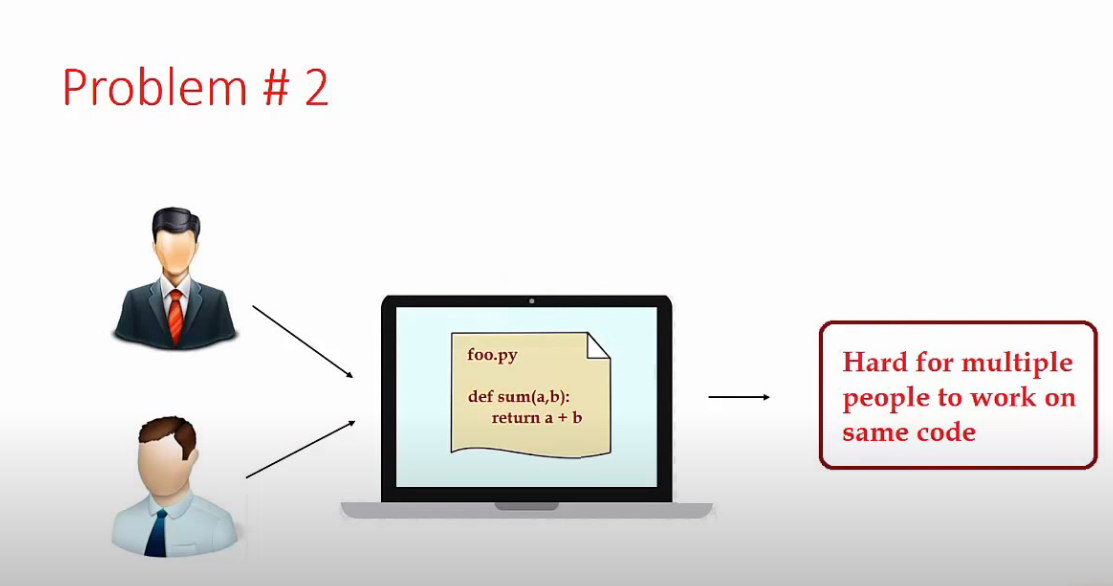
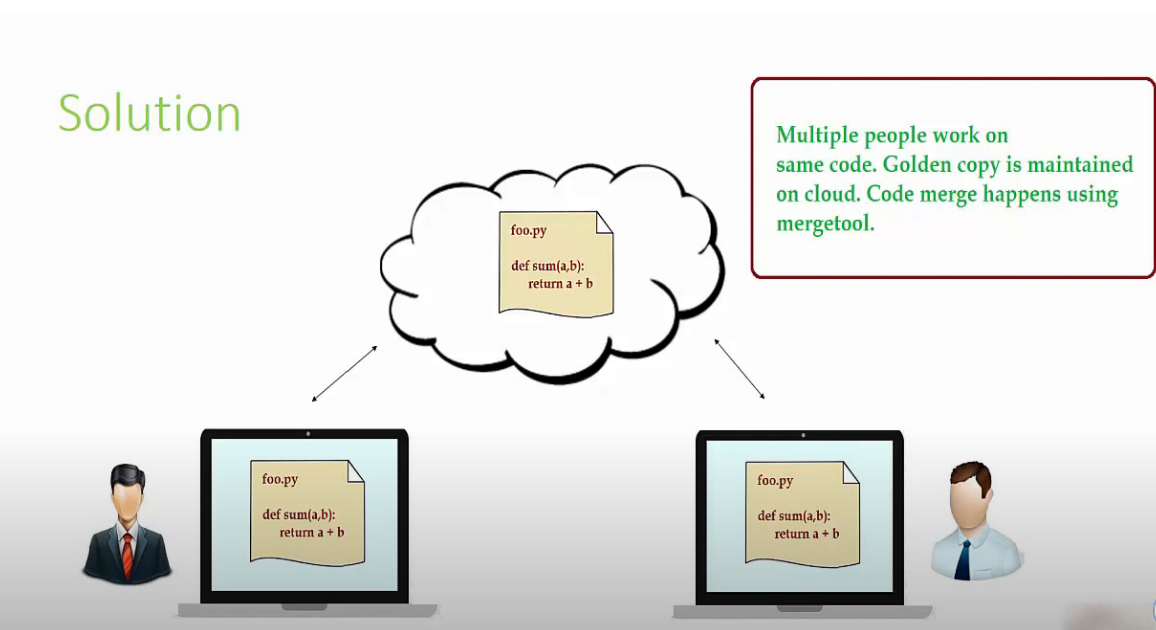
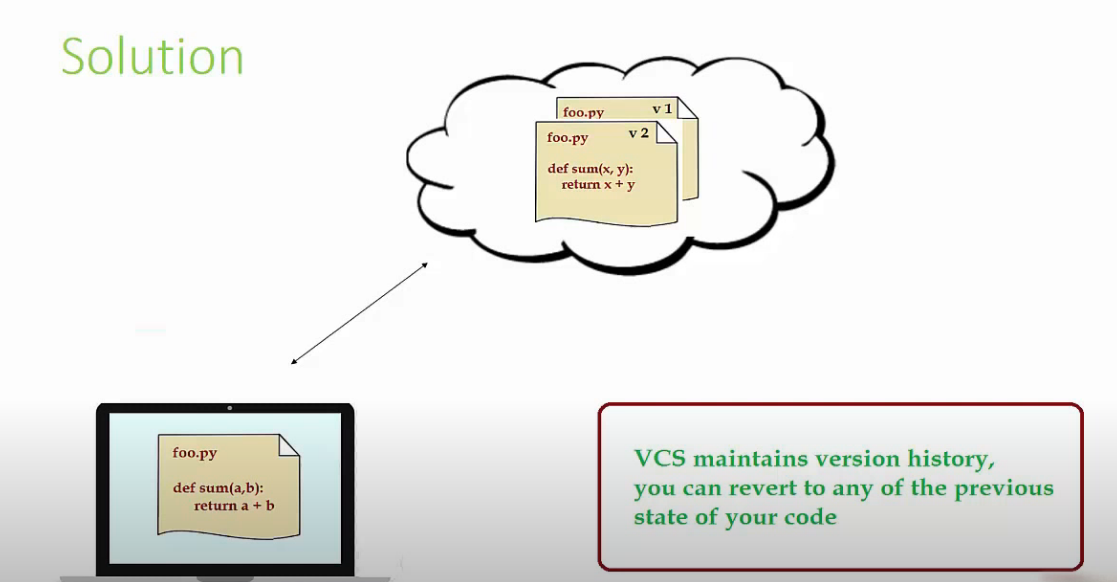
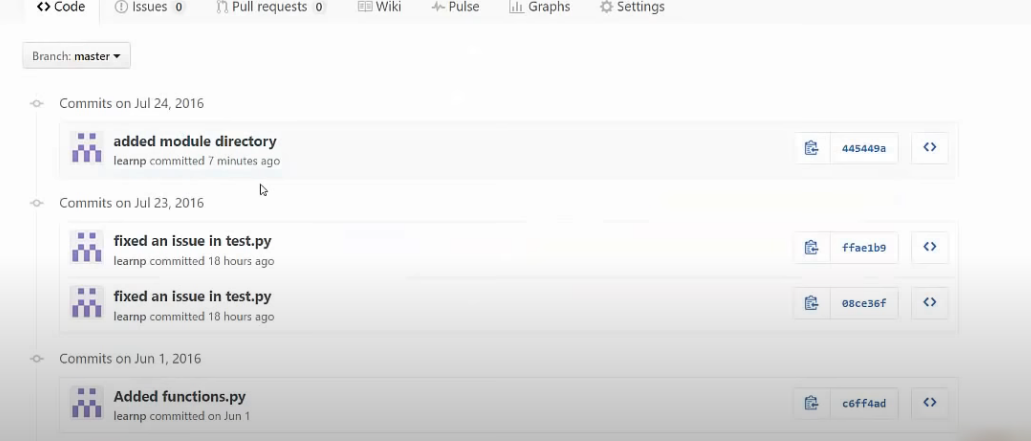
**Git:**

