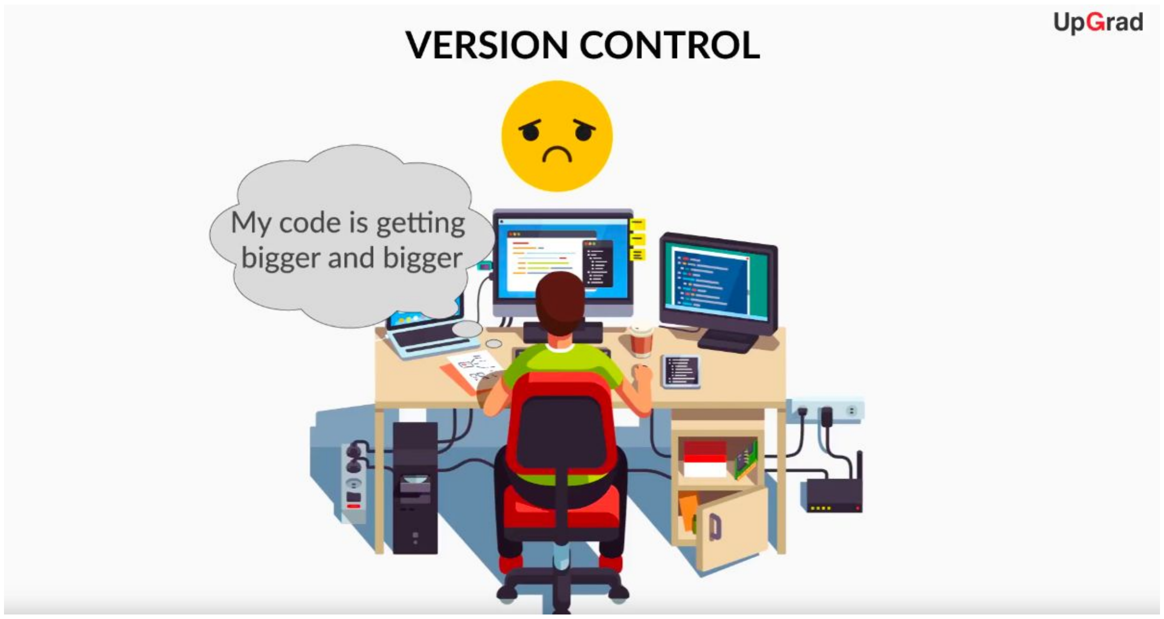
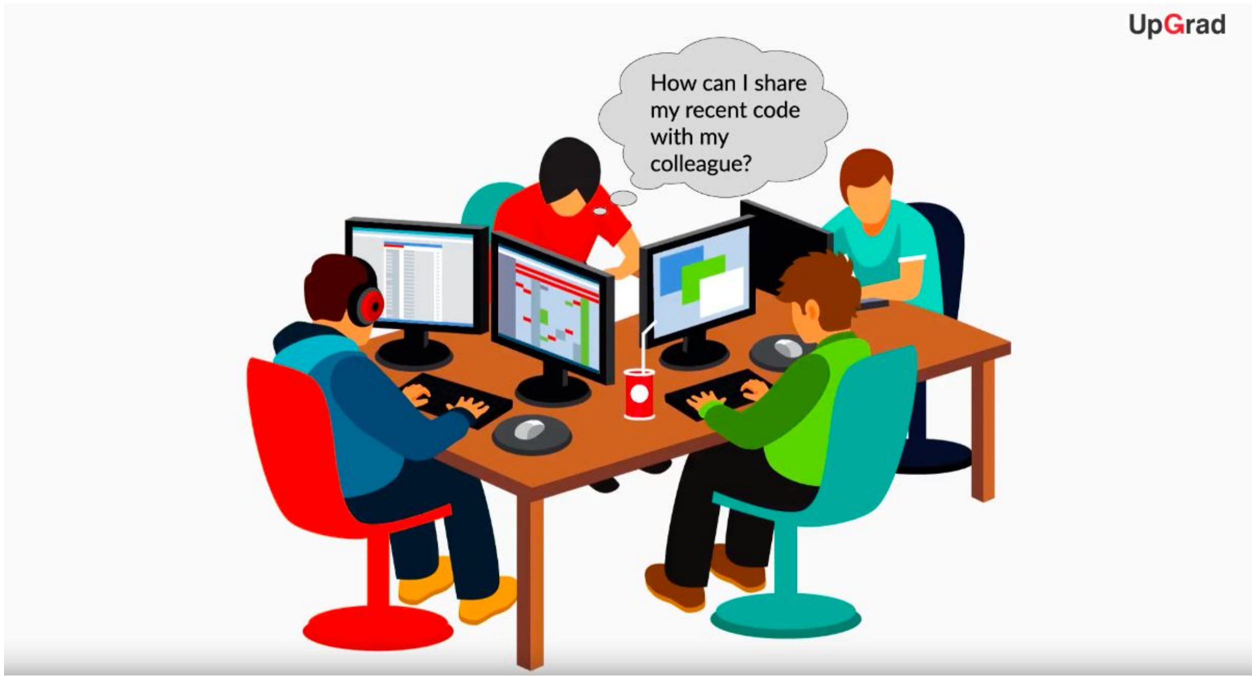
Why version control?

In this session we gave you a brief on the issues which you would have faced without version control.



How to manage your code if its getting bigger and bigger?

How to go back to the last good version of your code?



How can we share our files to other people in our team?

To summarize Version control is a solution to all the issues mentioned above like-

- Reverting back to an older version of your project in case you think that you have messed up things

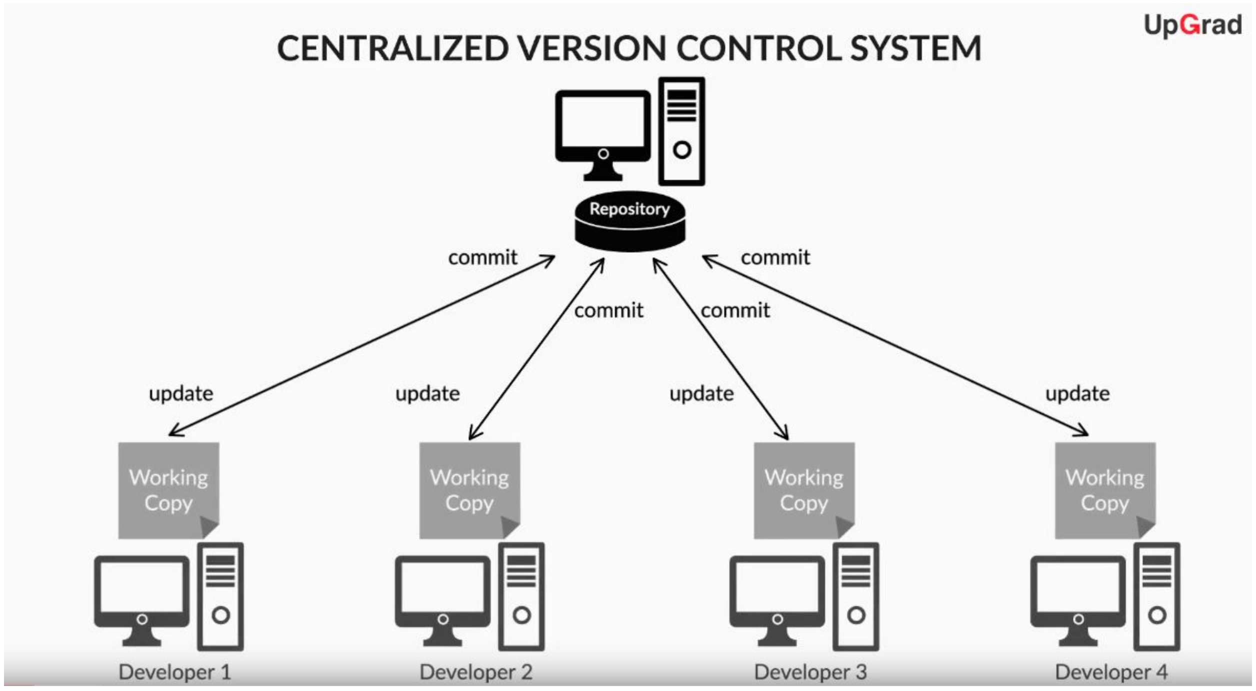
- An efficient way of keeping track of changes or additions to the project files by the various team members

Overview of version control?

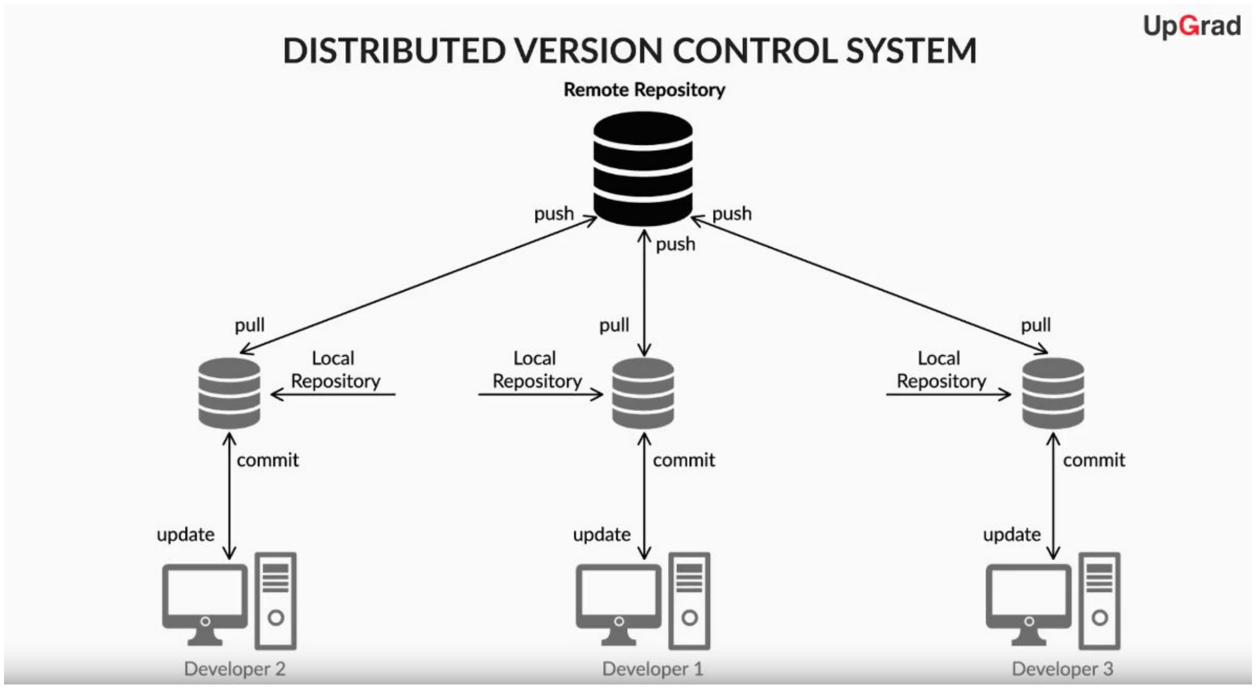
In this segment you learnt what is version control system and how version control works. You learnt the mechanism that goes behind it.

