed is:
  minor
  * main
  main is preceded by *, meaning that the current branch we are working in is the branch called main.

We can switch from main to minor:

- git switch minor
  The output displayed shows that we have switched from main to the branch called minor.
- git branch
  This is to check which branch is active currently. The output displayed is:
  *minor
  main
  Note that minor is preceded by *, which means minor is the active branch currently.

Let's work with the branch called minor.

- Go to new.txt and make changes.
- Open new.txt in Notepad.
- Then SAVE.

- Go to the command line/ terminal.
    - git status
      The output confirms that the file has been modified.
    - git commit -a -m "Changes are done to new.txt"
      We skipped the staging state. Minor is just a sub-branch. Staging and committing have already been done on new.txt in the branch called main.
    - git switch main
      Here, we switched back to the branch called main after we are done making changes to new.txt within the branch called minor.

The changes done in new.txt cannot be seen when we switch back to the main. This is because we made the changes when we were within the branch called minor. Now, we need to merge the two branches so that the changes would apply to the branch called main.

To merge:

- git merge -m "Merge minor and main." minor
  Go to file explorer. Open the file. You will see that the changes now apply to 'main' branch.

At this point, we have no need for the sub-branch hence, we delete it:

- git branch -d minor
  The output displayed is; Deleted minor (was ……).
- git branch
  The output displayed is;
  * main

Let's try another scenario.
Create a new branch. Make changes to new.txt, then before merging, choose a file, say pledge.pdf within 'main' branch and make changes to the file.

While in 'minor' branch:

- Go to new.txt, then make changes to the file.
- Go to the command line and type:
  - git status
  - git commit -a -m "Changes made to new.txt"
  - git switch main

While in 'main' branch:

- Go to pledge.pdf, then make changes to the file.
- Go to the command line and type:
  - git status
  - git commit -a -m "Changes made to pledge.txt"
  - git merge -m "Merge minor and main branch." minor

Making changes to the same file differently in two branches.

Let's say we make changes to new.txt and commit within 'minor' branch. Then we switch to 'main' branch and make changes to new.txt again and then we commit.
Here, we can't simply merge. Automatic merger fails. We need to do the merging ourselves instead of typing the 'git merge' command in the terminal.

Let's merge manually:

- Go to file explorer.
- Locate the folder where you have saved the file.
- Open the file. In this case, new.txt
  You will find some text that says HEAD.
  HEAD is what we have in 'main' branch. Down below is some text from 'minor' branch.
- Make the necessary changes then **SAVE**.
- Go to the terminal and type:
  (*) git commit -a -m "Updating and manual merging."

Pushing Data From Local Storage To Cloud Storage And Vice Versa

We can push our data from our local storage (PC) onto cloud storage (Github).

On Github, we have to create:

- An account (email address).
- A user name.

Anytime we need to push our data from the terminal onto GitHub:

- We need to first create a repository on GitHub
- Then, we need to retrieve (copy) the link to our repository (and paste it on the terminal).
- Then, we can follow the git instructions below.

To obtain the url of your git repository,

Type the following on the terminal:

- git remote get-url origin
  This works only when you have previously pushed your data onto Github. Also, make sure you are working with the correct directory.

On the terminal, type:

(*) git remote add origin https://github.com/Asfaam/Name-of-the-repository.git

Note that Asfaam used above can be different depending on the user. Asfaam is the git the name of a particular user who has a git account. Also, "Name of the repository" is the specific name Asfaam gave to his repository on GitHub.
You can simply get the link to your repository on GitHub. It is not necessary to memorize the format of the link.
"https://github.com/Asfaam/Name-of-the-repository.git" is the link of the repository above.

(*) git branch -M main

(*)git push -u origin main

To make sure you have pushed everything including all the sub-branches as well as ‘main’ branch, try the following code:

(*) git push --all

We can make changes to our file directly on Github but such changes won’t appear on our local repository. We need to manually fetch the changes from cloud storage onto our local storage. On the terminal, type:

(*) git fetch

(*) git merge

We can execute the commands above in one swoop:

(*) git pull

Output displayed is; Already up to date.

Cloning An Existing Repository

We can clone or duplicate data from a repository onto the local pc. This situation is more likely when you access your data on Github on a different device and you wish to transfer the data to your pc for easy access or for further addition of information:

- Go to Github.
- Select your repository.

- Select code.
- Select HTTPS.

- Copy the url of your repository.
- Go to file explorer.

- Create a folder.
- From there, open the terminal.
- Type: git clone https://github.com/Asfaam/Name-of-the-repository.git

Renaming A Branch

To change the name of a branch, say from master to main:

- Go to GitHub.
- Select your repository.
- Go to the branch.
- Select the current branch name “master.”
- Click on view all branches.
- Locate the current branch name “master” on the left side of the default branch.
- On the right side, select the pencil icon, which symbolizes ‘an edit function’.
- Type the new branch name “main”
- Select rename branch.
- Open your terminal.
- Ensure you are in the correct directory.
- Type:
    - git branch -m master main
    - git fetch origin
    - git branch -u origin/main main
    - git remote set-head origin -a