* Git is Nothing but a Version Control System
* Version Control System Solves the Following Problems.
* Graphical user interface

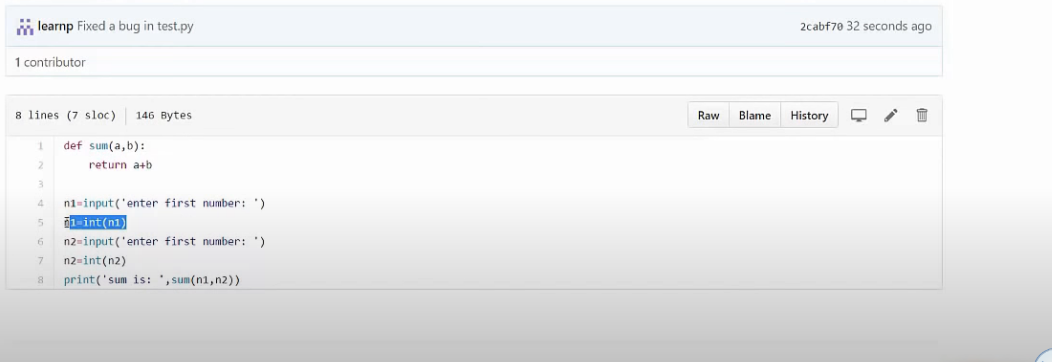
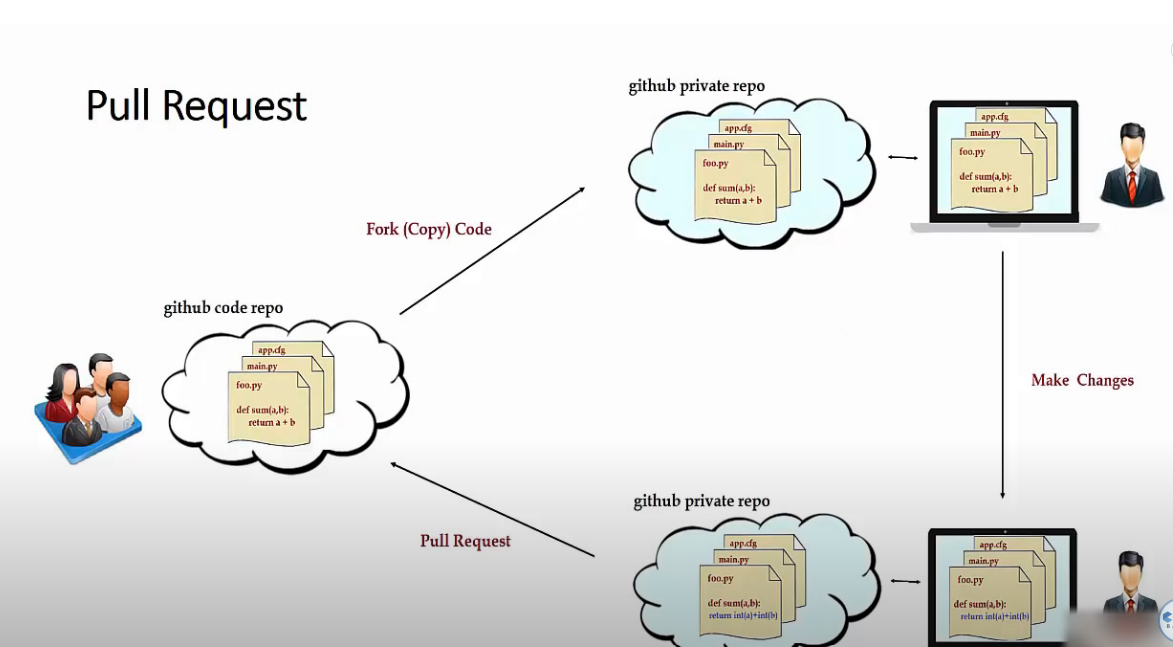
  Description automatically generated with medium confidence
* Problem 1: Suppose if You have a file on Your System. Suddenly, the computer gets crashed in that case all the Code gets Lost.
* 
* Problem 2:If multiple people want to contribute on that code , that is on one Computer.
* Diagram

