Welcome to the course on 'Software Engineering Frameworks and Server Side Development'. Let's hear from Prof. Meenakshi on what you can expect to learn in this course.

Welcome to the session on ‘**Introduction to Version Control and Git**’

**In this session**

You will learn —

* That version control is a very powerful tool. Developers all across the world are using version control and Git specifically for all the good reasons.
* Version control has the power to solve almost all problems that you might face while working on a project.
* You will learn about the issues that you would face if you were not using version control for software development.
* One of the major scenarios where version control is used is —
  + Suppose you are working on a code, and after making many changes, you realize that you have really messed it up and now you want to revert to the last good version of your project. How would you do that without version control?
* You will also dive deep into learning what version control is and how it comes to our rescue.
* You will learn about two types of version control systems (VSC)
  + Centralized
  + Distributed
* You will be able to differentiate between centralized and distributed version control systems and conclude how and why using the latter is beneficial to us.
* You will learn about Git, which is a distributed version control system and why we prefer using it over all the other distributed version control systems
* Next, you will learn about GitHub and get to know that, Git != GitHub.
* In this session, you will do everything practically using the command line and see how files move.
* Lastly and most importantly, you will learn about one of the most important uses of Git.

Sounds intimidating? No, not at all! Leave all your thoughts aside and jump in!

# Why Version Control?

Before learning anything new, you should be clear in your head about a few things such as —

* Why should you learn it?
* What purpose is it going to serve?

Summary:

This video must have given you a decent idea of the utility of version control. To summarize,  
version control is a solution that allows you to —

* Revert to an older version of your project in case you think that you have messed things up
* Keep track of changes or additions to the project files by the various team members

*Some possible problems a team of programmers may face if they do not use version control are -*

* *Maintaining project code when it starts getting bigger*
* *If you mess up a code or the current version of your code may have some issues and now you  would like to revert to the last good version of your project*
* *If you are not maintaining copies of the various versions of your code then you will be in trouble*
* *If you are working in a project team, every time you work on the project, you should know exactly what has already been completed, added, changed and so on. How would you know who made the changes? Which files were changed?*

# Overview of Version Control System

Now you know that version control can help you solve most of the problems that you have been facing with managing and sharing your files.

Surely, you would want to know what a version control system is and how it works. What happens in the background to the files, and how do they move?

Let’s now find out what a version control system is and how it works, that is the mechanism that goes behind it.

Note:-

**Version Control**is a system that records or keeps track of the changes to a file or set of files over time so that you can recall or go back to specific versions later.

**Important:**

In the video, the professor is using **ls** command to list the stored files in the directory. It will not run in windows. Use the **dir** command in windows. It works the same way as **ls** works on mac

