[L-(1)] NOTES By:- "Bushra Nazish"

Page No. Date: / /

"Complete Git and GitHub Tutorial"

* Git and GitHub allows us to maintain the history of the project — At what particular time, which person made which change, where in the project.
"Git" helps us in doing this.

* "GitHub" is a platform, an online website that allows us to host our Git repositories.

* Repository is basically a folder where all the changes are saved.

→ Think this as email service & technology behind it.

→ Ily, other platforms are Github, BigBucket, etc. So, Git is like the version control system like the tech and there are so many online platforms that allow us to host these folders or repositories, i.e., our projects online so that other people around the world can share, look, contribute.

* Downloading GUI clients is not recommended. Because we will heavily focus on the command line (terminal thing).
So, only command line is downloaded from git-scm.com.

Teacher's Signature.....

Now, on cmd or Git Bash —

```
$ mkdir project          // making 'project' folder
$ ls                     // list command
$ cd project             // Going inside project
                            (cd : change directory)
$ ls                     // listing inside project folder
                            (empty now)
```

Now,
say we want to maintain the history of this 'project' using git. Where is this entire history been stored? We create a new file or make some changes in the file or delete the file, these all will be in the history.
So, all of these histories are stored in another folder that git provides us → This is known as a Git Repository and it's name is .git (name of the folder).
These dot files are hidden.

To get this folder where all the history is being saved :

```
$ git init               // Initialise an empty git
                            repository (folder)
$ ls                     // It's hidden & not shown
$ ls -a                  // 'a-tab' means to show all
↳ ./ ../ .git/              the hidden files
```

Teacher's Signature.....

* Any dot file is hidden in macOS and linux.

→ To see what is inside the .git folder:

$ ls .git
↳ HEAD config description hooks/ info/ ...

→ Now, we will be able to maintain the history of the project. Any change made in this project now, git will pick it up.

→ Let's make a change:

Create a new file using 'touch' command.

$ names.txt

→ How do we know what changes have been made in the project that are not currently saved in the history of that project i.e; no one knows these changes have been made. A command for that is:

$ git status

git status says → Hey you have names.txt file that someone has added and is currently untracked; means if we share this project with someone, this git repository or project folder on my GitHub, no one knows that name.txt file is added here

Teacher's Signature.....

by Bushra on 19th December.

So, this is the git status command that tells us that these are currently the changes which are not in the history of your project; no one knows about it.

Now, we want to maintain these changes; we want these ~~proje~~ changes in our project history. People should know that Bushra has done a commit — made a new file or changed or deleted a new file.

=> Comparing with a "Wedding Example" Scenario —

The guests whose photos are not yet taken, are known as: untracked files
Now, placing all these files whose photo has not been taken yet, means whose history is not been saved in the project yet, onto the stage:

$git add .
↓
'dot' means everything in the current project directory that is not currently having it's history (all red files), put all these files in the staging area.
We can also individually add the names:
$ git add names.txt

Teacher's Signature.....

$ git status → Now, names.txt file is changed to green from red.
So, now these people are on the stage and we can click the picture.
So they are permanently saved in the Git history (photo album):—

$ git commit -m "names.txt file added"
↓ provide a message ↓ any message can be provided

↳ 1 file changed, 0 insertions(+), 0 deletions(-)

Now, to check whether there are more people left whose photo is to be clicked - "no".

$ git status
↳ nothing to commit, working tree clean

Since I took the last photograph, is there any other change made in this Project folder? No, there wasn't. So, no new person left whose photo needs to be clicked.

* While starting Git Bash, configuration needed:
$ git config --global user.name "Bushra Nazish"
$ git config --global user.email "bushranazish1998@gmail.com"
Now, checking:
$ git config --global user.name
$ git config --global user.email

Teacher's Signature.....

Now, let's add something in this 'names.txt' file :

$ vi names.txt

Type : INSERT

Kunal Kushwaha
Rahul Rana
Community Classroom

Press Esc
~~Do ctrl + C~~ over INSERT below the screen and Type → :x and press Enter

There is another command to see the contents.

$ cat names.txt → displays whatever is available in 'names.txt' file
↳ Kunal Kushwaha
Rahul Rana
Community Classroom

Since last we took the picture, we modified names.txt. So, to see ~~th~~ it's status :

$ git status
↳ modified : names.txt

Again let's take the picture of this.
Let's ~~take~~ get all these people who have been modified, on the stage :

Teacher's Signature.....