  Description automatically generated
* Problem 3: Suppose you made some changes, there arises a bug and you want to fix it immediately. Either you can fix it or come back to previous version.
* **Version Control System** Solves all these problems.
* Problem 1 solution: In Version Control system code is Stored in Cloud. Even if computer crashes we can directly get the code from cloud.
* 
* Problem 2 solution: As the Code is Stored in the cloud, you can sync it to your computer and can easily make changes.
* 
* Problem 3: In Version Control System, we have a history of all the changes made. So you can go back to previous versions of code.
* 

**GITHUB:**

* GitHub is a website that allows you to store your code on the cloud. It is not only cloud storage but complete version control powered by GIT.
* In Git Hub we have a commit history, using which we can go back to the previous code changes.
* ****
* Basically, we push the code to github. We have the same code and same list of files on local system github cloud storage.
* **Graphical user interface, text, application

  Description automatically generated**
* Whatever changes we do, and then commit and push to the github we can see all the changes in the following way.
* **Graphical user interface, application, email, Teams

  Description automatically generated**
* All The green lines denotes , the changes made in the file.
* The same code changes will replace the old code in the file.
* ****
* **Another important feature of Git is that forking.(pull request)**
* Pull request allows you to collaborate with other projects
* Suppose there are bunch of people who are working in Inventory management and the code is stored in github repo. Lets say X wants to make Changes in the code. So first he will **fork(copy)** the code in to his github private repo. Then he will sync up the code to his local computer make changes and push his to his own github repo. After that once he is done making his changes and he tested the code he will send the request to the owner to pull his changes to their repository. They will do code review and if it is good then they will accept the request,
* ****

**How To Install Git:**

* **Go to** [**https://git-scm.com/downloads**](https://git-scm.com/downloads)
* **Download and install it.**

**GIT COMMANDS:**

* **git –version:** To check the version of git.
* **How to Create a repository on GitHub:**
  + Go to GitHub website
  + Create an account
  + On the top right corner click + icon
  + Select New Repository
  + Fill all the details.
  + Create Repository.

Graphical user interface, text, application, email

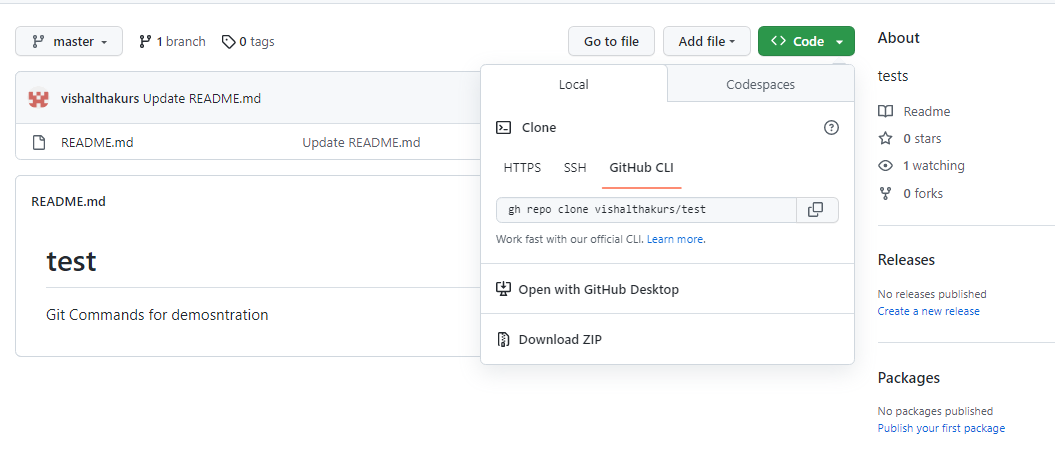
Description automatically generated

* After Creating the repository it will look like the following.