## ****vim Command****

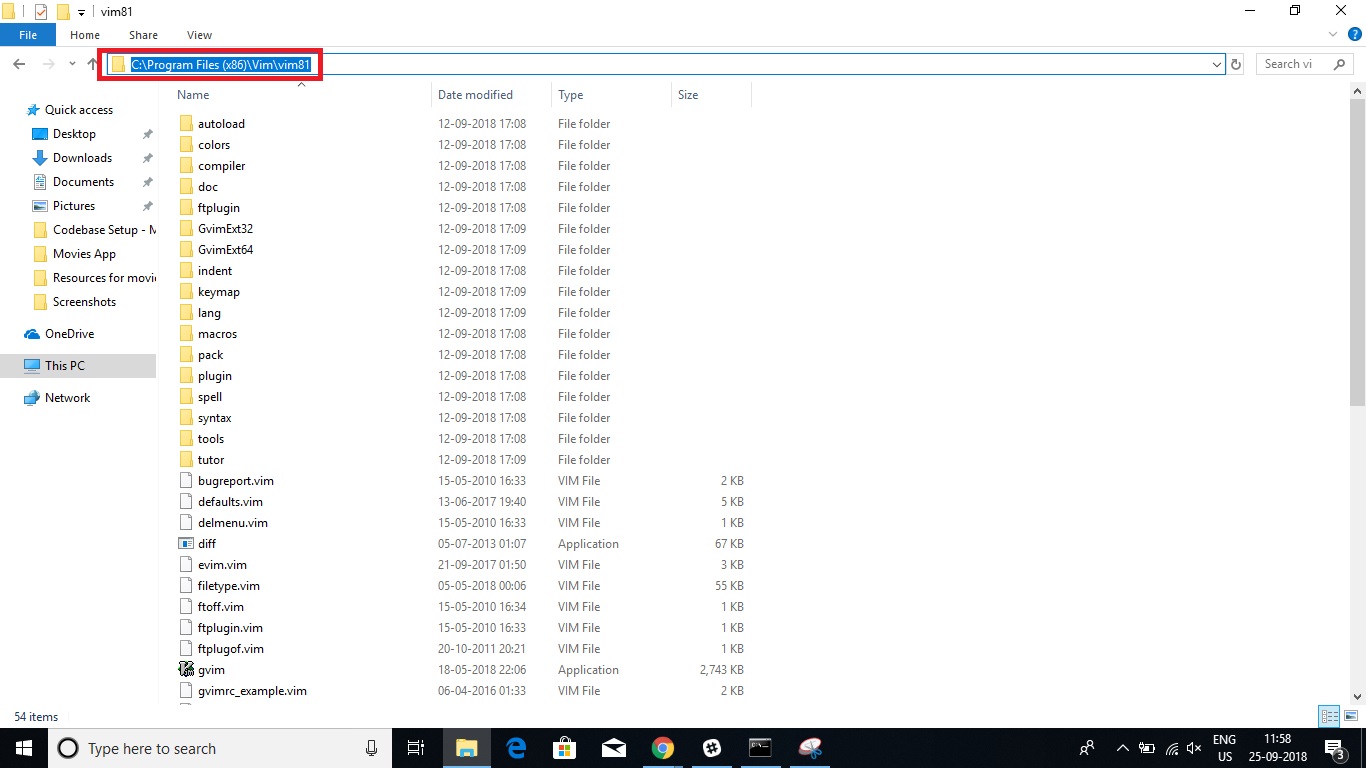
If ***vim*** command is not running on your pc then you will have to install vim on your system. Follow these steps to install vim:-

1) Download the installer according to your system (Windows, Mac, Unix, etc.). You can access the link [**here**](https://www.vim.org/download.php).

2) Install the downloaded file.

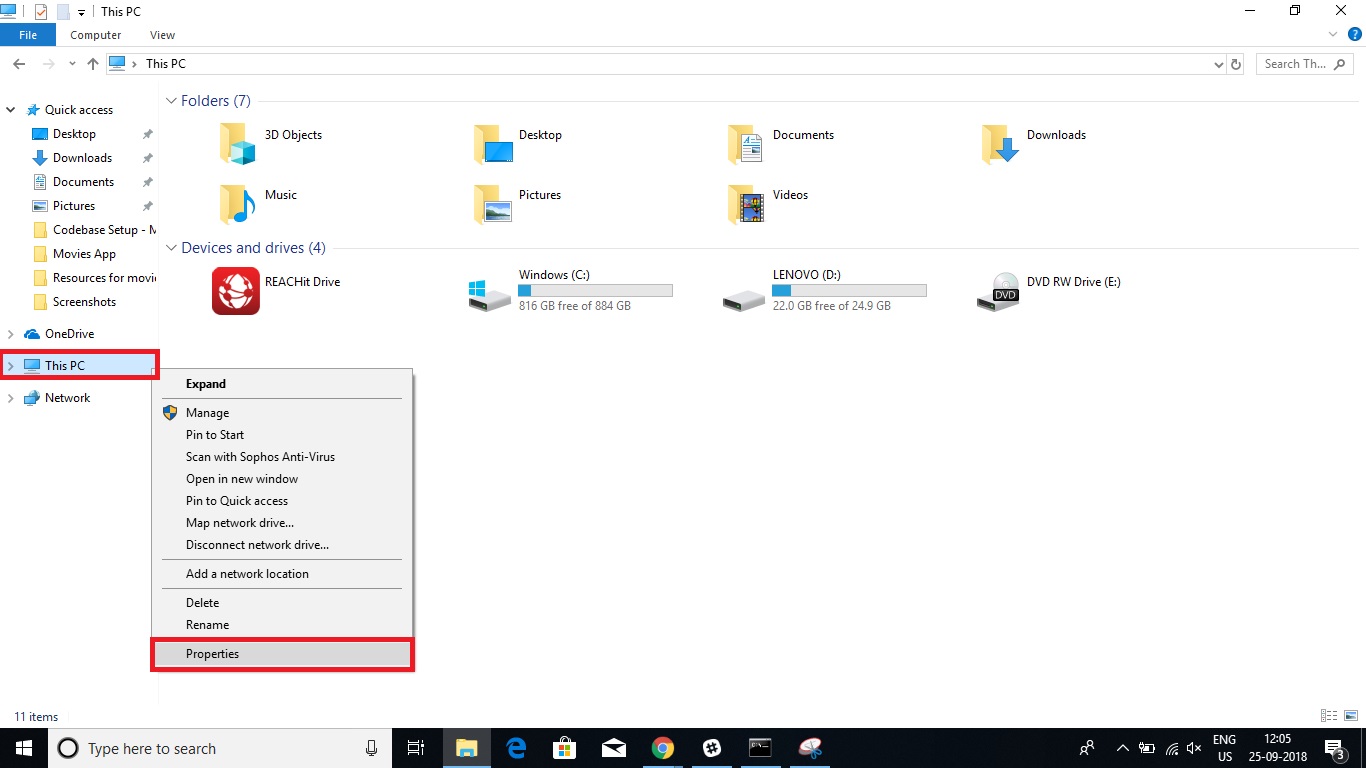
3) Now you will have to set the path variable as shown in the following points.