$ git add .
$ git status
↳ modified : names.txt

Now, we have this person on stage. We can click his picture.

Now, to remove them without committing it. Like you got them on the stage by mistake & we are like, we don't want to take your photo; Your photo has already been taken:

$ git restore --staged names.txt
$ git status

They are now removed from the stage.

To get ~~back~~ them back on stage:

$ git add names.txt
$ git status

Again,
$ git commit -m "names.txt files modified"
↳ 1 file changed, 3 insertions (+)

Since we added three lines in it

To see the entire history of project:
(all commits made in history)

$ git log

There are two commits - modified & added.

2) "Making few more commits":

To delete this file:

$ ~~rec~~ rm -rf names.txt
$ git status
↳ deleted: names.txt

Now, let's add it:

$ git add .

$ git commit -m "names.txt file deleted"
↳ 1 file changed, 3 deletions(-)

Now,
$ git log
↳
names.txt file deleted
" " modified
" " added

Peacock
Page No.
Date: / /

Imagine, I deleted by mistake. Basically want to remove the upper 1st commit from history of my project.
Each commit is made on top of other commits. We can navigate above two commits i.e., we want to go back to Tue Dec 21 16:35 when file was added.

So, copy the hash no. of the commit of added one. (whatever commit we copy, the commits before it are removed).

$git reset f0b93a2 - - - - 5536

Now, only one commit remains:

*$ git log
↳ names.txt added

⇒ What happened to all the files that were modified or changed or whatever in the previous commits?
They are now in the unstaged area.

$ git status
↳ deleted: names.txt

This means the people whose photos taken are deleted & they are in the section of people whose photo has not been taken yet.

Teacher's Signature.....

7) Stashing changes:

Now, if all the above changes made in these commits were by mistake or deleted it by mistake, then we can restore i.e., we can put it in the stash area.

Imagine, we don't want to delete these changes right now: like, you are a few people whose photo has not yet taken, Go to the backstage & whenever we want you to come back, we'll get you back!

Suppose, while working on a project, ~~we~~ I can put all my work somewhere out without making a commit (without making a history in the project) and whenever I want it to get back, I get it.

~~$ git stash~~ So, the people who are in the backstage, whom we want put on the stage:

```
$ git add .
$ git status
        ↳
        deleted: names.txt  ⇒ It says this file was deleted (this is a change)
```

Now, I am going to git stash this; i.e., Go ~~back~~ into the backstage, whenever I want all these changes to be made, I will bring you back.

Teacher's Signature.....

Peacock
Page No.
Date: / /

Say, I make another change:

```
$ touch surnames.txt
$ git add .
$ git status
    ↳ renamed: names.txt → surnames.txt
```

Let's make some changes:

```
$ vi surnames.txt
```

Press I for insert
write then ~~Esc :~~
type : w

```
Kushwaha
```

Let's make another file

```
$ touch houses.txt
$ git add .
$ git status
    ↳ renamed: names.txt → houses.txt
      new file: surnames.txt
```

We've done bunch of things.

Now, all these changes we've made, we don't to commit it & also don't want to lose these changes.

[Go Backstage & I will bring you back whenever I want you]

Teacher's Signature.....

$ git stash

$ git status
↳ working tree clean

$ git log
Now, my project ~~was~~ is exactly like it was on Tue Dec 21 16:35 (names.txt file added) - my first names.txt file which was empty :

$ cat names.txt

~~Here, we~~ This was how we can go back to our project.

Now, all these changes which we made, how can we do that :

$ git stash pop → Hey, all the people in the backstage, come to the staging area [Unstage area]
↳
new file: houses.txt
new file: surnames.txt
deleted: names.txt

Now, we can see all the changes that were made previously.

Let's send them again send to ~~backstage~~ backstage as I don't want all these changes ~~[We can say~~

PTO

Teacher's Signature.....

$ git add .
$ git stash

And we can say : Your photo is not being taken ; Go away !

$ git stash clear

So, those changes which we had made which were not committed & in the seperate structure, they are now gone and we can't get them back now.

2) "Starting GitHub" :
We will 2 things.
First, we have our own projects (personal or local) ~~on GitHub~~ and we want to push those on GitHub, we want to share it with other people.

Go to GitHub.com.

Create a new repository : Community/classroom-git and copy it's path

$ git remote add origin https://---.