You learnt that version control system is the management of changes to collection of information like documents, project code.

- There are two major types of version control model namely
    - a. Centralized
    - b. Distributed
- The **centralized version control system** is working as a client-server model. Here we have one centralized server and a localized repository filesystem that is accessible by a number of clients



- In the **distributed version control system** model, all developers have their own local filesystem, and changes between file system are implemented locally on their machines.



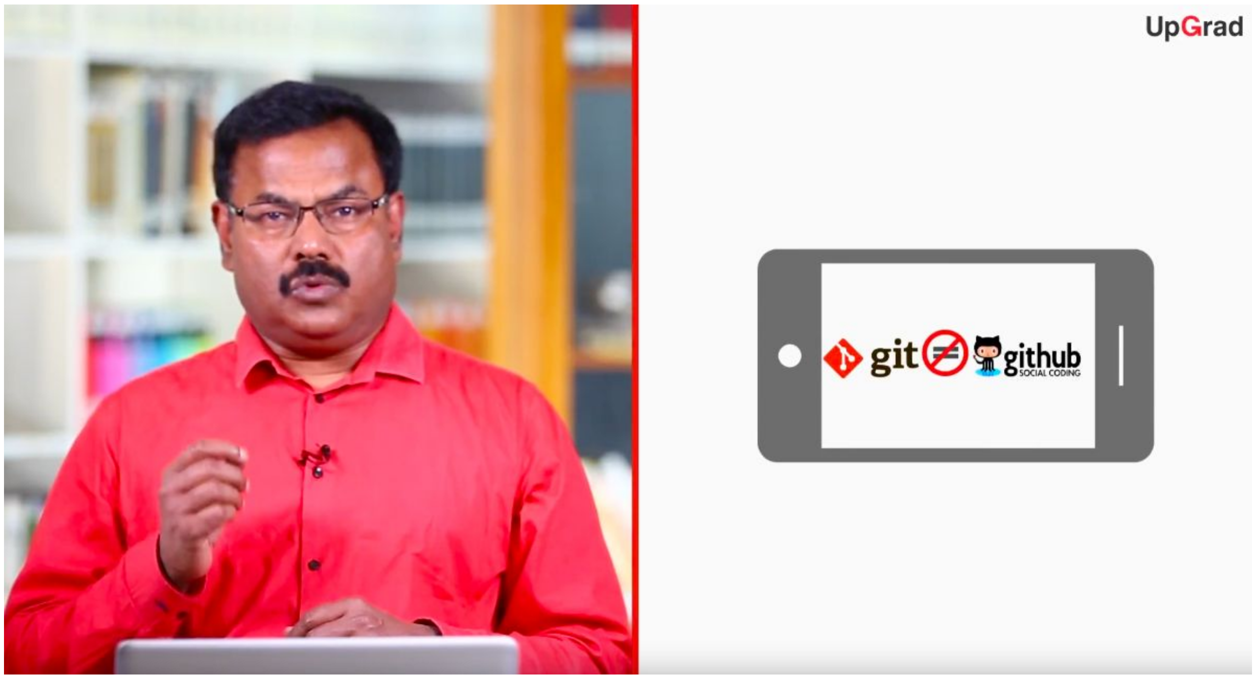
- You also learnt why we use Git over all the other distributed version control systems.
- You did the git installation on your system

Git and Github

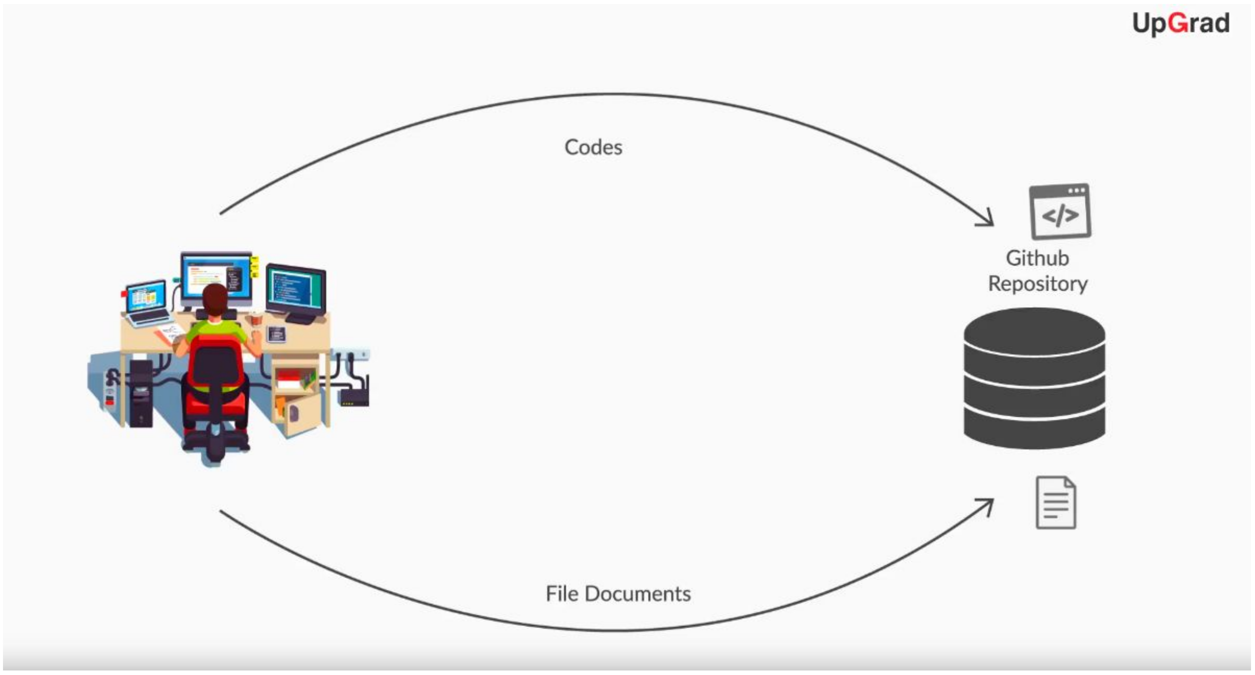
So far you just worked with Git and you had files in your local systems, there was no way by which you could share your files with your team members or other developers.

In this video you were introduced to Github.

- You learnt that using Github you can share your file system like files, documents etc with others, access other user's file system and store remote files and projects of other developers on your local system.



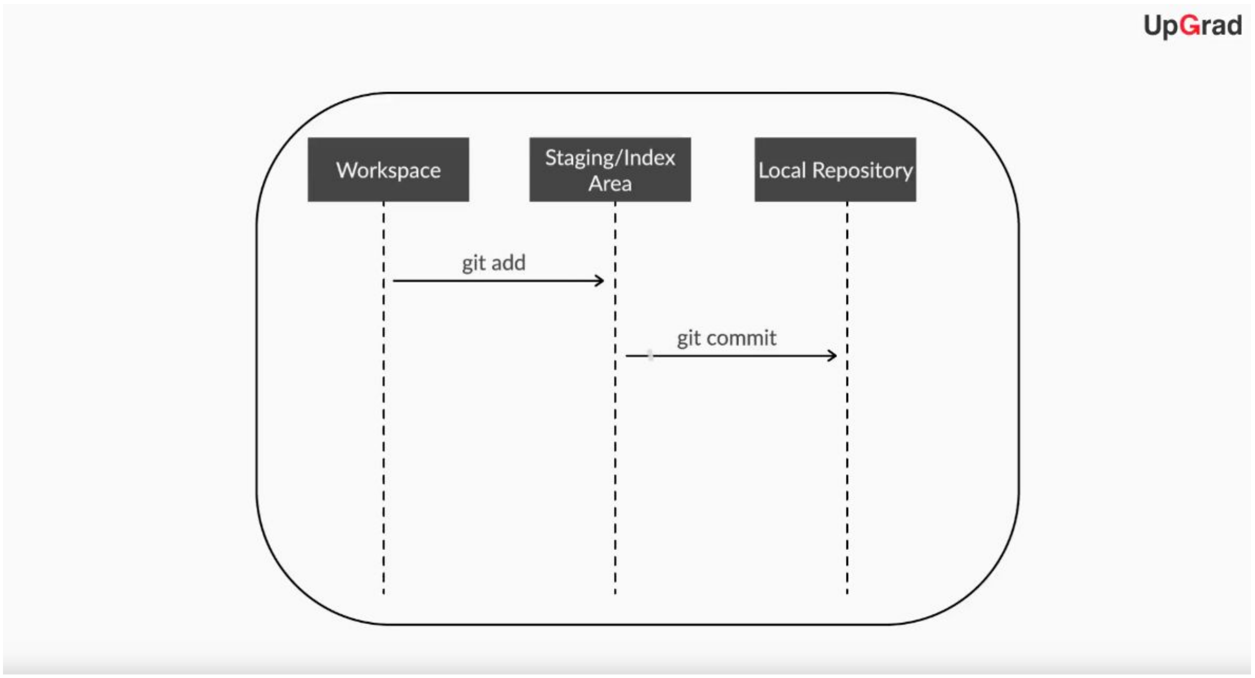
- You learnt about the differences between Git and Github
- You learnt a new term which is known as a **Repository.**
- A **repository** is a directory which will contain our project work. All the files in the repository can then be uploaded to Github and shared with other people either publicly or privately.