4) Copy path of the folder vim81 from the folder where you have installed vim.



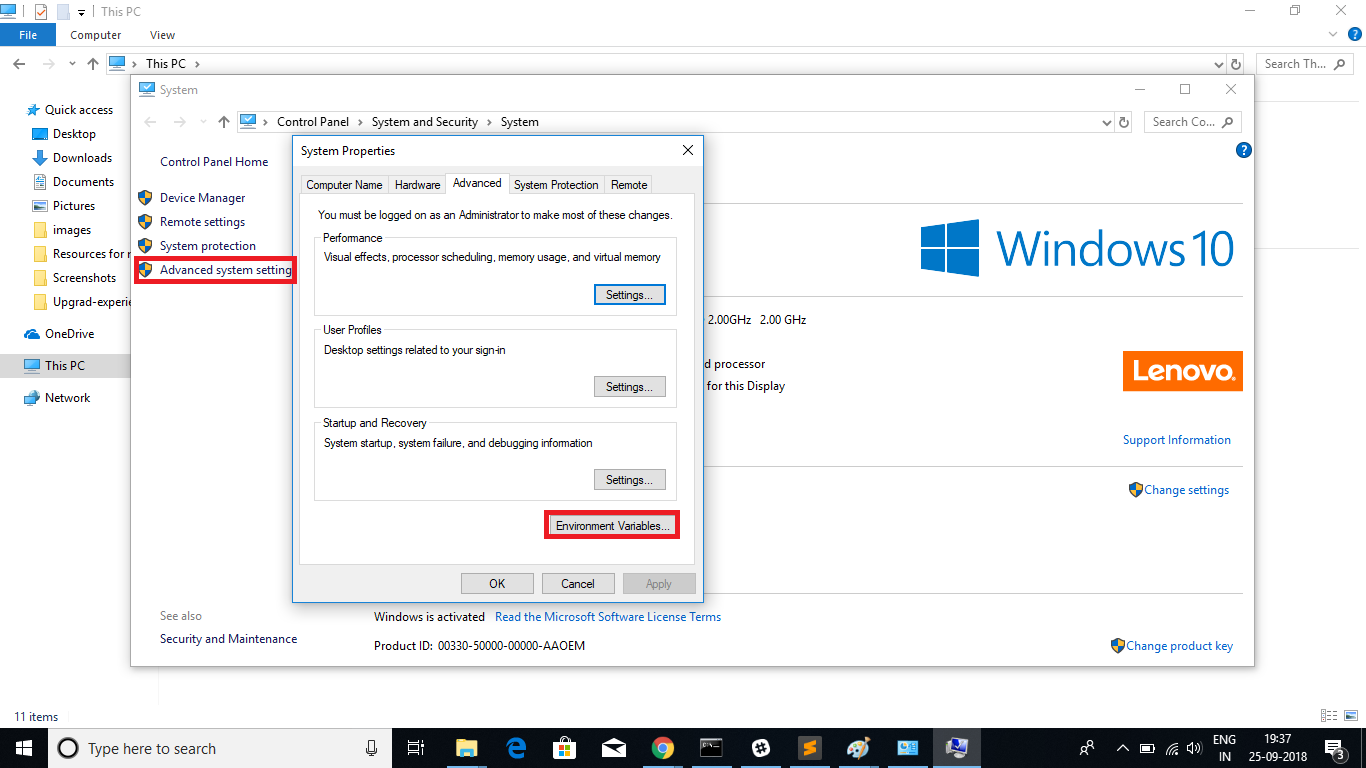
5) Set the path variable as follows.