I did my own "Google"

error : remote origin already exists

$ git remote set-url origin https://---

Teacher's Signature.....

Here, git : git command
remote : means you are working with URLs.
add : adding a new URL.
origin : name of the URL to be added by us (like phone number as we save it with names in contact. Similarly, origin is name of URL.)

* By convention, all the repositories & folders that are in our personal account (or our own account), they have a name called origin.

$ git remote -v → Shows all the URLs attatched to this folder.
https://... (fetch)
https://... (push)

So, now this URL is connected to the folder.
But,
names.txt file can't be shown on our GitHub account now because we've not shared the changes on this URL.
A command for this is "push" to branch "master":

$ git push origin master

Commit done on GitHub.

Teacher's Signature.....

Let's make few more random commits.

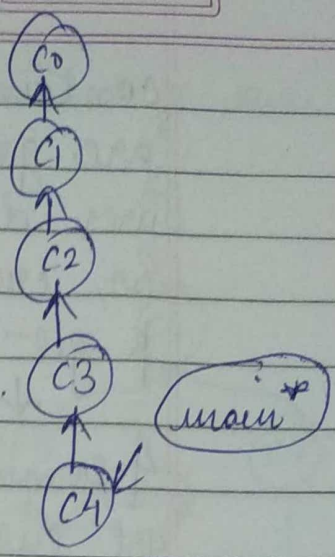
```
$ touch hotel.txt
$ git add .
$ git commit -m "hotel.txt added"
    ↳ 1 file changed, 0 insertions, 0 deletions

$ touch rollno.txt
$ git add .
$ git commit -m "roll number added"

$ vi rollno.txt        → making changes in same file
345687

$ git status
    ↳ modified : rollno.txt
$ git add .
$ git commit -m "roll number ~~added~~ modified"

$ git log              → 4 commits created
    ↳ commit fds ---- (origin/master)
```

⇓

This commit is the latest commit on my remote branch.

* What is a branch ?

eg: $ git commit
$ git commit
$ git commit



This is sort of a branch structure.

Similarly, these are linked to each other in GitBash.

Internally, it's a directed acyclic graph.

It is a branch and by default the name of this branch is called main.
Previously, it was known as master.

* Use of a branch : Whenever creating a new feature or removing a bug, always create a seperate branch.
[Reason → later in this video.]

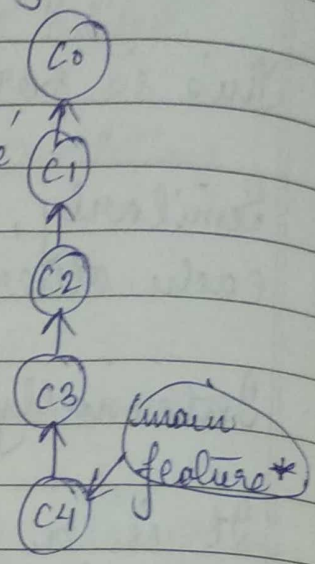
$ ls .git

HEAD ORIG_HEAD config description ---

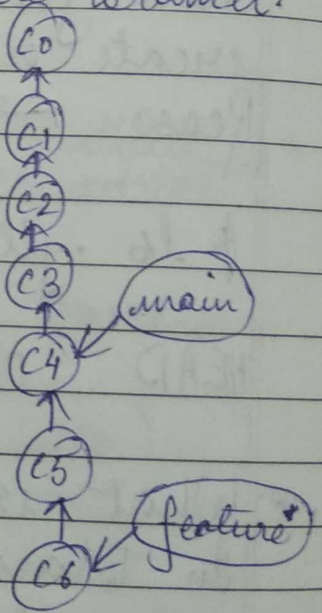
* What is HEAD ?
In branches, never commit on main branch or master branch; because in open-source project 'Main' branch is the default branch used by people (users / developers); as our code which is not finalized yet might

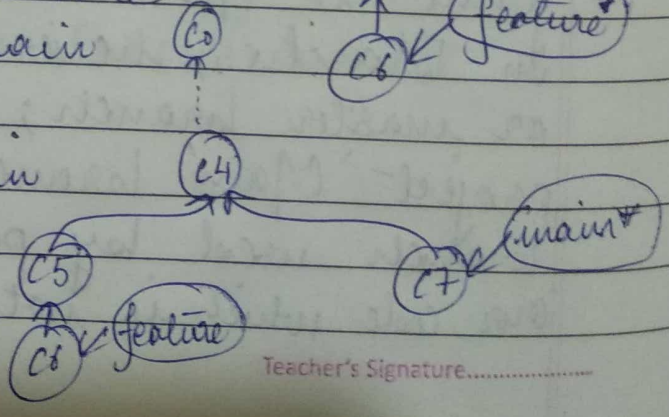
Teacher's Signature.....