Graphical user interface, text, application, email

Description automatically generated

1. **How to sync the repo to local computer**
   1. Go to Git bash.
   2. Just clone the repository from github (i.e cloning the repo means copying the repo from github to local computer)
   3. go to code and copy the link



* 1. Type the following command and paste the link In the git bash:
  2. **git clone https://github.com/vishalthakurs/test.git**

Text

Description automatically generated

* 1. Open the project in any one of the code editor. Here using Pycharm.

1. Git Basic Comands
   1. **git status**:
      1. To check which files are to be added on to the staging area.
   2. **git add <filename>** :
      1. To Add the Files in to the Staging Area. Later on Staging area will be used to commit the files.
   3. **git add .** :
      1. To add ALL the files present in the directory into staging area at once
   4. **git commit -m “message”** :
      1. To commit the files into the local version database and ready to pass to cloud. We still not have pushed the changes to remote server(github).
   5. **git push:**
      1. To push the changes on to the git hub(Only while using git bash)
   6. **git remote add origin <gitrepourl>**: To add the website on which you want to push the changes.
   7. **git push REMOTE-NAME BRANCH-NAME:**
      1. The git push command takes two arguments:
      2. A remote name, for example, origin
      3. A branch name, for example, main
   8. **git remote:** 
      1. To see the remote name.
      2. git remote -v :to see with the url

Diagram

Description automatically generated

UNDOING/REVERTING/RESETTING :

1. **git checkout -- filename:**
   1. undo uncommited changes.
   2. Without adding into the staging area.
2. **git revert commitid:**
   1. undo committed changes
   2. It means after adding the file into staging area and after performing commit if we want undo the change.
   3. Directly commit the change
   4. If you don’t want to directly commit the change then Use
      1. git revert -n commitid
      2. it will just throw the file into staging area after that you need to commit it
3. **git reset –hard commitid:**
   1. It will reset and all the revert changes will be gone.
   2. It is very powerful and it will take you to that timestap
4. **git log: To check the history of changes**

**BRANCHES CONCEPT IN GIT:**

1. Creating a Branch
2. Merging a Branch
3. Deleting a Branch

Why do we need Branches?

* We Create branches so that all the experimental changes should be in that branch and original code should be in master branch. After making sure the code is good enough we merge the branch.
* It allows you to create parallel branches of code.
* Master is the default branch git will give you

1.Creating a Branch:

1. git branch:
   1. It is used to see how many branches are present.
2. git branch <branchname>:
   1. It is used to create a branch
      1. Ex: git branch thirsty

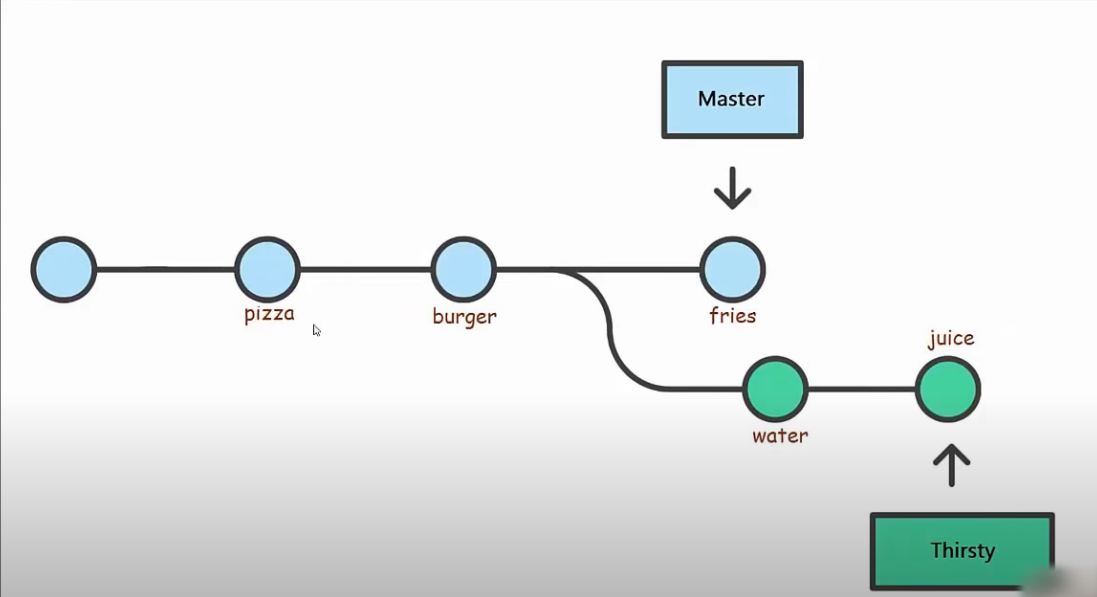


1. git checkout branchname:
   1. To make a branch Active we use the above command.
      1. Ex: git checkout thirsty

Text

Description automatically generated

1. Now after making some changes in the file we add and commit it. Now, if we checkout again to master branch we notice that all the changes have gone .Because, those changes are experimental in nature and available only in thirsty branch.
2. When we switch back to thirsty we will notice all the changes in the file.
3. If we switch to master branch and make code changes , we wont be able to see in thirsty branch.
4. To Easily Understand the Concept see the below pictorial view.



