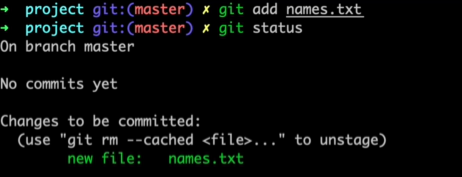
GIT and GitHub

Youtube Link: https://www.youtube.com/watch?v=apGV9Kg7ics

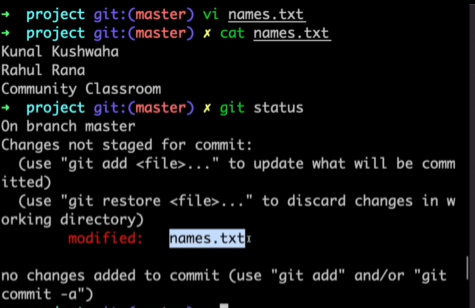
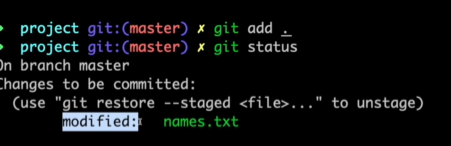
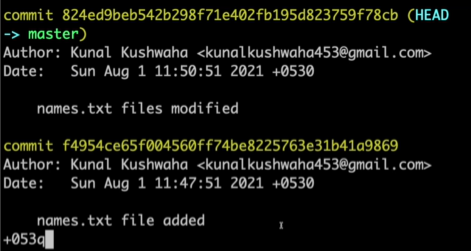
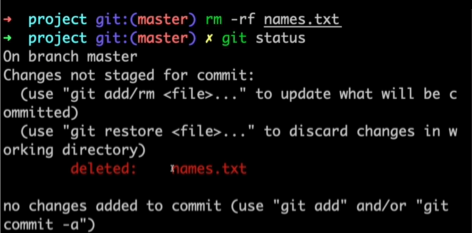
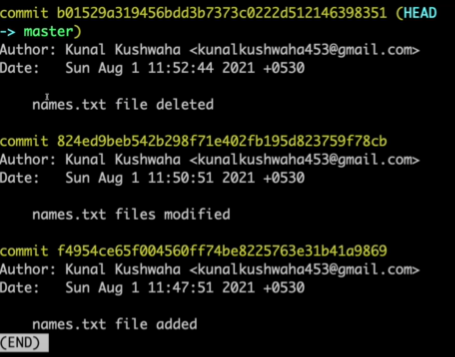
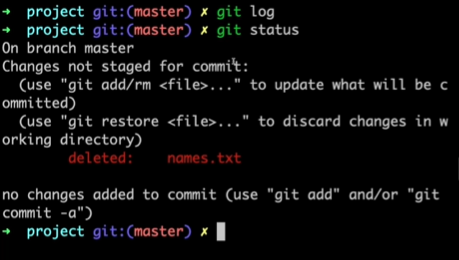
* GitHub is a platform or place in online where the project files can be stored and Git maintains the History of changes made to it along with the line number and the person name who made changes to the project.
* Repository is a folder where the changes have been saved to the project.
* **ls**- shows the list of files or folders in the path
* **mkdir** – create a folder in the path specified.
* **Cd folder name** – change the directory.
* **Git init** (in the GIT CMD) – it creates a file in the same path that stores the history of the changes being made.
* **notepad names.txt** – create a file in the folder with the name names.txt
* **type names**.txt – this command shows the content in the files names.txt
* **git status** – this command shows the status of the applications in the project folder. With the message as untracked files names.txt.

To add the file to the git need to write a command **git add names.txt**

* Now the git status command shows the file names.txt as tracking



* To Commit the changes made in the folder to the Git write command   
  **git commit -m “names.txt file added”.**-m is used add a message while committing the changes  **A computer screen with text