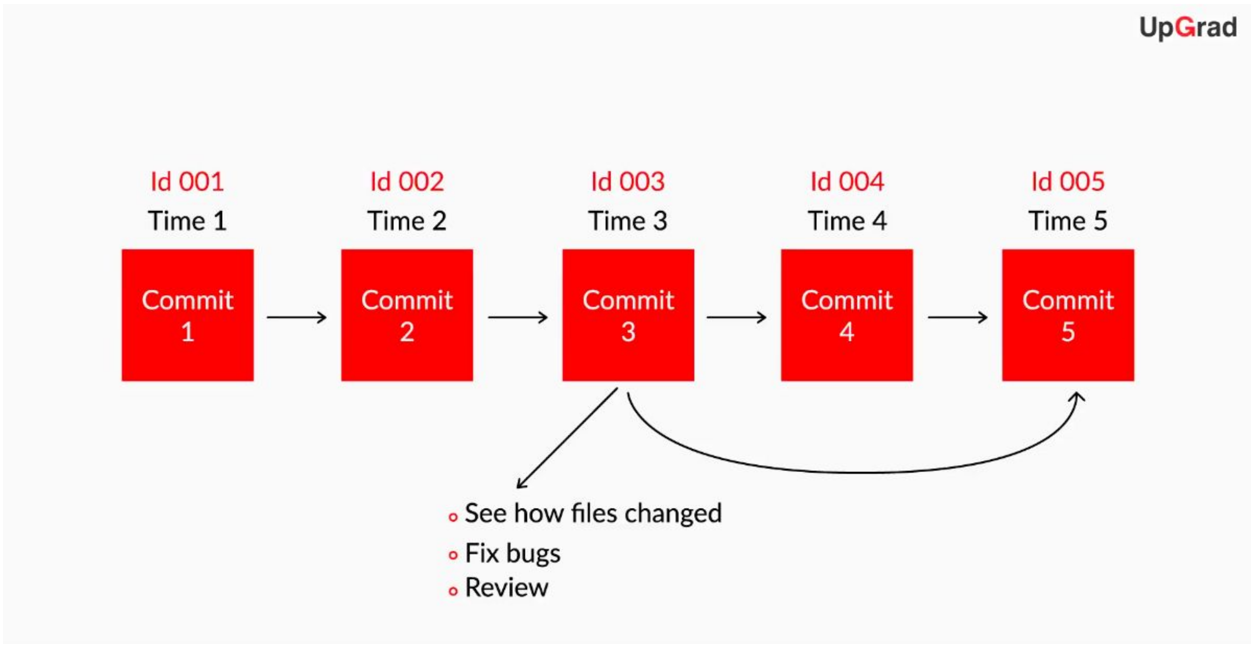
- You also learnt about -

The steps our files might be going through internally. The three steps being -

1) **Modified**
2) **Staged**
3)Committed



You learnt that each commit has a unique id or SHA and that is how they are distinguished from each other.

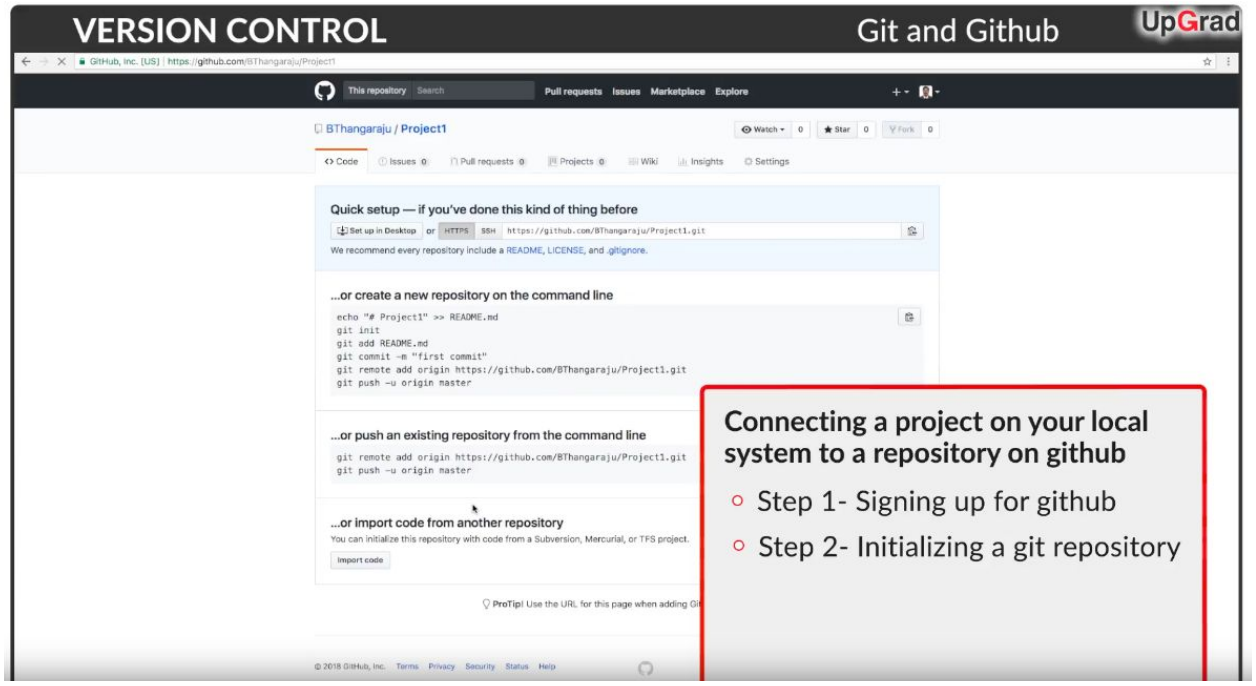


Now don't you want to know how to sign up for a github account and upload your code to github?

Making your project remote

In this segment you learnt the following -

1. How to set up our github account
2. Making a repository on Github
3. Configuring Git for the very first time
4. How to make a commit
5. How to push commits in our local repository from command line to our repository on Github


VERSION CONTROL

Git and Github

BThangaraju / Project1

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/BThangaraju/Project1.git`

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# Project1" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/BThangaraju/Project1.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/BThangaraju/Project1.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

ProTip! Use the URL for this page when adding Gi

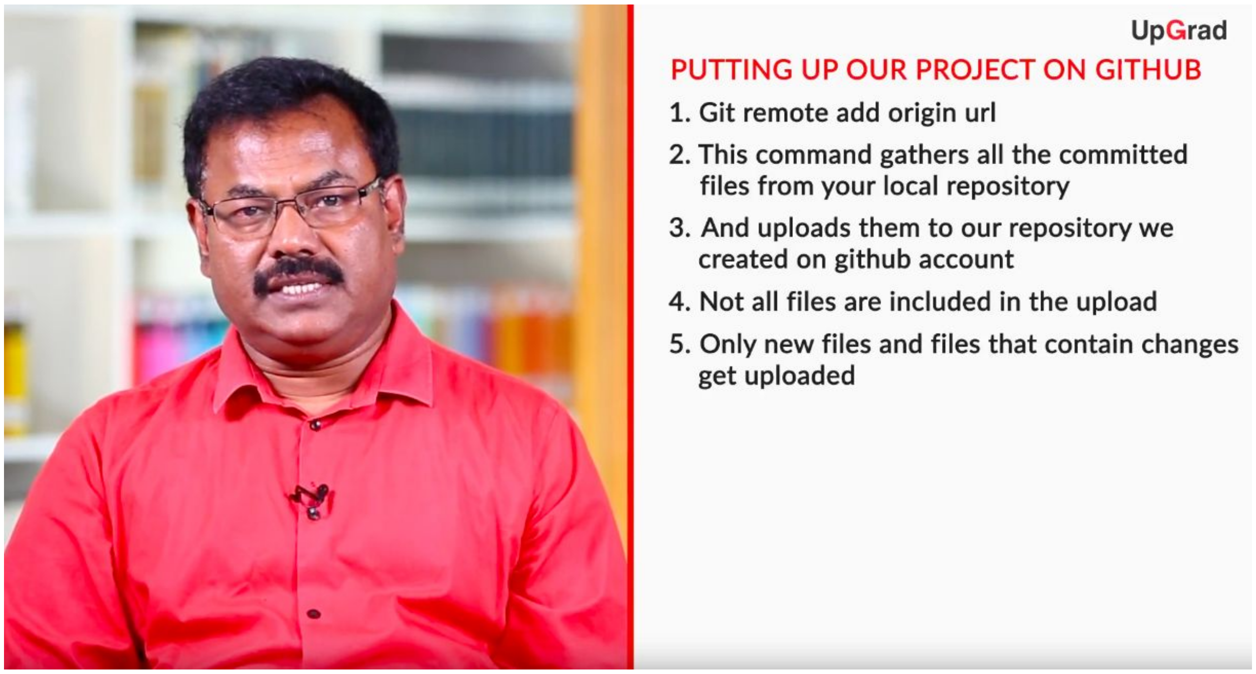
Connecting a project on your local system to a repository on github

- Step 1- Signing up for github
- Step 2- Initializing a git repository

- **You did the first time git setup**
    - For the first time git configuration we use the following commands
    - git config --global user.name "random”
    - Here we will enter our Github username
    - git config --global user.email “random@example.com”

    - Here we will enter our Github email id

- **Commands to be followed to connect your local repository to remote repository on github-**
    - **git init**
    - **git add filename**
    - **git commit -m "my first commit"**
    - **git remote add origin url**
    - **git push -u origin master**



PUTTING UP OUR PROJECT ON GITHUB

1. Git remote add origin url
2. This command gathers all the committed files from your local repository
3. And uploads them to our repository we created on github account
4. Not all files are included in the upload
5. Only new files and files that contain changes get uploaded

- **Some important Git commands which are used very frequently**
    - ***git status***
    - **git log**

Note:
We will only need to run the following git commands to commit any changes into our git repository and push those changes up to github:
1) **git status** - to check which files are changed and not yet moved to the staging area
2) **git add filename** or **git add .** - to add specific files giving the filename or using **git add .** to add all the unstaged files to the staging area
3) **git commit -m "New commit message" -** Give a new commit message and commit all the files sitting in the staging area

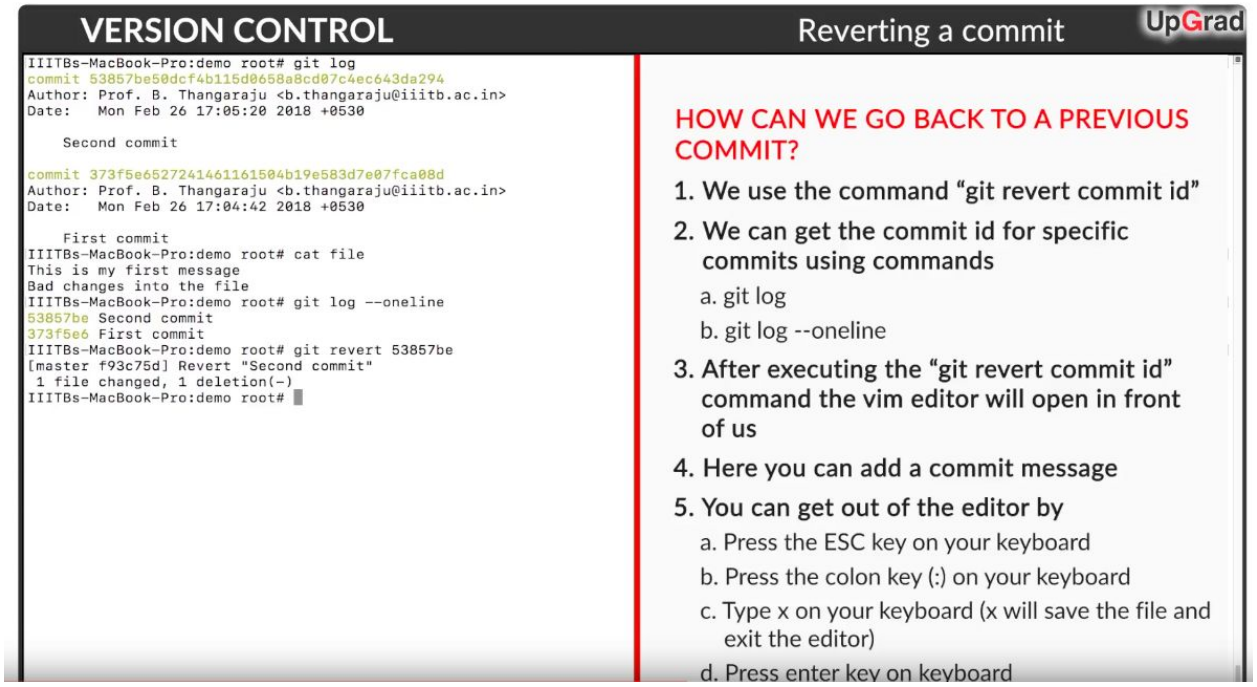
4) **git push -u origin master** or **git push -** To upload all the files and changes that were included in the most recent commit to your remote repository on Github

’.