1. Here we can see, that till burger we were on master branch, then we created thirsty branch.
   1. So all the changes till burger will available to Thirsty branch
   2. The chain broken and all the changes are maid to Thirsty Branch wont be available to master branch.
   3. Similarly, all the changes made to Master wont be available to Thirsty.
   4. All the blue colour circles are commit made to master branch.
   5. All the green colour circles are commit made to Thirsty Branch.

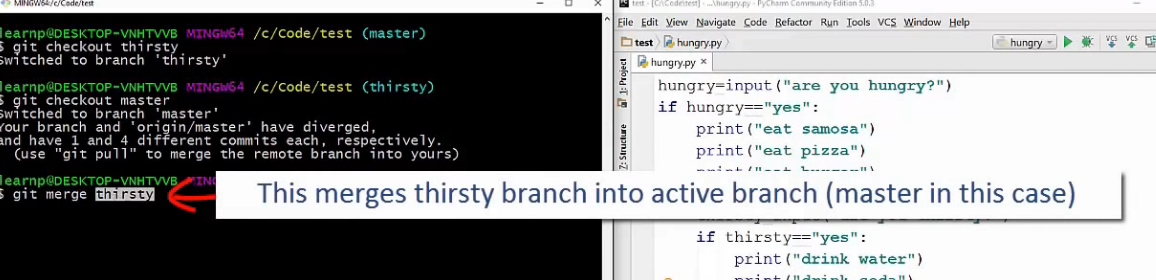
**2.Merge Branch**

1. To merge the changes of thirsty branch to master branch.

* first checkout to the master Branch
* git checkout master

2.git merge branchname:

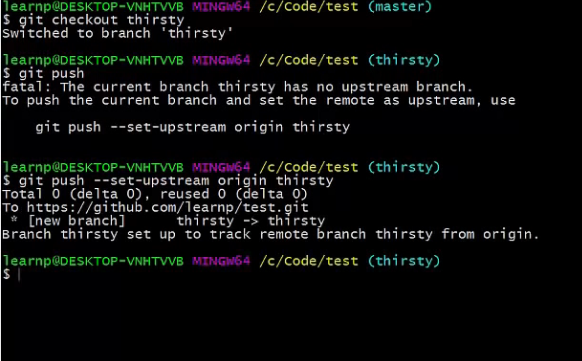
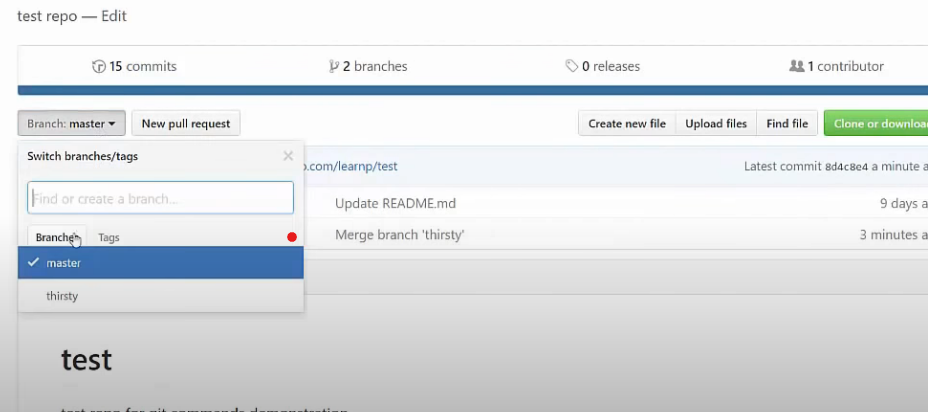
* Here we need to give the branch name of that branch whose changes we want to include in master branch
* Ex: git branch thirsty