  Description automatically generated**
* **vi names.txt** – open the file names.txt and we add data into the file.
* **cat names.txt –** shows the content of the file in the CMD.
* After adding the content to the files we have the commit the changes to the git so that the git will be aware of the changes made.  
  git status command will show this. Which states that the files names.txt has been modified.  
  ****
* git add. and then git status  
  ****
* **git restore –staged names.txt** – This restores the last file that has been added.
* Now commit the changes to the git with **git commit -m “names.txt file modified”**
* **git log** – command used to check the history of all the changes made to the Project folder.  
  ****
* **rm -rf names.txt -**  removes the files names.txt from Git. 
* Now check the git status which shows the message as files names.txt was deleted.   
  Now give add and commit the changes that you have made to the project folder.  
  Now give check the history of the project folder by checking the logs   
  git log  
  ****
* Suppose if something bad has been committed and we want to restore the files to the files available on Sun AUG 1 11:47 then copy the commit ID  
  **git reset \*commit id\* **This removes all the changes that have been made after Aug 1 11:47
* Now give **git staus** which shows only one commit. 
* **git stash:** The git stash command saves your uncommitted changes (both staged and unstaged), and then reverts them from your working copy. This is useful when you need to temporarily save your work and switch branches, or if you need to make a quick change to another branch without losing your current changes.  
    
  **git stash pop:** will restore your stashed changes.  
  **git stash clear:** This helps in clearing(completely deleted) the stash memory

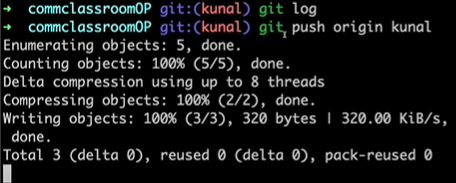
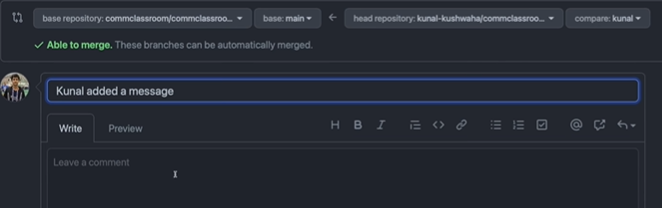
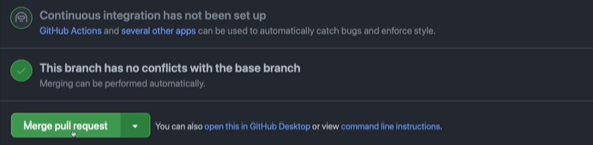
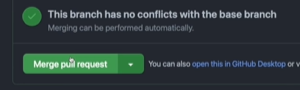
**GitHub –** Create a repository in GitHub and then add this to the Git so that it keeps track of the history of the changes that are being made on the GitHub repository.

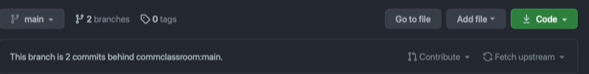
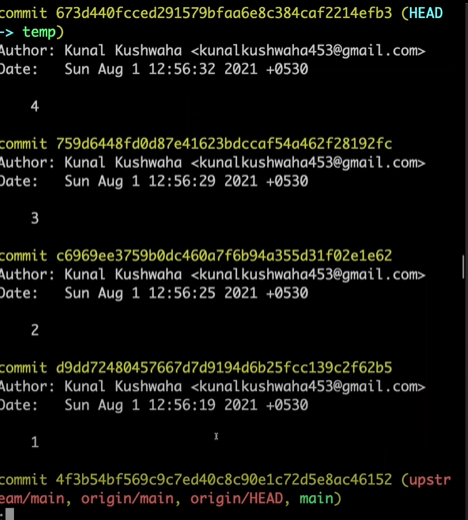
* Use command **git remote add origin \*URL of GITHUB repo\*  
  **origin is the alias name given to the GitHub URL. The alias name can be anything that makes sense.   
    
  **git remote -v**: This shows us all the URL attached to the folder.
* **Git push origin master:** This command helps in pushing all the files to the URL.  
  **origin** is the alias name   
  **master** is the branch into which you want to push those changes
* Suppose we have a situation where we need to contribute to some project for which you don’t have access to the main Account. In order to resolve this issue we use the **fork** functionality. We cannot make any changes to the others account so we have to fork it  
    
  This helps in forking the project to your account so that you can make all the changes in to the project.  
  **A screenshot of a computer

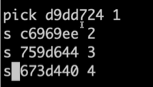
  Description automatically generated**
* Now go to Git and use the command   
  **git clone \*URL\* -**  This downloads all the files available in the Project.
* The URL where the original code is available before forking is called as the Upstream URL.  
  **git remote add upstream** **\*URL\*** - This must contain the URL of original codebase.  
  **A computer screen with text