Accessing previous commits

In this segment you learnt the following-

- If you have been working on a project and you realize that you have messed up your code
- You can go back to your previous working version and start from that point, rather than starting from scratch
- We can go back to our commits with help of commit id or SHA
- We can get these ids or SHA using **git log** command or **git log --oneline** command



VERSION CONTROL

Reverting a commit

UpGrad

```
IIITBs-MacBook-Pro:demo root# git log
commit 53857be50dcf4b115d0658a8cd07c4ec643da294
Author: Prof. B. Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Mon Feb 26 17:05:20 2018 +0530

    Second commit

commit 373f5e6527241461161504b19e583d7e07fca08d
Author: Prof. B. Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Mon Feb 26 17:04:42 2018 +0530

    First commit
IIITBs-MacBook-Pro:demo root# cat file
This is my first message
Bad changes into the file
IIITBs-MacBook-Pro:demo root# git log --oneline
53857be Second commit
373f5e6 First commit
IIITBs-MacBook-Pro:demo root# git revert 53857be
[master f93c75d] Revert "Second commit"
 1 file changed, 1 deletion(-)
IIITBs-MacBook-Pro:demo root# 
```

HOW CAN WE GO BACK TO A PREVIOUS COMMIT?

1. We use the command "git revert commit id"
2. We can get the commit id for specific commits using commands
   a. git log
   b. git log --oneline
3. After executing the "git revert commit id" command the vim editor will open in front of us
4. Here you can add a commit message
5. You can get out of the editor by
   a. Press the ESC key on your keyboard
   b. Press the colon key (:) on your keyboard
   c. Type x on your keyboard (x will save the file and exit the editor)
   d. Press enter key on keyboard

- We learnt commands like
  - **git checkout -- filename -** This command helps us when we have made some modifications to our file say file1 and we have not

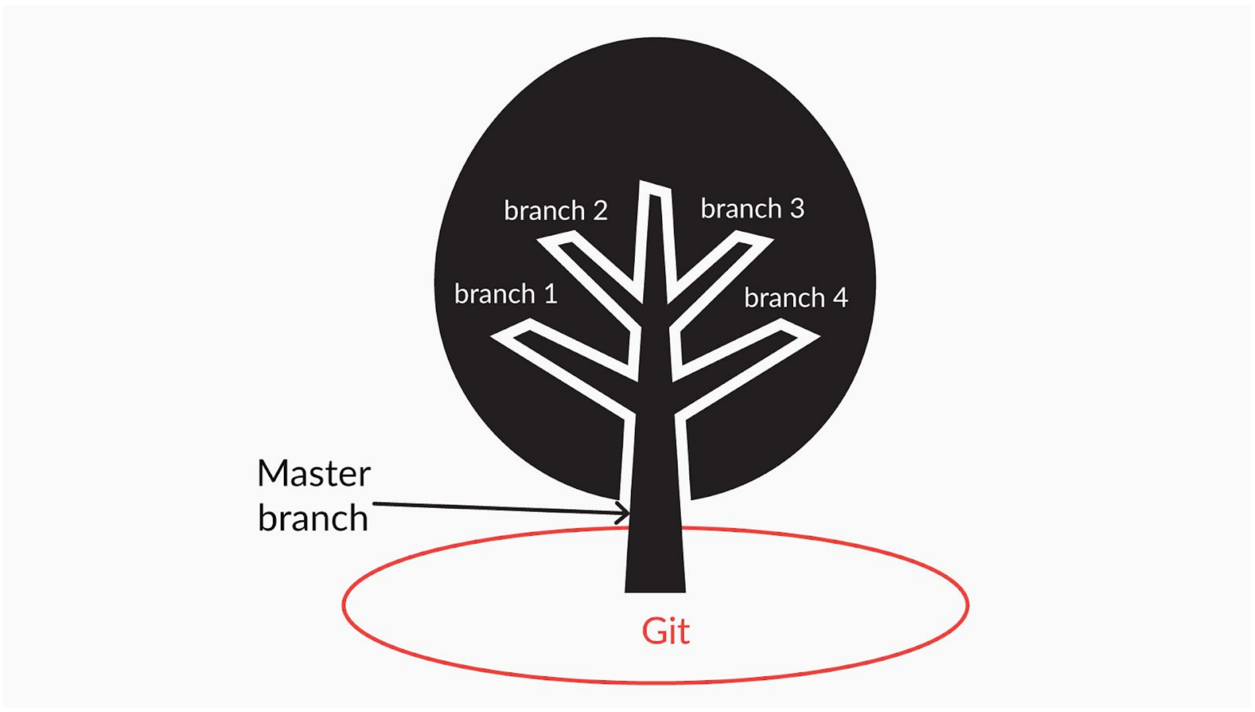
added those changes to our staging area/development history, then we can use this command and will go back to its previous state

- **git revert commit_id -** This command helps us when we have already staged our files and committed our changes and we want to go back to our previous commit. This command let's us go back to our previous version while preserving the current version

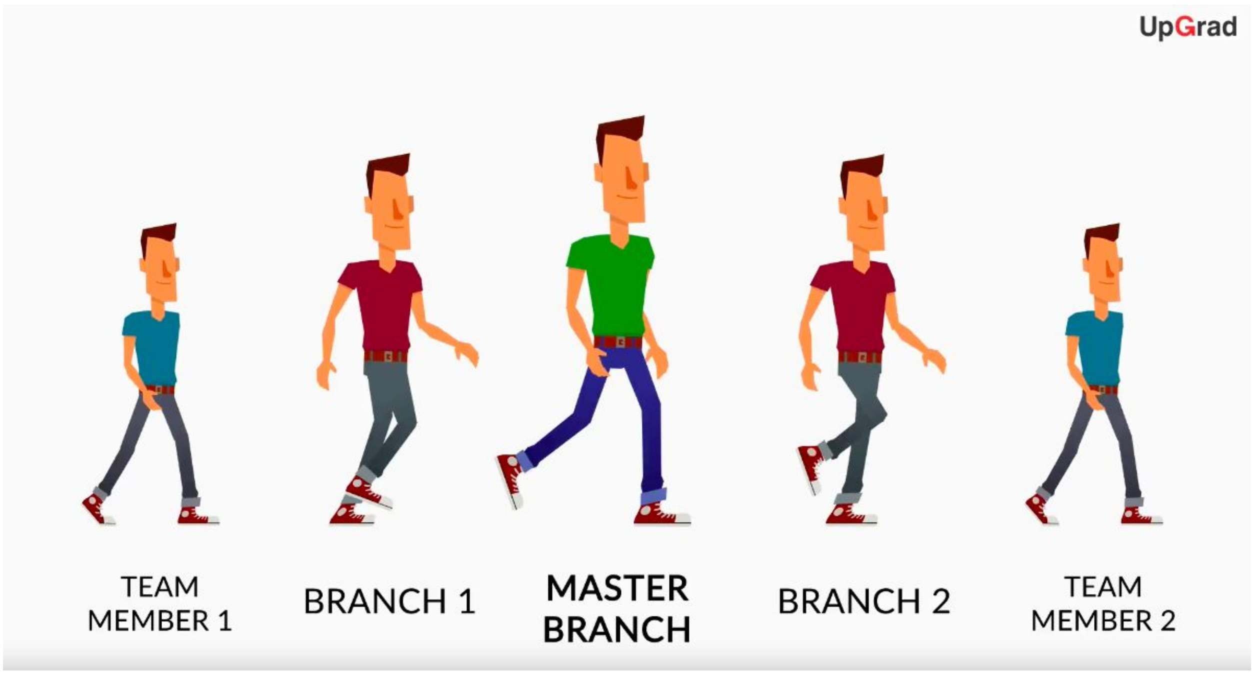
- **git reset --hard commit_id -**This command helps us when we have already staged our files and committed our changes and we want to go back to our previous commit and we want to remove our present commit.