6) Go to This Pc  properties

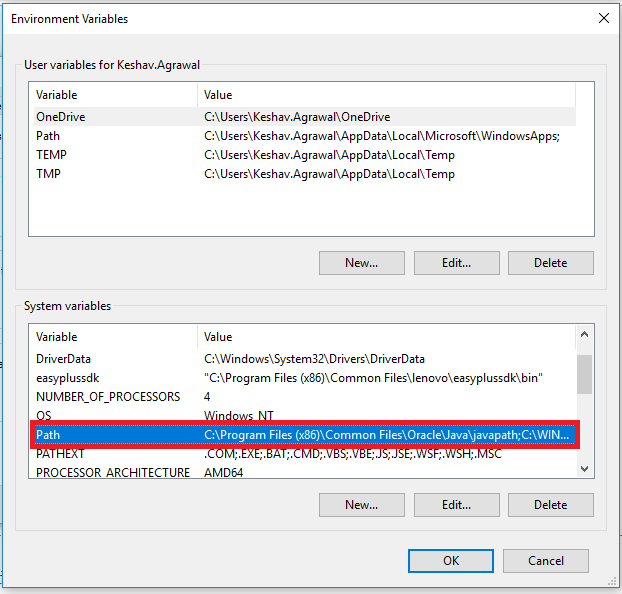


7) Then click advanced system settings from the left menu.

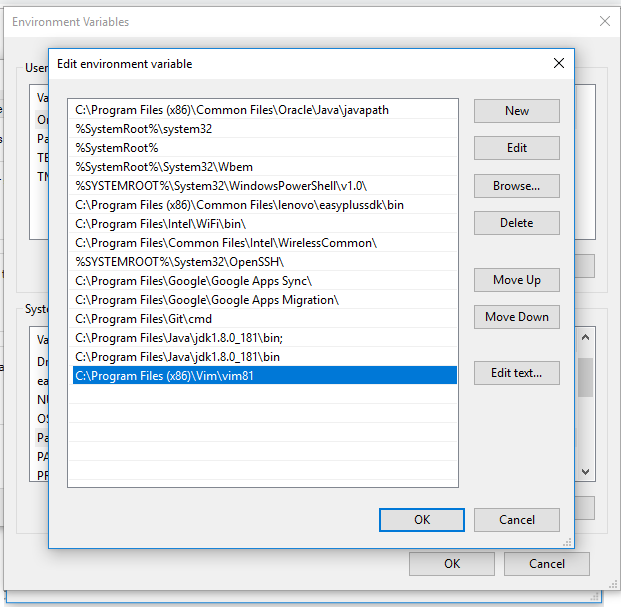
8) Now click environment variables.



9) Now, double-click on the path under heading System Variables.