1. Below Image Clearly depicts the how thirsty branch is merged in to master branch

Diagram

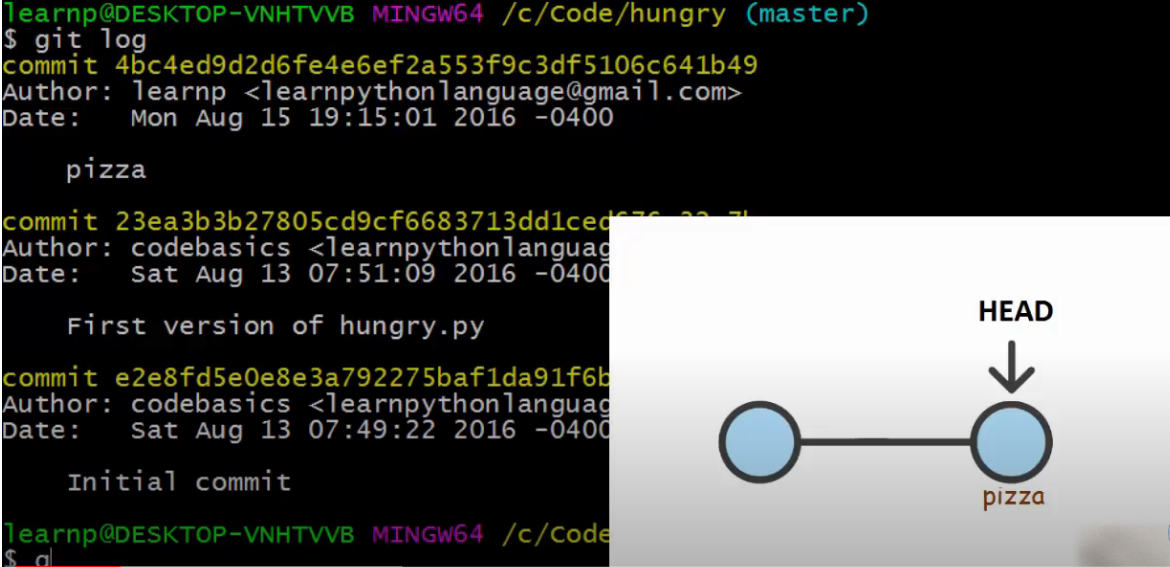
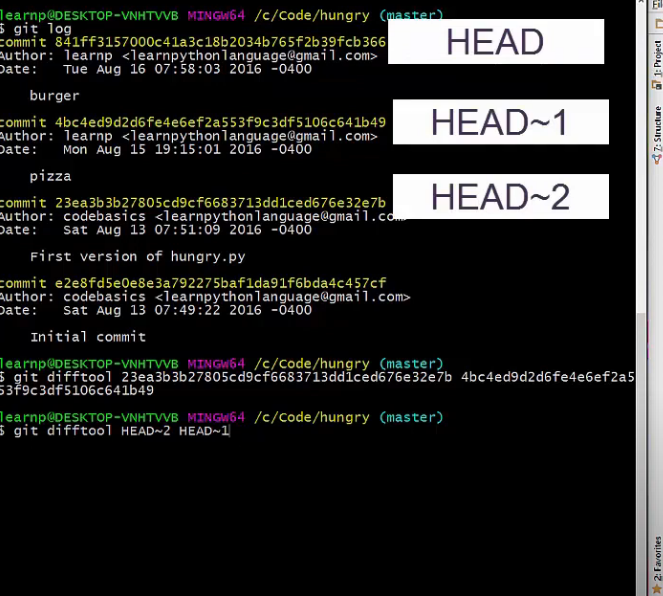
Description automatically generated with medium confidence

1. Inorder to push the thirsty branch on to the github make that branch active and push. Below image depicts
2. 
3. 

**3.Delete a Branch:**

* We can create and checkout to the branch with one command: **git checkout -b branchname**
* Ex: **git checkout -b dummy**
* Command to delete the branch is : **git branch -d branchname**
* Ex: **git branch -d dummy**

**HEAD:**

* Head is nothing but reference to the most recent commit in current branch (In most of the cases)
* Ex: Below image
* 
* **git show HEAD**: command to see the Head reference.
* It is nothing but short hand form of most recent commit
* It is also used instead of writing the whole commitid we can use HEAD in the following manner.
  + Head is the most recent commit id
  + Below that is Head~1
  + Below that is Head~2
  + So on….
* It is used for difftool that is seeing the difference between them
* 
* Graphical user interface, text, application

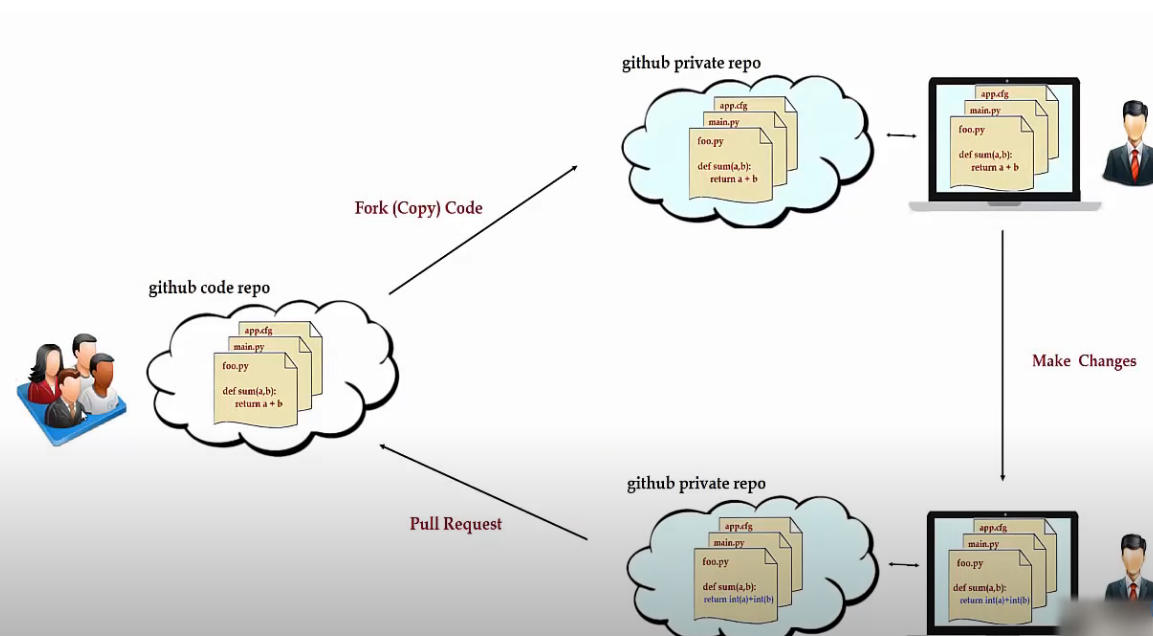
  Description automatically generated
* When head doesn’t point to most recent commit, it goes in to **detached Head** state

.**gitignore file:**

* We include all the files which we don’t on to the upStream i.e on to the github
* Example: we don’t want node\_modules file because it is a large library file which can be installed again.
* We include that in .gitignore file.