Branching

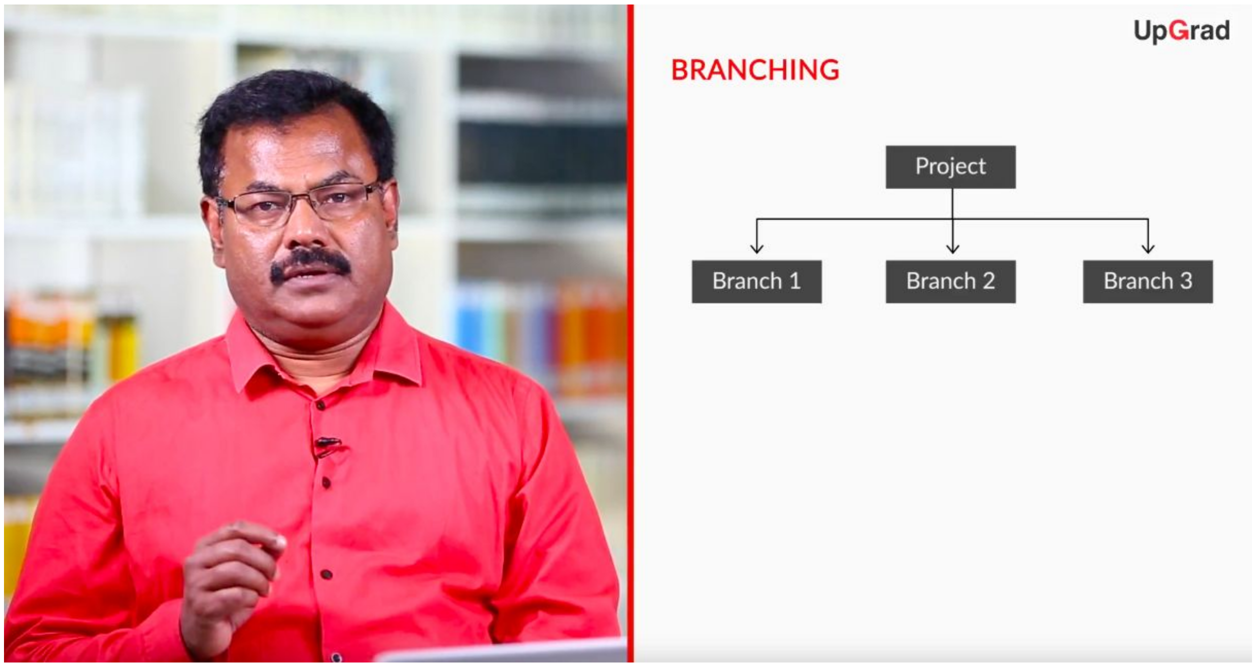
You learnt that if you want to do parallel development with the existing project code, without making any changes to your initial/original branch then you can create different branches based on your need



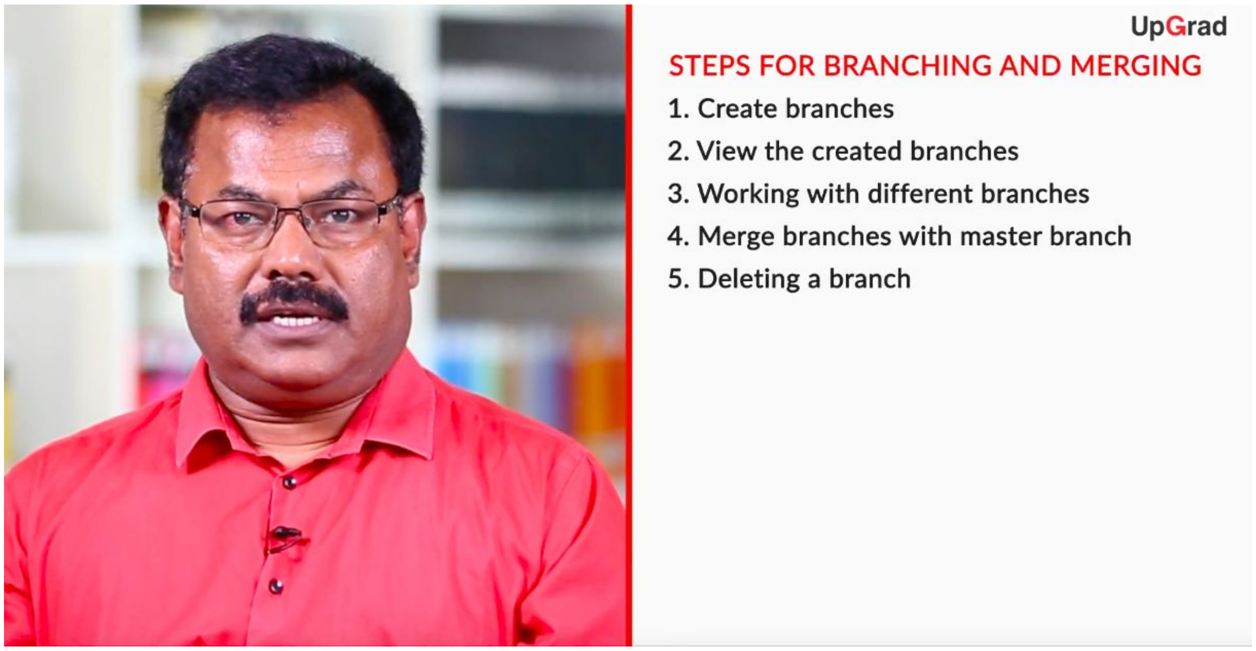
And each branch will have the same copy of the initial/original branch project source code.



You can have your own copy and work on that, make modifications to it, and if the changes look fine you can merge them back to you master branch.



Next we had a recap of the steps we would be following in the next segment.



STEPS FOR BRANCHING AND MERGING

1. Create branches
2. View the created branches
3. Working with different branches
4. Merge branches with master branch
5. Deleting a branch

Working with branches

In this video we executed the following steps-

- Creating branches
- Viewing the created branches
- Working concurrently with different branches

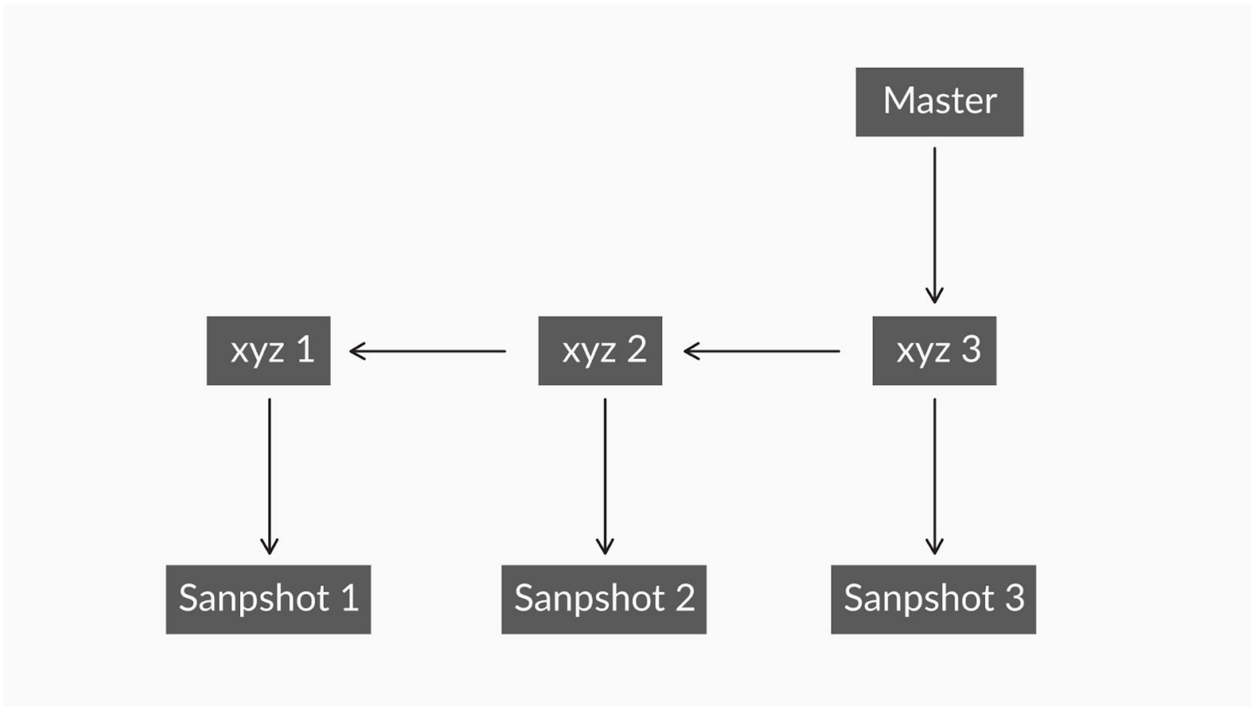
In this video we learnt the following commands-

- **git branch branchname -** This command will create a branch with the given branch name
- **git branch** - This command will show us all the branches, along with the **HEAD** pointing to the branch we are currently on
- **git checkout <branchname> -** If we want to move from one branch to other we can run the command

In-depth concept of branching

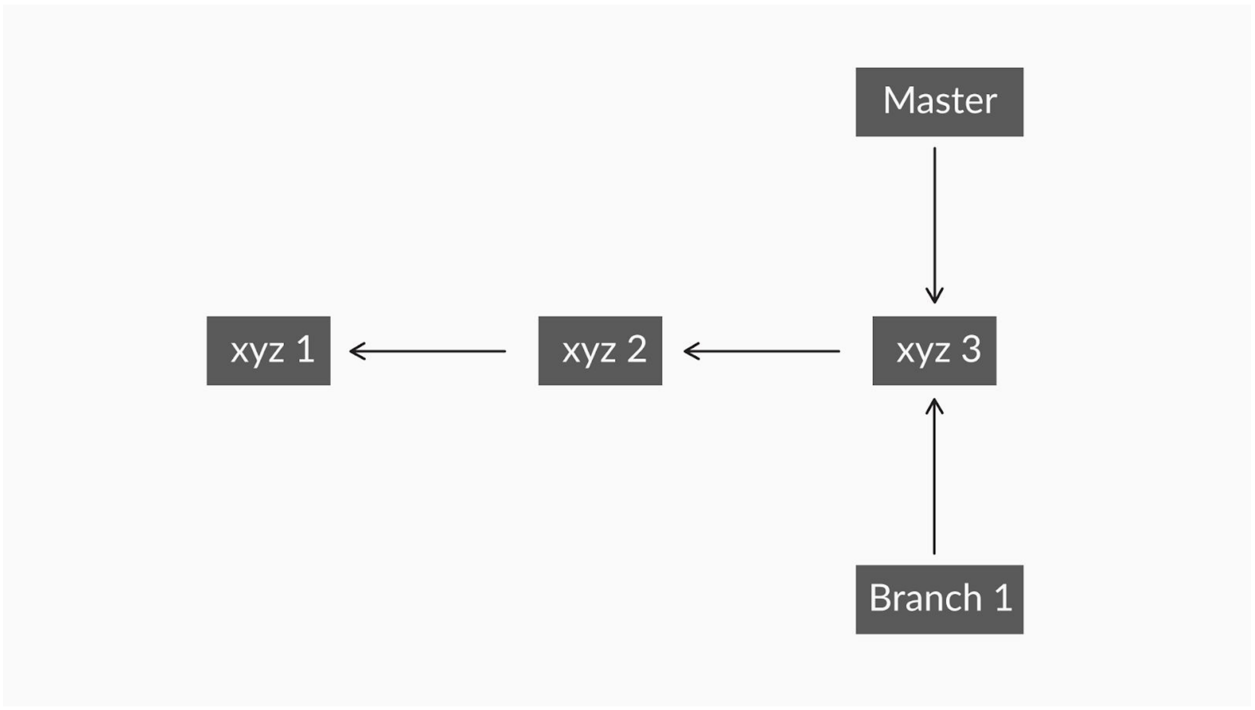
- A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is **master**. Master will point to the last commit. Every time you make a commit, the master will move forward to that commit.