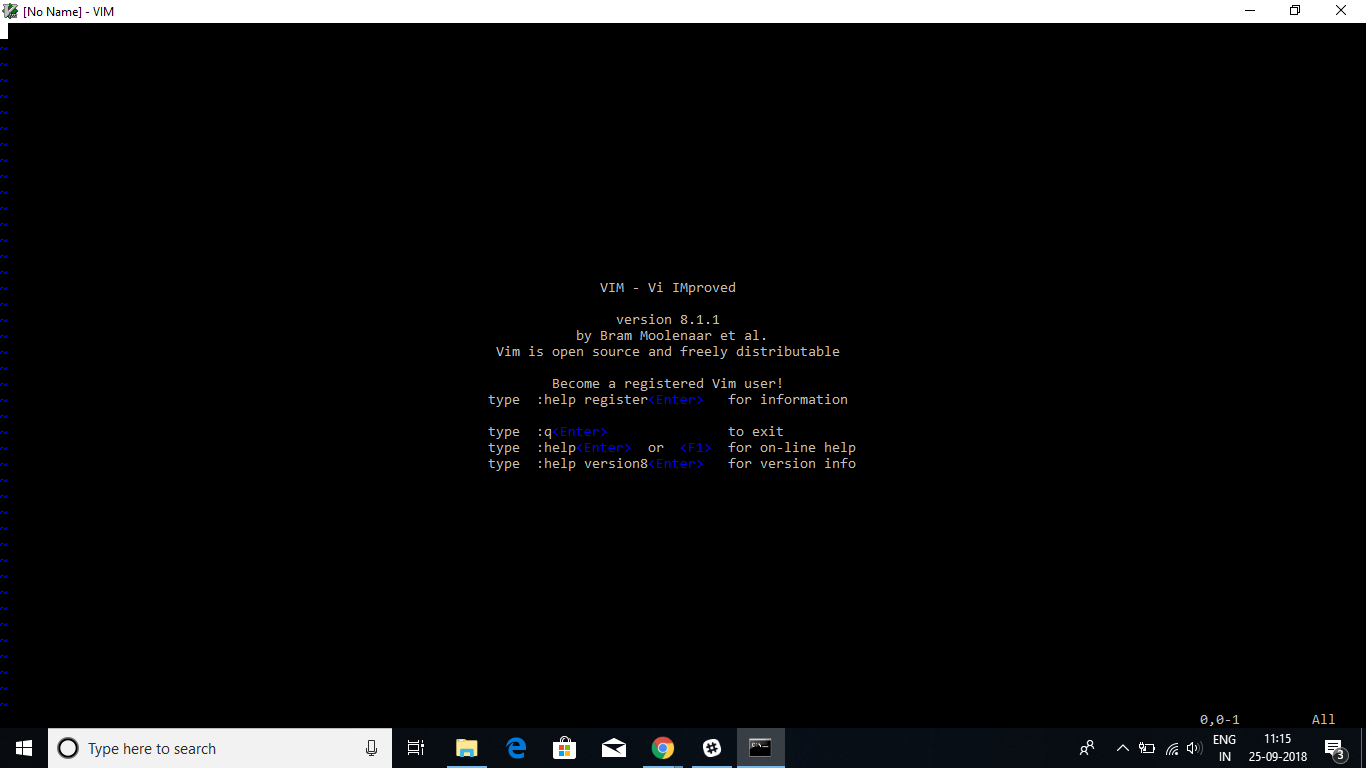


10) Click on new from right menu and paste the path that you copied in step 4.



11) Now click "ok" on all the windows open.

12)To check whether **vim** command is working or not, run command ***vim*** **-vim** in your command prompt and you will see this.



13) Boom! vim is now working fine on your system.

14) To go back to command window press "**esc"** followed by "**:"** followed by "***wq"***(to save the changes and exit) or just "***q"*** (to exit without saving) and then press ***enter.***

## Summary:

* Version control system is the management of changes to a collection of information such as documents, project code, etc.
* There are two major types of version control models, 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 file system, 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.

**Git Installation**

Before moving on to the next segment, make sure you have set Git up on your system. Steps for the same are mentioned below:

1. Git installation for Mac/Linux/Windows

(Git is generally installed on MacOS, but you can reinstall it to get the latest version.

* To do so, go to [https://git-scm.com/downloads.](https://git-scm.com/downloads)
* From the page that opens, download the software for Mac/Linux/Windows based on the system that you are working on.
* Next, go ahead and install git on your system by choosing all of the default options.
* Then, open your terminal/command line, and enter the command.
* **git --version**- This will show you the version of git installed on your system.

*Git is a distributed version control system and a tool to manage your project source code history. Git will help you keep track of the different versions of your project, but it tracks those changes locally on your computer; only you can see your project code and all the changes and revisions made to your project code.*

*GitHub, on the other hand, is a web-based, git file hosting service that enables you to upload your project code, along with its changes and revisions, so you can showcase/share your projects and files with others.*

*A repository is a storage space where all your files, their revision history, and anything related to the project can be stored. This can be either local to your system or in some storage space on an online host.*

Summary: Let’s summarise the learnings from this video:

* You learnt that using GitHub, you can share your file system like files, documents, etc. with others, access another 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 called Repository
* A repository is a directory that contains your 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 three steps your files may be going through internally. These steps are —

|  |
| --- |
| 1. Modified |
| 2. Staged |
| 3. Commited |

**A glimpse of the next segment:**

In the next segment, you will learn how you can make your local work shareable with other people. In technical terms, you will learn how to link your local repository with a remote repository on GitHub. Confused? Don’t worry, the next segment will answer all your curiosities.