**PULL REQUEST:**

* What is pull Request?
  + On GitHub, Owner can share his code with others.
  + Other person can make code changes and send a request to owner to pull/merge his code changes into owner’s repository.
  + This Request above is called pull request.
* It can be easily Explained in the below diagram.

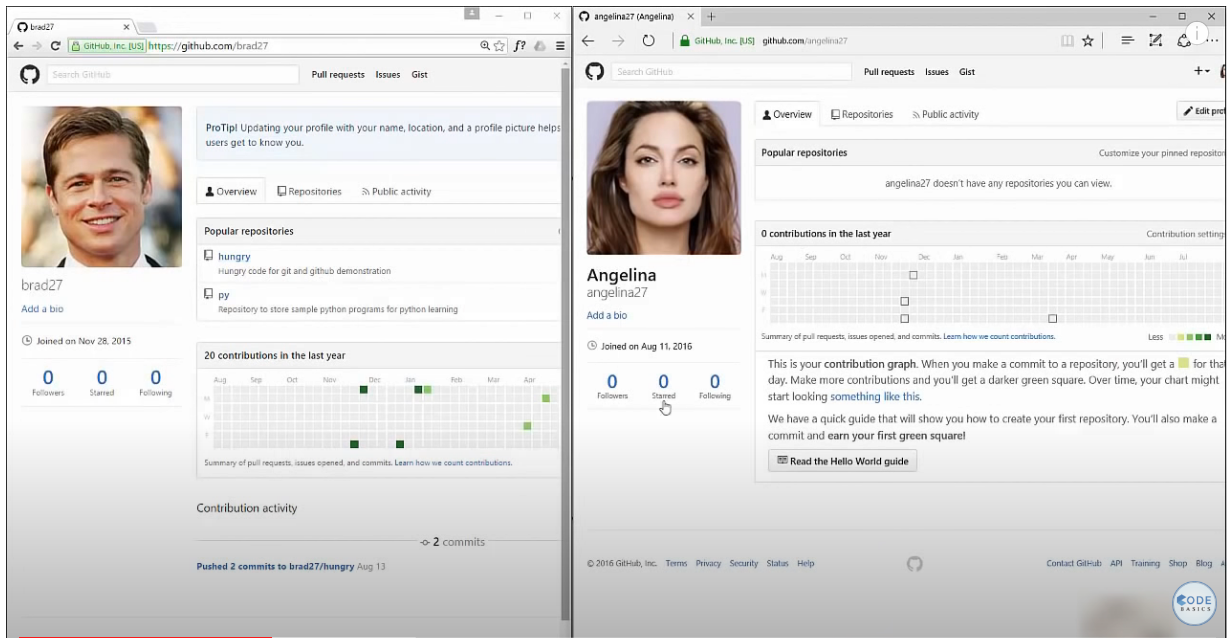


* In the above diagram, there is a group of people(Xyz) who owns the code and a person(Ramesh) who wants to do changes to the code
  + Firstly , Ramesh will fork(copy) the code in to his github private repository
  + He will sync the code to his personal computer
  + He will Modify the code(blue colour) and push the code to his github private repository
  + Then, he will pull request to xyz to review the code and if they are ok with it then merge the code.
* Lets go step by step to easily understand the above concept.

**Following Steps will make you understand the Pull request concept.**

**Step 1:**

Consider we have two people **Brad** and **Angelina** with their GitHub accounts.



**Step 2:**

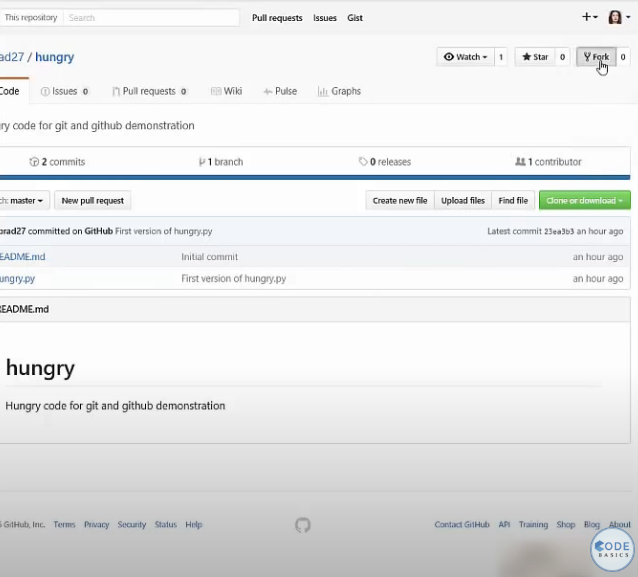
Consider Brad has a Code Repository Named Hungry.