Figure 1. Here the master (branch) is pointing to our last commit



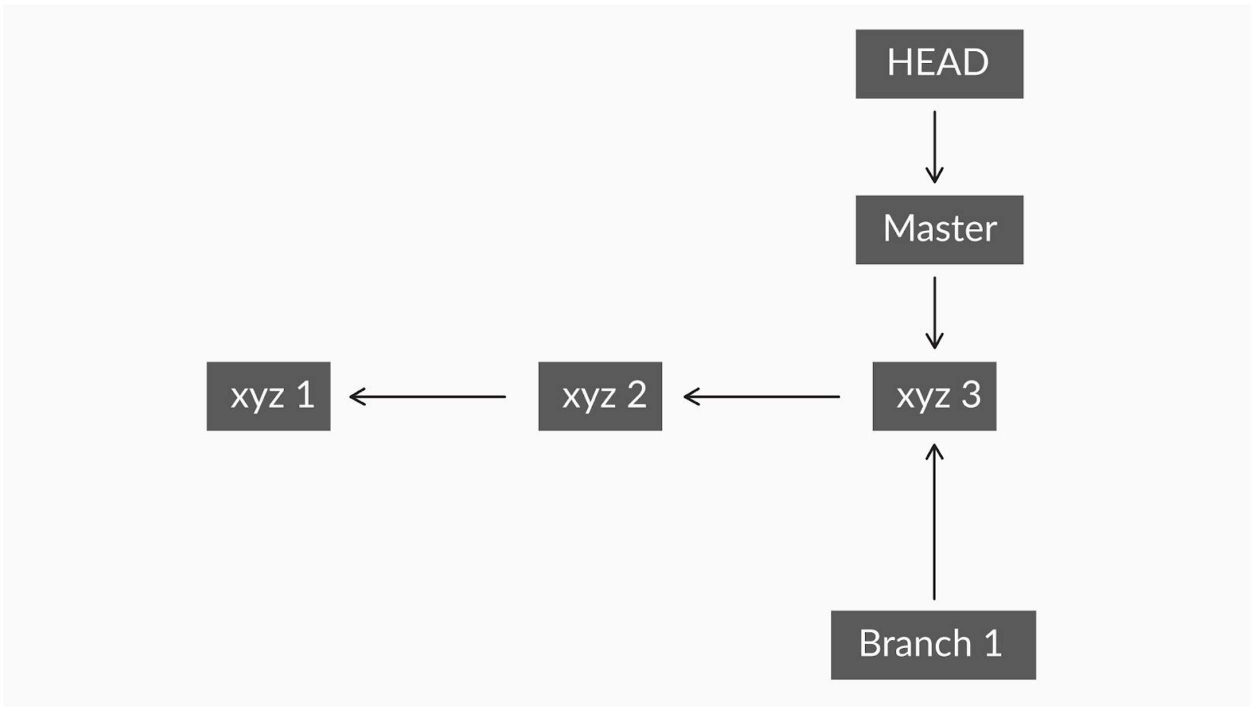
- Now if you create another branch what do you think will happen? When you create a new branch, this will now create a new pointer at the same commit you're currently on

Figure 2. A new pointer at the same commit is created



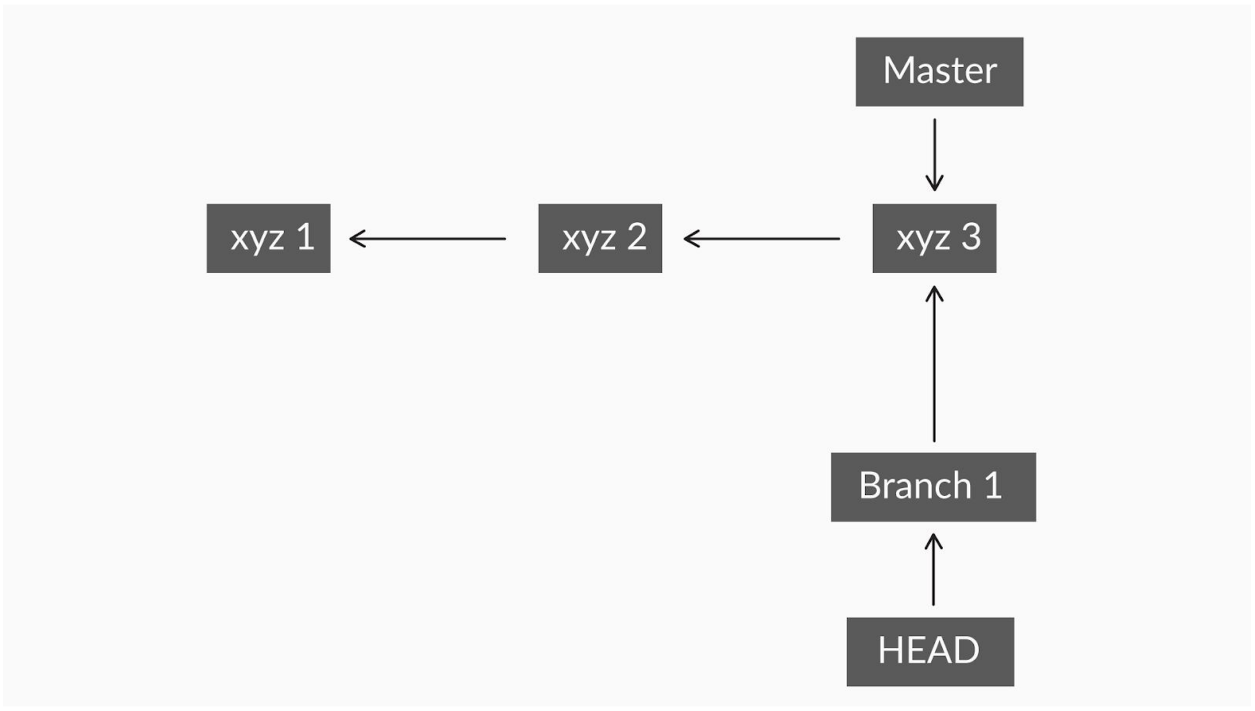
Now how do you think Git knows what branch you're currently on?

- Git keeps a special pointer called **HEAD**. HEAD will point to the branch we are currently on. To make HEAD point to the new branch i.e. **branch1**, we will have to switch to that branch — it didn't switch to that branch.



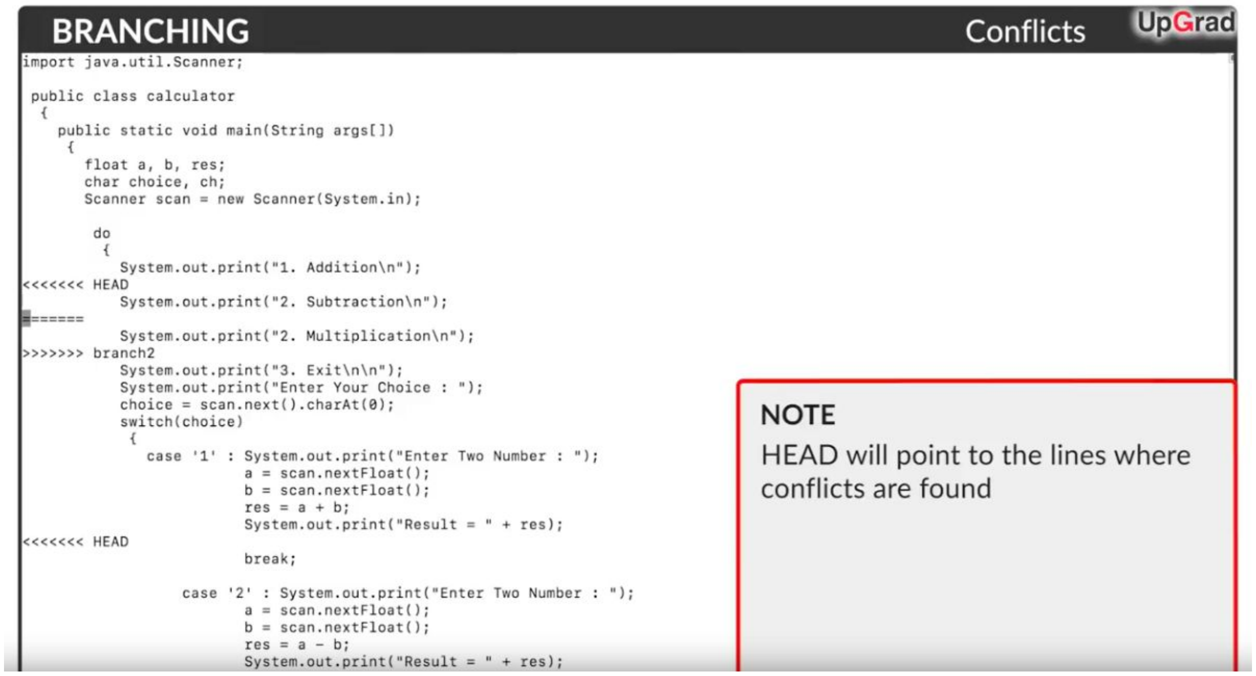
- To switch to **branch1** you need to run the command.
    - **git checkout branch1**

- This will now move HEAD to point to our newly created branch i.e **branch1**


Master
xyz 1
xyz 2
xyz 3
Branch 1
HEAD

Managing Conflicts

IN this segment we learnt that merging is not always as easy as it looks. If two people change the same lines of code, don't you think git will get confused? Yes! To get out of this conflict we follow the steps below-



BRANCHING

Conflicts

```
import java.util.Scanner;

 public class calculator
  {
    public static void main(String args[])
     {
       float a, b, res;
       char choice, ch;
       Scanner scan = new Scanner(System.in);

        do
         {
           System.out.print("1. Addition\n");
<<<<<<< HEAD
           System.out.print("2. Subtraction\n");
=======
           System.out.print("2. Multiplication\n");
>>>>>>> branch2
           System.out.print("3. Exit\n\n");
           System.out.print("Enter Your Choice : ");
           choice = scan.next().charAt(0);
           switch(choice)
            {
              case '1' : System.out.print("Enter Two Number : ");
                         a = scan.nextFloat();
                         b = scan.nextFloat();
                         res = a + b;
                         System.out.print("Result = " + res);
<<<<<<< HEAD
                         break;

                  case '2' : System.out.print("Enter Two Number : ");
                         a = scan.nextFloat();
                         b = scan.nextFloat();
                         res = a - b;
                         System.out.print("Result = " + res);
```