*Files in GitHub can be in any of the following three states:*

* *Modified: In this state, modifications are made to a file or files, and the changes are still on our local system.*
* *Staged: In this stage, the changed files are added to the the staging area, which means the files will now become a part of your development history.*
* *Committed: In this stage, we make a record or take a snapshot of the files we have added to the staging area or our development history.*

*After staging your files, you can take a snapshot of your changes, and git will remember the changes. This snapshot is known as a commit. Commit is essentially a record of your progress in the project in the form of snapshots. Once you are done with the project, you can go back and have a look at these snapshots to revisit your progress during the project.*

# Making Your Project Remote

So far, you have learnt how your file moves from one stage to another in your local computers to GitHub. Now, don't you want to know how to sign up for a GitHub account and upload your code? Don’t you want to see things happening in real time? Sure you do! Let’s get going.

Summary: In this videoo, you learnt about the following topics:

1. Setting up your GitHub account
2. Creating  a repository on GitHub
3. Configuring git for the very first time
4. Making a commit
5. Pushing commits in your local repository from the command line to your repository on GitHub

**Basic git commands:**[**click here**](https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html)(These commands will help you throughout the course in case you forget anything)

Here are your takeaways:

* **First time git setup:**
  + For the first time git configuration, you use the following commands:
    - git config --global user.name "random”
      * Using this, you will enter your GitHub username
    - git config --global user.email “random[*@example.com*](mailto:johndoe@example.com)”
      * Using this, you will enter your GitHub username
* **Making a commit and pushing your changes to GitHub:**
  + Commands to be executed are —
    - **git init**
    - **git add filename**
    - **git commit -m “commit message”**
    - **git remote add origin <url of the remote repository>**
    - **git push -u origin master**
* **Some important git commands that are used very frequently:**
  + **git status**: This command will display the state of the working directory and the staging area. In other words, it lets you see the changes that have been staged and the changes that haven’t been added to the staging area.
  + **git log**: This command shows you the commit details. It lists out the commits made in the repository in reverse-chronological order, that is, the most recent commits show up first. It shows commits with the following details:
    - The commit ID or SHA
    - Author’s name (who made the commit)
    - Date and time
    - Commit message

**More on the ‘git remote add origin url’ command**

* + Using this command, you can add a new remote repository to your local repository. To do so, you should use the **git remote add**command on the terminal, in the directory your repository is stored at. **The git remote add** command takes two arguments:
    - A remote name, for example, origin (it can be any name)
    - A remote URL, for example, <https://github.com/user/repo.git> (the address of the repository on your GitHub account that you want to link your local repository to.)

**A glimpse of the next video:**

In the next video, you will execute all the commands from the command line. In the upcoming videos, the professor will execute step 6, which is —

Making modifications to your files and uploading those changes to GitHub

## Making your project remote Part -2

So, you have the tools, which in this case are the commands in your hand. In the following video, the professor will show you how to execute the git commands from the command line. You will also learn to make modifications to your files and add them again to GitHub.

All set? Brace yourself because now you are going to get the real feel of how developers across the globe work.

Video

Summary: Some important points to keep in mind:

When you are pushing your changes after the first commit, i.e. after —

1. Initialising your git repository and
2. Linking it with the remote repository on git

You only need to run the following git commands to commit any changes into your git              repository and push those changes to GitHub:

1. **git status**: Used to check which files are changed and not yet moved to the staging area
2. **git add filename or git add** : Used to add specific files giving the filename or using the ‘git add .’ command to add all the unstaged files to the staging area
3. **git commit -m** “**New commit message**”: Gives a new commit message and commits all the files sitting in the staging area
4. **git push -u origin master or git push**: Used to upload all the files and changes that were included in the most recent commit to your remote repository on GitHub

**Glimpse of the next segment:**

Oops! Messed up your code? No worries at all, git is here to your rescue. In the upcoming video, you will learn how you can go back to your previous commits, which is one of the most important uses of git.