contain some errors → these all codes are gone to a seperate branch so the user doesn't get affected.
So, create new branch say feature
$ git branch feature
↓
4 command '$ git checkout feature' got executed; and the star is now on feature.
This means that my head is pointing now to the feature branch.
head is just a pointer that says all the new commits made will be added to the head.
Currently, the head is on feature branch so all the commits which I will be making, will be on the feature branch.
$ git commit
$ git commit



Another thing, so many people contribute to the code base. So, while we are working on our issues someone else gets their code added on the main branch:
$ git checkout main
$ git commit

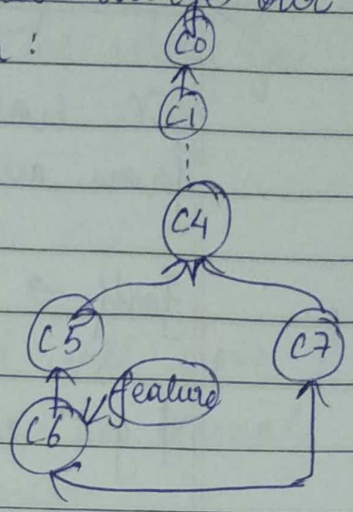




Teacher's Signature.....

Now, our code is finalised. So to merge our feature branch to main branch:
$ git merge feature

$ git checkout C5
$ git checkout C8
$ git checkout main
Now, people can see the code on C5 and C6.



This merging happens via a Pull Request (PR).

⇒ To open our repository on Git Bash:
$ git remote -v

To push the new commits on Master branch (above 3 commits of (origin/master → leaving it)):
$ git push origin master
(origin → to which URL, master → to which branch)

4 commits & 3 code → on GitHub.

$ git log
↳ HEAD → master, origin/master ← here
and
Head is on the master branch & the URL added is also here.
This means, no commits are above it that are extra which are not already on the URL

Teacher's Signature.....

2) "Working with Existing Projects" :

We have to copy the contents or fork from another's to own account.

Fork → Open Code → Copy link

```
$ git clone https://----
$ cd commclassroomOP
$ cat README.md
```

reading the content in it

* From where we've forked this project, is known as Upstream URL by convention.

```
$ git remote add upstream https://---
$ git remote -v
    origin   https://---   (fetch) (push)
    upstream https://---   (fetch) (push)
```

origin is my personal

from where I've forked it

→ Making few commits :

```
$ vi README.md
$ git status
    ↳ modified: README.md
```

Teacher's Signature.....

** Never commit on the main branch. Always, create a new branch for whatever work you are doing:

$ git branch kunal

$ git checkout kunal → head will now come on kunal branch (main change to kunal)

$ git add.

$ git commit -m "Bushra added a message"

$ git log

⇒ "What is a Pull Request"?

Now, whatever code I've on this kunal branch, of my own account, Please merge that into the main's project main branch.

* We should never commit on master branch and ~~make~~ always create such branches: If you are working on a long or have a new feature or thing, make sure to open a new branch and Pull request for that.

Pushing this branch to origin as we can't push to upstream (as we don't have access)

[Pressed q]

$ git push origin kunal

(Typed 'reset' → Press Enter) to kill.

Teacher's Signature.....

Let's say we make some other changes:
$ touch names.txt
$ git add .
$ git commit -m "names.txt added"
$ git push origin kunal
(Head is on kunal branch)

Now, it will not allow me to make a new pull request. It will add these commits to 'commit section' on GitHub only (from 1 to 2 commits).
This is the reason we should never commit on main branch.
Imagine we are working on 10 projects or 10 features and for every feature, we create just one PR, it will be very difficult to review our code and have the discussion; all the 10 different things on just 1 PR.
That is why, for every new feature or new bug we are working on, create a new Pull Request.
And from this example, we can see if a branch already has a Pull Request associated with it, it will not allow you to Pull request; all the commits will be added to that PR only.
So, 1 pull request means 1 branch. [simply]
1 branch can only open 1 PR.
So, any particular new thing you are working on, create a new branch in your local folder, make a Pull Request from that.