NOTE

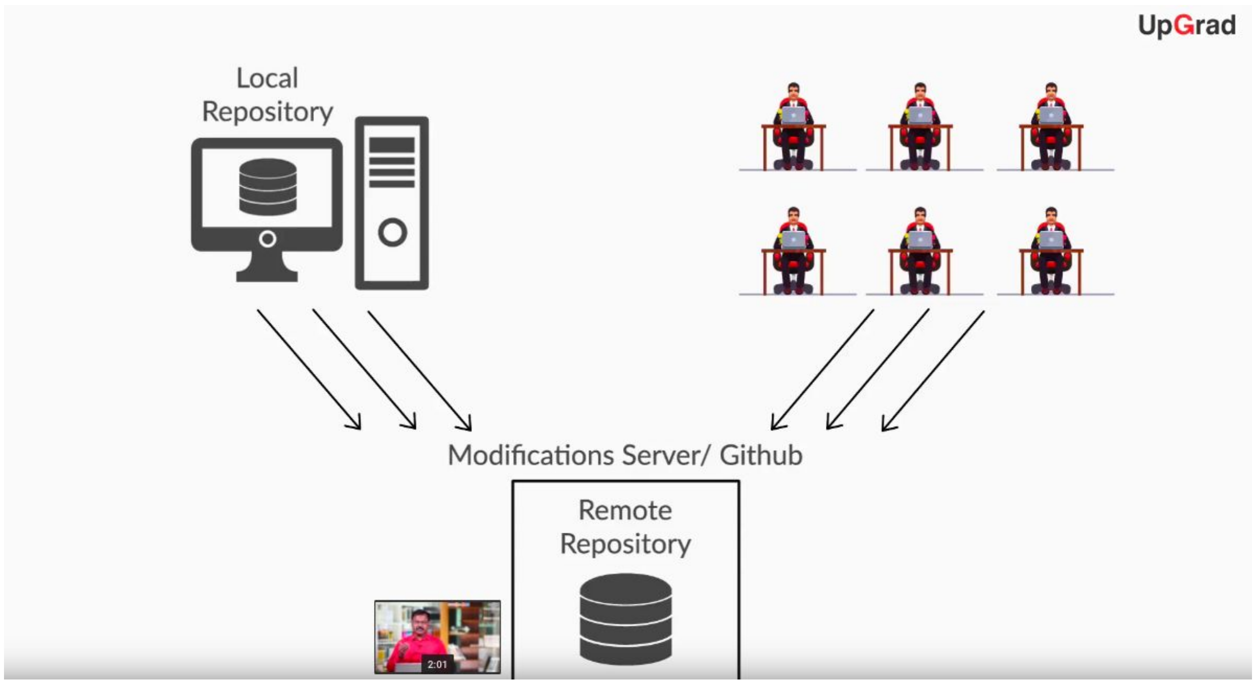
HEAD will point to the lines where conflicts are found

The best way to deal with a merge conflict is to use your best decision that is -

- You can keep all the changes by making necessary modifications
- Or simply discard some changes

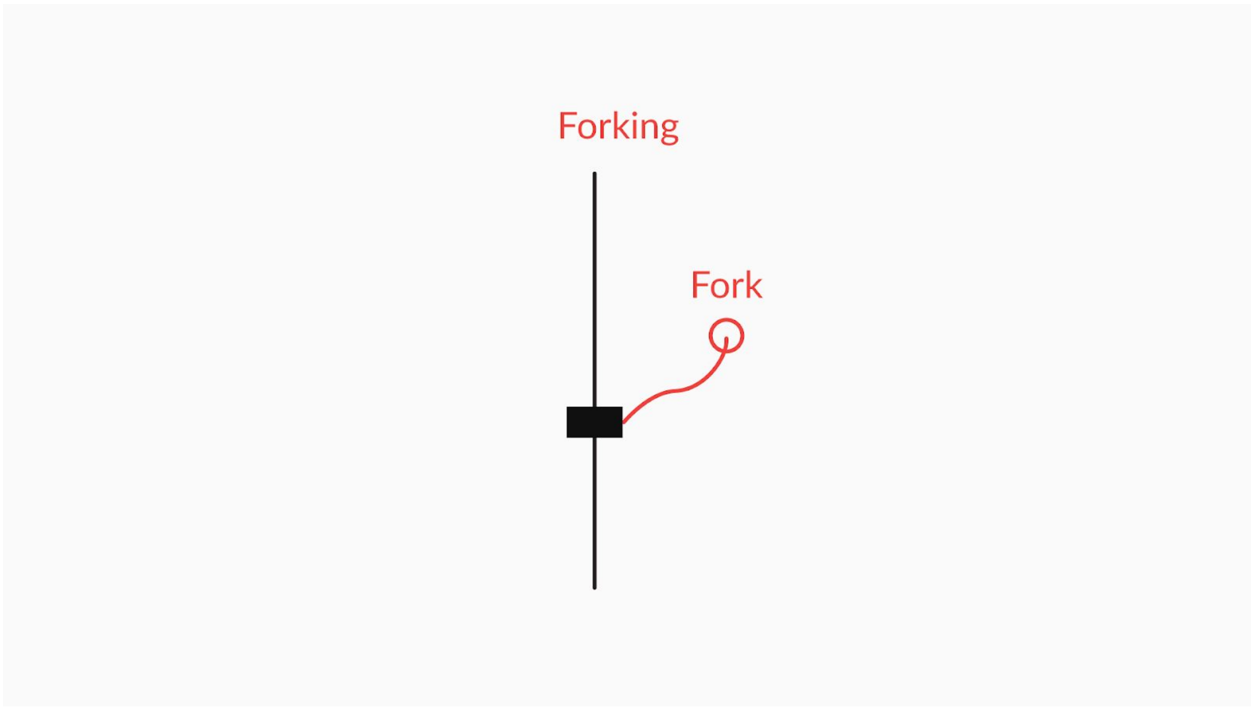
Collaboration

In this segment we learnt that collaboration helps you to work with people around you and produce something even better than before. You can help add features or improvise some of the features in someone's project who is sitting miles away from you.

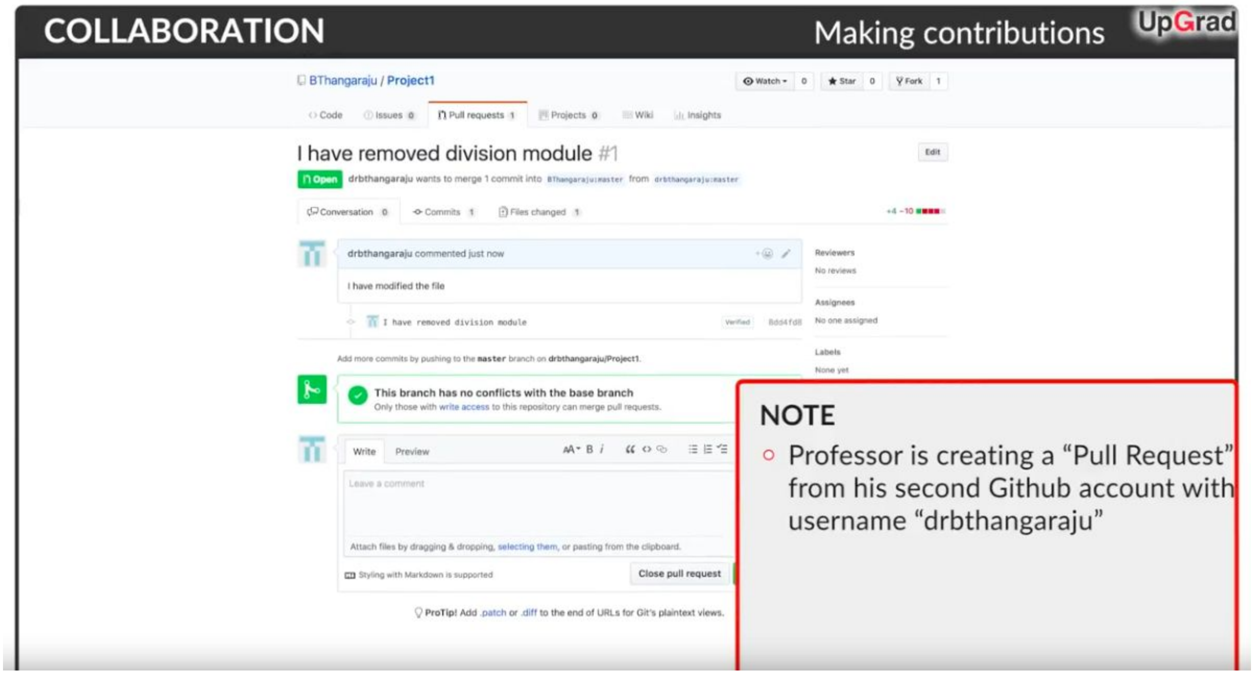


To contribute to someone's project we need to follow the steps below-

- To contribute to someone's project we first need to **fork** it. Forking will get you that project on your github account.



- Now to bring that project to your local system you need to **clone** it.
- Next you will make modifications and push the changes back to your Github account.
- Now to inform the owner of the project about new changes that you made you will create a **pull request**


COLLABORATION
Making contributions
UpGrad

BThangaraju / Project1

Watch 0 · Star 0 · Fork 1
Code · Issues 0 · Pull requests 1 · Projects 0 · Wiki · Insights
Edit

I have removed division module #1

Open drbthangaraju wants to merge 1 commit into `BThangaraju:master` from `drbthangaraju:master`

Conversation 0 · Commits 1 · Files changed 1
+4 −10

drbthangaraju commented just now

I have modified the file

`I have removed division module` Verified 8dd4fd8

Reviewers: No reviews
Assignees: No one assigned
Labels: None yet

Add more commits by pushing to the **master** branch on **drbthangaraju/Project1**.