Graphical user interface, text, application, email

Description automatically generated

**Step 3:**

Now Angelina will try to fork the code in to her own GitHub Account. She will copy the url and paste it. She will copy by clicking on to fork button



**Step-4:**

After forking the repo, she will pull the code in to her local system and after modifying the changes she will push back to her own repository in github. Consider she added a single line such as instead of accepting only ‘yes’ as input she also added ‘y’ to be accepted.

Before changes the code was:

Graphical user interface, application

Description automatically generated with medium confidence

After Changes:

Graphical user interface, text, application

Description automatically generated

**Step 5:**

After Making the changes, now is the time to perform the pull request. Now she is ready to merge the changes in to the Brad repo. For that she will go to Pull request Tab and then click on New Pull Request.



Graphical user interface, text, application

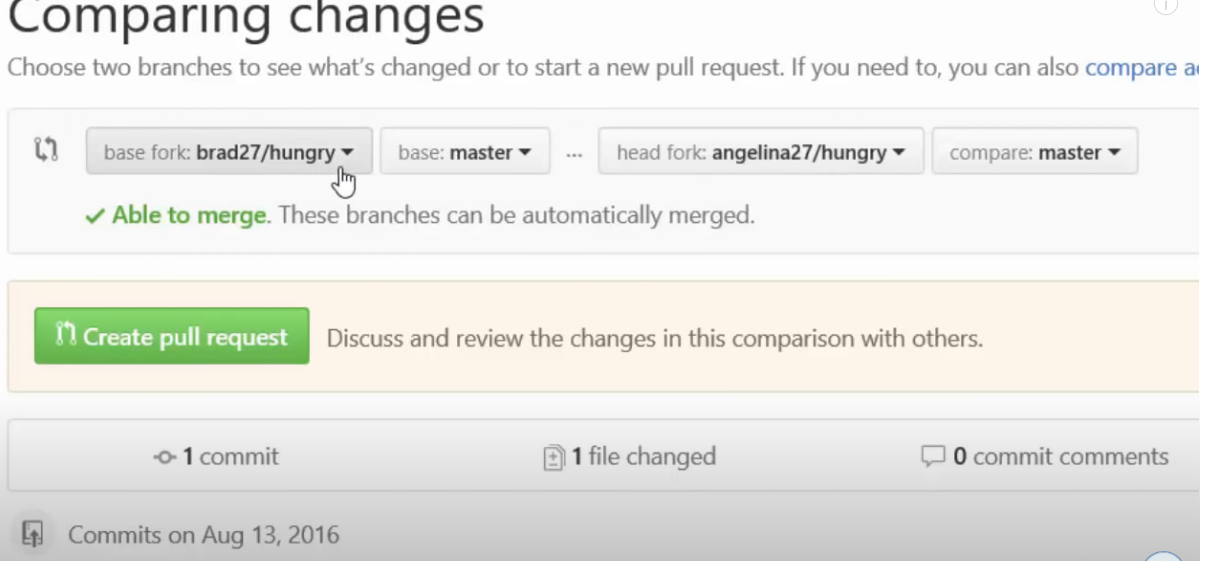
Description automatically generated

Next->

Graphical user interface, application

Description automatically generated

Next->



Now click on Create Pull request. Add the meaning full comment . Now Angelina is waiting for Brad response.

**Step 6:**

Now Brad will get the notification and if he will do the review and if he thinks that it is good enough to be added then he will Merge the request. The Code inside the Brad repo will be changed.

Graphical user interface, text, application, email

Description automatically generated

Step 7:

Angelina will also be added to the contributor .

Graphical user interface, text, application, email

Description automatically generated

**Common Errors in Git:**

* fatal: remote origin already exists.:
  1. This error occurs when we try to execute the following command.
  2. git remote add origin <SOME-URL>/<SOME-REPOSITORY-NAME>.git
  3. Understanding this message is actually easy. Unlike centralized VCSs, Git doesn't have a central server. In Git, you can have what we call remote repositories, or simply remotes. Remotes represent repositories that you might have read and/or write access to. Those are usually on machines other than your own, and you access them via SSH or HTTP. Keep in mind that, despite the name, remotes aren't necessarily located on remote machines: despite sounding like an oxymoron, local remotes are totally possible.
* **Solving the problem:**
  1. Remove the Existing Remote:
     + git remote remove origin
     + Create a new repository online using GitHub or GitLab.
     + Go to your local repository and remove the existing origin remote.
     + Add the new online repository as the correct origin remote.
     + Push your code to the new origin.
  2. Update the Existing Remote's URL:
     + git remote set-url <REMOTE-NAME> <NEW-URL>
     + Ex: git remote set-url origin <https://github.com/git/git.git>
  3. Rename the Existing Remote:
     + git remote rename <old-name> <new-name>
     + Ex: git remote rename origin backup
     + Easy. Rename the existing remote before adding the new one.
     + So let's say you want to rename your origin remote to backup. You will run the above command.
     + Then you can add your new remote called origin normally, and you should no longer see the "remote origin already exists" error.