Teacher's Signature.....

7) "Removing a commit from the pull request by force pushing to it":

We are removing the extra names.txt added commit from GitHub:

$ git status

$ git log
↳ commit --- names.txt added } → We've to remove this
commit a4---9a0 ~~Kf~~ Kunal added a message

So, Copy this

$ git reset a4---.9a0
$ git status
} Now, 'names.txt' will be unstaged.

$ git add .
$ git stash → It will go back to ~~the~~ some stash area
$ git log → Now, it's gone
↳ commit --- Kunal added a message

names.txt file deleted & also removed from commit history

$ git push origin Kunal -f → When I push this, I would have to force push this because the online repository contains the commit that my repository does not.

 PTO

And we know commits are interlinked. On refreshing, names. 1 nt added' commit gone on GitHub and 'Files changed' is also 1 only now.

2) "Merging a Pull Request":

Confirm merge can be done by authorized user of main project.
(Now, the changes I made in BusloraNazia. buslora branch (BusloraNazia: buslora), they will now be represented in the main project. [in README.md]

2) "Making forked project even wiser main project":

The fork is not updated on my account. If I can't directly change Kunal-Kushwaha account same it can't also change my account.
They are 2 ways to do it.

Click Fetch upstream button —
The branch is 2 commits behind the main branch.
See commit section: it contains 2 extra commit that my main branch does not contain; b'coz the extra commit I had was in the Kunal branch & not in the main branch.

Teacher's Signature.....

This is the reason, we can encounter as: some other person merged their code in the main branch; I want to look at that code sample as well in my own fork.
How do I make sure that the main branch of the upstream & my fork's main branch is always maintained.
There are few commands (manually):

$ git checkout main → Switched to 'main' branch

$ git status
$ git log
} Here in main branch of my fork, I have only 1 commit but in reality in upstream it has 3 commits

First, we need to fetch all these commits and change these:
$ git fetch --all --prune
(prune means the ones who are deleted are also fetched)
Now, I have fetched all the changes.

Second step, we reset it (reset the main branch of my origin to the main branch of upstream):
$ git reset --hard upstream/main

$ git log → Now 3 commits both in GitBash & GitHub

Teacher's Signature.....

Now, my folder that I've on my local system, the main branch of that is exactly same as the one I've in the upstream.
The fork on my account is not updated because I have to push my changes:

$ git push origin main

* If 4 commits in upstream & 3 commits in mine:
$ git pull upstream main

$ git log

"git pull internally does same as git fetch"

* So in the JAVA codebase, whenever you want to pull the code that I'm submit, you will do —
git pull upstream main
So, fork & upstream is same

2) "Squashing commits":

If we've a lot of commits & we are working on and we want to merge all these.

Teacher's Signature.....

$ git branch temp → creating new branch
$ git checkout temp

(New branch always gets created from your head. So make sure while creating it, your main or master branch is up-to-date.

Squashing commits in one single commit:

Creating random files →
$ touch 1
$ git add . ; git commit -m "1"
$ touch 2
$ git add . ; git commit -m "2"
$ touch 3
$ git add . ; git commit -m "3"
$ touch 4
$ git add . ; git commit -m "4"
$ git log → contains all these commits

(Now ~~s~~merging : on reset last commit, all the commits will be unstaged (on the stage area)
Copy last commit, then

$ git rebase -i 4f3 - - - 152

↓
interactive environment

Teacher's Signature.....

Now, we can see that all the commits above it, we can either pick or squash:

```
pick  bcfc196  1
pick  9f...    2   →  s 9f... 2
pick  87       3   →  s 87... 3
pick  90       4
```

squash (s) means, whichever one which is listed as pick, merge it into the previous commit.
Above these 's', whatever 'pick' you have, merge your commit on that.

To exit out of it: Esc + ':x'

This will now allow you to create a message, we are adding a new msg: "commits merged" (on deleting 1 to 4)

```
$ git log
  ↳ commits merged   (all 1,2,3,4 are gone & merged into a single commit)
```

Now, again "git push origin temp" → all files will be available on GitHub.
But, I want to delete them:

```
$ git log
  ↳ commits merged
    commit 49f...
```

copy this

Teacher's Signature.....

$ git reset --hard 49f...

use it with caution : it will reset hard

$ git log

3) "Merge Conflicts and how to resolve them"

If two people changes same line no. 3 (say), then git asks us to keep which change.