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

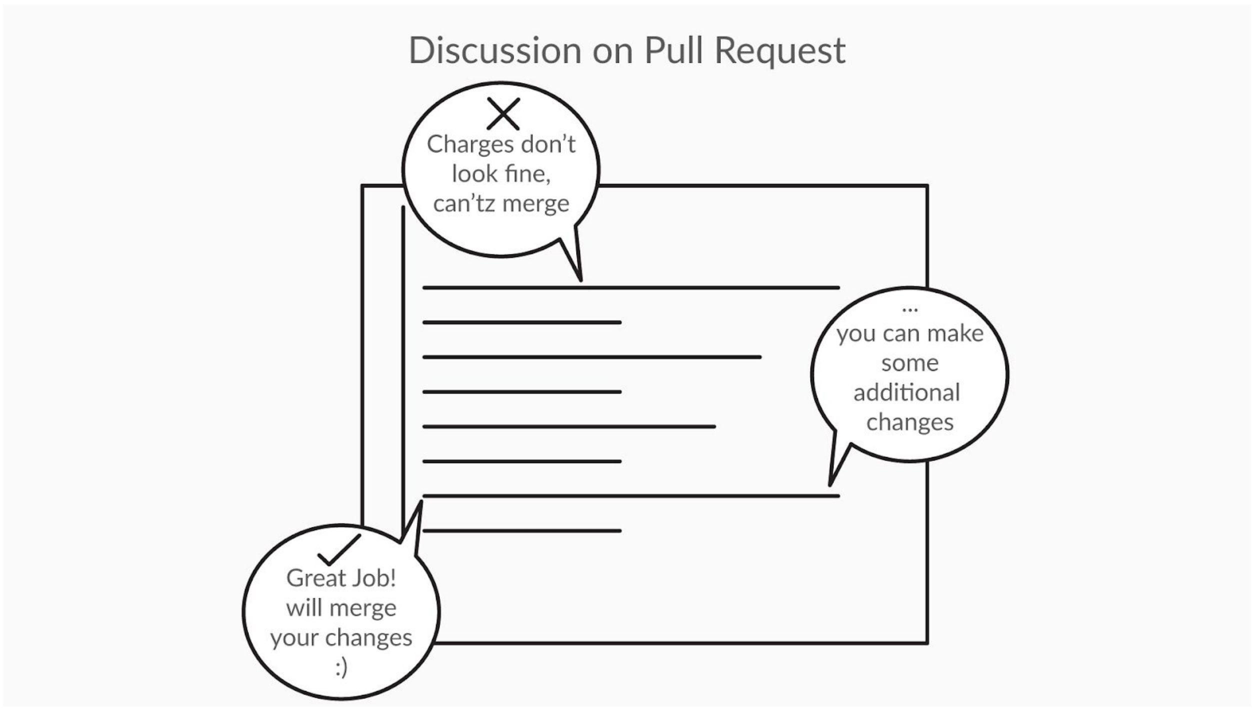
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

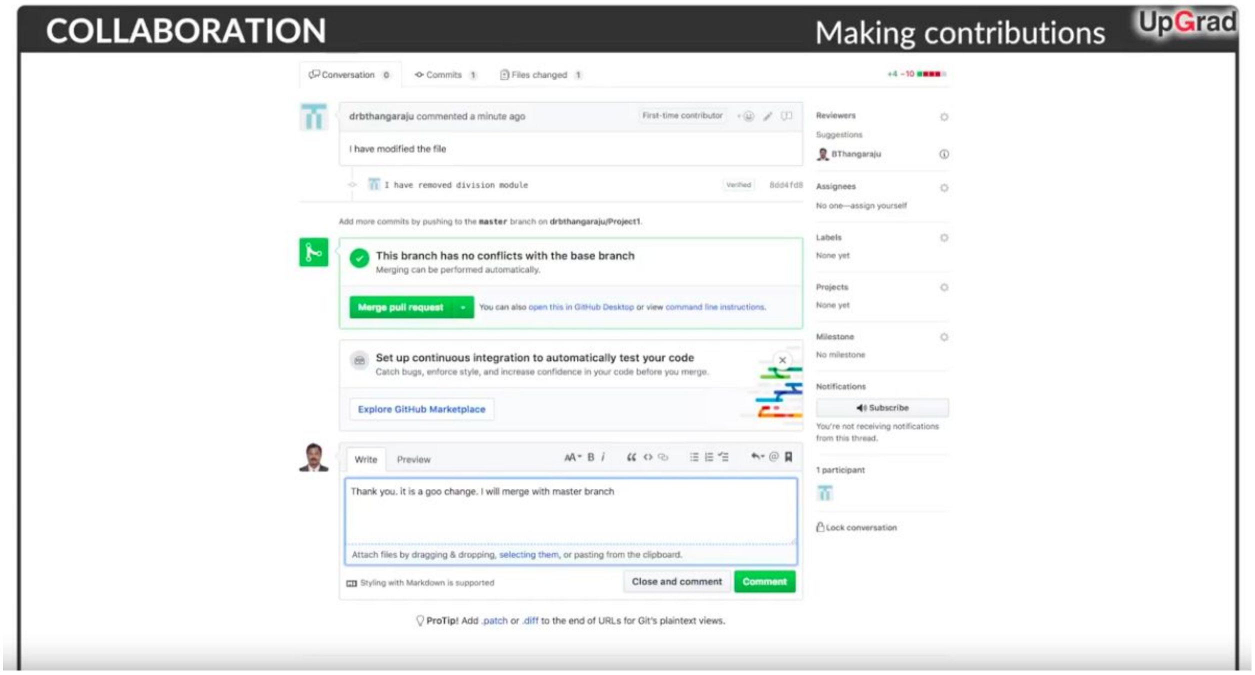
ProTip! Add .patch or .diff to the end of URLs for Git's plaintext views.

NOTE
- Professor is creating a "Pull Request" from his second Github account with username "drbthangaraju"

- There will be discussion on your changes


Discussion on Pull Request
Charges don't look fine, can'tz merge
... you can make some additional changes
Great Job! will merge your changes :)

- If your changes are good, the owner will merge your changes :)



drbthangaraju commented a minute ago

First-time contributor

I have modified the file

`I have removed division module` Verified 8dd4fd8

Add more commits by pushing to the **master** branch on **drbthangaraju/Project1**.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

Set up continuous integration to automatically test your code
Catch bugs, enforce style, and increase confidence in your code before you merge.

Explore GitHub Marketplace

Thank you. it is a goo change. I will merge with master branch

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

ProTip! Add .patch or .diff to the end of URLs for Git's plaintext views.

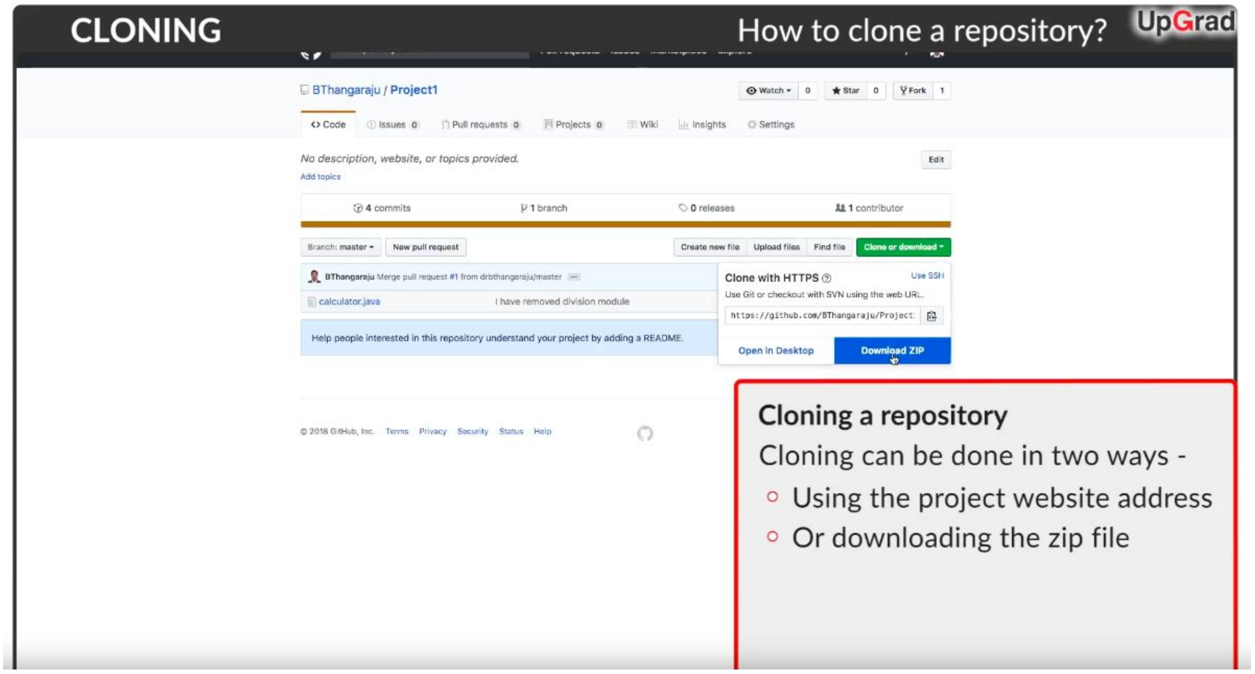
Reviewers	Suggestions: BThangaraju
Assignees	No one—assign yourself
Labels	None yet
Projects	None yet
Milestone	No milestone
Notifications	Subscribe. You're not receiving notifications from this thread.
1 participant	

Lock conversation

Cloning

In this segment we learn what cloning is?

Cloning means making an identical copy. This video will help you clone a github project and bring it to you local machine.



CLONING

How to clone a repository?

No description, website, or topics provided.

Add topics

BThangaraju Merge pull request #1 from drbthangaraju/master

I have removed division module

Help people interested in this repository understand your project by adding a README.

Use Git or checkout with SVN using the web URL.

`https://github.com/BThangaraju/Project`

© 2018 GitHub, Inc. Terms Privacy Security Status Help

Cloning a repository

Cloning can be done in two ways -

- Using the project website address
- Or downloading the zip file

We learnt that there are two ways of cloning

- Cloning from your github account
- Cloning on command line using command
  - git clone url(of git repository)

Staying in sync with a remote repository

In this segment you learnt how can you keep your local repository in sync with the remote repository

To do so there are two commands -

- git fetch
- git pull

The differences between them are as follows :-

git fetch	**git pull**
● git fetch will check your remote repository and retrieve any new changes made to the remote repository. The changes downloaded by **git fetch** is read only, which means you can only look at the changes and not modify the changes. ● Fetch is great for checking out any new changes that may have happened in a remote repository. ● Since git fetch only presents you with a "read-only" view of the changes, you can rest assured when using the fetch command. It will never manipulate, destroy, or screw up anything. ● Note: It is recommended to do a git fetch very often in order to know	● git pull is similar to git fetch In addition to downloading the new changes from the remote repository, it will also update your current HEAD branch to include the latest changes that have been made to the remote repository. ● This means that git pull not only downloads new changes; it also directly merge those changes into your current working copy files. **git pull = git fetch + git merge** ● Note: git pull might create a merge conflict

about the changes that have occured	

Keeping your forked repository on github in sync with the remote repository

To keep your fork up to date follow the steps below-

- Checkout to the branch you wish to merge to. For example if you want to merge to master do a -
    - **git checkout master**

- Pull the desired branch from the upstream repository. This method will retain the commit history without modification.

    - **git pull https://github.com/Username_of_owner/Name_of_original_repository.git branchname**
      [git pull [url of the remote repository that we forked from] <branchname>]

- If there are any conflicts, resolve them using your best judgement
- Make a git commit with the appropriate commit message
- Push the merge to your GitHub repository using the command
    - **git push origin *master***