  Description automatically generated**

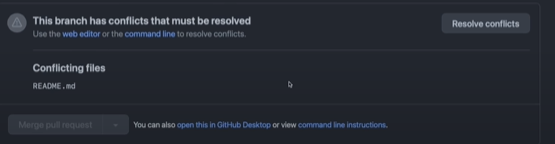
**PULL Request:**

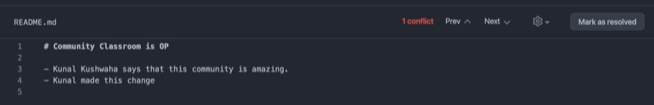
* We can create as many branches as required for very fix that we are doing on the GitHub can have a separate branch.  
  **git branch kunal** is the command to create a new branch.  
  **git checkout kunal** – This helps the HEAD to move to the new branch kunal  
  Once the new branch is created the HEAD will now be pointing to the new branch which results in the changes that are being made will be reflected on the new one rather than the main branch.  
  ****
* Let's take an example suppose I have forked a project from Ashok, And I have found a document missing in the Project. To add the document or code to the project in GitHub we need to create a push request with a new branch created in our own branch with the command  
  **git push origin kunal –** push the code changes to our account.  
  ****  
  We will automatically get notified in the GitHub UI asked for compart and pull request.  
  ****  
  Once clicked on it we will be asked to communicate with Ashok as they will review the code and approve for merging the code to the main.  
  ****
* Now create a Pull Request from the bottom of the conversation and it asks you to merge the code
* ***For Every change that you are working on the project try creating a new branch so that the changes will not mess up.***
* Suppose we have made changes on the two files and made commits to the branch and we got a requirement saying that the last commit needs to be removed then copy the commit id and then   
  git reset \*commit id\*  
  git status  
  git add .  
  git stash  
  Now force push the changes that are made on the repository using command   
  git push origin -f
* Now merge the changes made on kunal fork branch to the main project click on Merge -> confirm Merge.  
  A screenshot of a phone

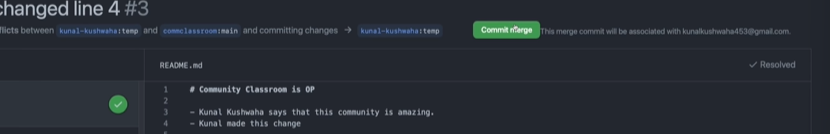
  Description automatically generated  
  With this, the merging will be completed with the changes made on the kunal’s branch
* Once the merging has been completes then the changes will be reflected on the main project but the same will not be reflected on kunal’s forked branch.  
    
  Let’s suppose we are working on one change and the other person has made the changes and merged the code to the main branch then the forked branch is not updated we have some otpions to either sync the main and fork branch with the commands or else with the option available as **fetch upstream.** ****  
    
  Manual commands to Sync the main and forked branch  
  -git checkout main  
  - git fetch –all –prune  
  - git reset –hard upstream  
  -git push origin main
* We can also use the pull command to pull the changes from the main branch to the Kunal’s branch.  
  - git pull upstream main   
  -git push origin main
* Squashing Commits-Squashing allows you to combine multiple commits in your branch's history into a single commit.  
  Suppose we have a situation on which we have made several changes to the project and committed all the changes made. This Creates a lot of commits in the Git.  
    
  git -rebase -I \*commitid\*  
  A black background with white text

  Description automatically generated  
  Now change the pick to Squash based on the commits that need to be merged  
    
  In the above case, the commits with 2,3 and 4 will be merged into pick 1   
  Add a message to the commit that you have made by pressing-> esc ->:x
* To remove the commit that was made from the stash use   
  **- git reset –hard \*commitid\* :** do it with caution.

***Merge Conflicts***:

* We used to see these Merge conflicts in a case where two persons have made changes to the same line in the Project file  
    
  For example: If Lokesh is working on line 4 in the README file and Ram is also making the changes on the same line in a different branch both Lokesh and Ram have committed their changes to GitHub and after pushing the changes they receive the Merge Conflicts.  
    
  

Now click on the Resolve Conflicts button in GitHub and then manually remove the lines that are not required and then click on mark as resolved.  


Now commit the merge to the project then you can see these changes on the main Project.

* **git merge Kunal:** This command is used to merge the code changes to the GitHub repository so that the end user can use the features added to the project.