What is the significance of **-m**in ***git commit -m*** command?

**Suggested Answer**

The -m lets you add a message to the commit which will be made/saved. The commit message can also be used to remind you of the changes you had made with a specific commit. Also, it is important to write good commit messages when you are working in a team, as commit messages would help other members of your team in understanding what files you have created or what changes you have made to them.

**git commit command**

After running the command ‘git commit’, will your file get pushed to the remote repository on GitHub?

*The git commit only takes a snapshot of our changes, and the changes remain on our local machines. Further, the changes are not pushed onto our remote repositories on GitHub at the time.*

Now you know that a git commit command will not upload your changes to your remote repository. Which of the following commands will push your changes from your local system to your remote repository on GitHub?

*The ‘git remote add origin url ’ will only link our local repository to the remote repository on GitHub for the very first time. However, ‘git push’ will push all our changes from our local repository to the remote repository on GitHub.*

The command ***git log*** will show you your files in reverse-chronological order. In other words, the most recent commits will be displayed first.

*The command****git log****will display a list of the commits you made, as well as its commit messages and commit IDs. In addition,****git log****will show the most recent commits first, i.e. commits will be shown in reverse-chronological order.*

# Accessing Previous Commits

Hey, ever faced a situation where your code is suddenly not working after you’ve changed it? Showing you errors? To make things worse, you don’t know where the bugs came in from? But you remember that the last time you saved the changes to your code, it was working fine.

Wouldn’t it be great if you could somehow go back to the last working version of your code and start from where you left off instead of from scratch?

Relax! Git is here to your rescue!

To use the command ***git checkout***, what would you require?

*When you use the command****git checkout****, you require the filename because the git checkout helps you to return to the previous version of your file/code/project only when the files are not staged and not committed. In addition to that, a file will only have a commit ID after it has been committed.*

Using which command you can go back to the previous changes you made? How will you get the commit ids for that command?

*We will use the command****git revert (commit id)****(In place of commit id paste the id of commit where you want to direct)**to go back to any commit we need. It will point us to that commit who's id you used in the command. We will get the ids by using the command***git log oneline .***This will give us the commit names and ids.*

Is the ***git revert***command is more powerful compared to the ***git reset***command

**False**

**Feedback :**

*The****git reset****command is more powerful than the****git revert****command, because —*

* *It has a host of options such as --soft, --hard, etc. and*
* *Once you use this command, it will remove the commit*
* *The****git reset****command changes which commit the HEAD points to (i.e. resetting will cause the HEAD to point to a previous commit instead of the most current commit with the bad changes. As the most recent comit will get deleted)*

*On the other hand  the****git revert****command —*

* *Adds a new commit to undo the changes.*
* *It does not delete any previous commit, just the HEAD is made to point to the new commit that undo the changes in the previous commit.*

**HEAD** - It is simply a pointer to the latest commit

Summary of the video:

* 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 the last point when your code was working, rather than starting from scratch
* You can return to your commits with the help of commit IDs or SHA
* You can get these IDs or SHA using the **git log**command or the **git log --oneline** command

In this segment, you learnt important git commands such as —

* **git checkout**-- **filename**: This command helps you when you have made some modifications to your file, say, file1, and you haven’t added those changes to the staging area or development history. Using this command will take you back to its previous state.
* **git revert commit\_id**: This command helps you when you have already staged your files and committed the changes and want to go back to your previous commit. Instead of removing the previous commit from the project history, git revert will figure out how to undo the changes introduced by the previous commit and appends a new commit that reverts the content and the changes introduced by the previous commit. This prevents git from deleting any commits, which is important to maintain the integrity of your revision history
* **git reset**--**hard commit**\_**id:**This command helps you when you have already staged your files and committed your changes, want to go back to your previous commit, and want to remove your present commit.Specifically, this command tells git to think of <commit\_id> as the latest commit in your history and revert any file to what they were at <commit\_id>. Any commit that you made after <commit\_id> will no longer be in your history. In other words, git will dispose of any commits that happened after <commit\_id> as if they never took place.

**Glimpse of the next segment:**

The next segment is a summary of all that you have learnt in this session so far with some assessments, which should be attempted on your personal systems.

Worried about what will happen if you mistakenly delete the file from your local system?

No worries git hub is here. You can bring back the file from your git repository using a simple command.

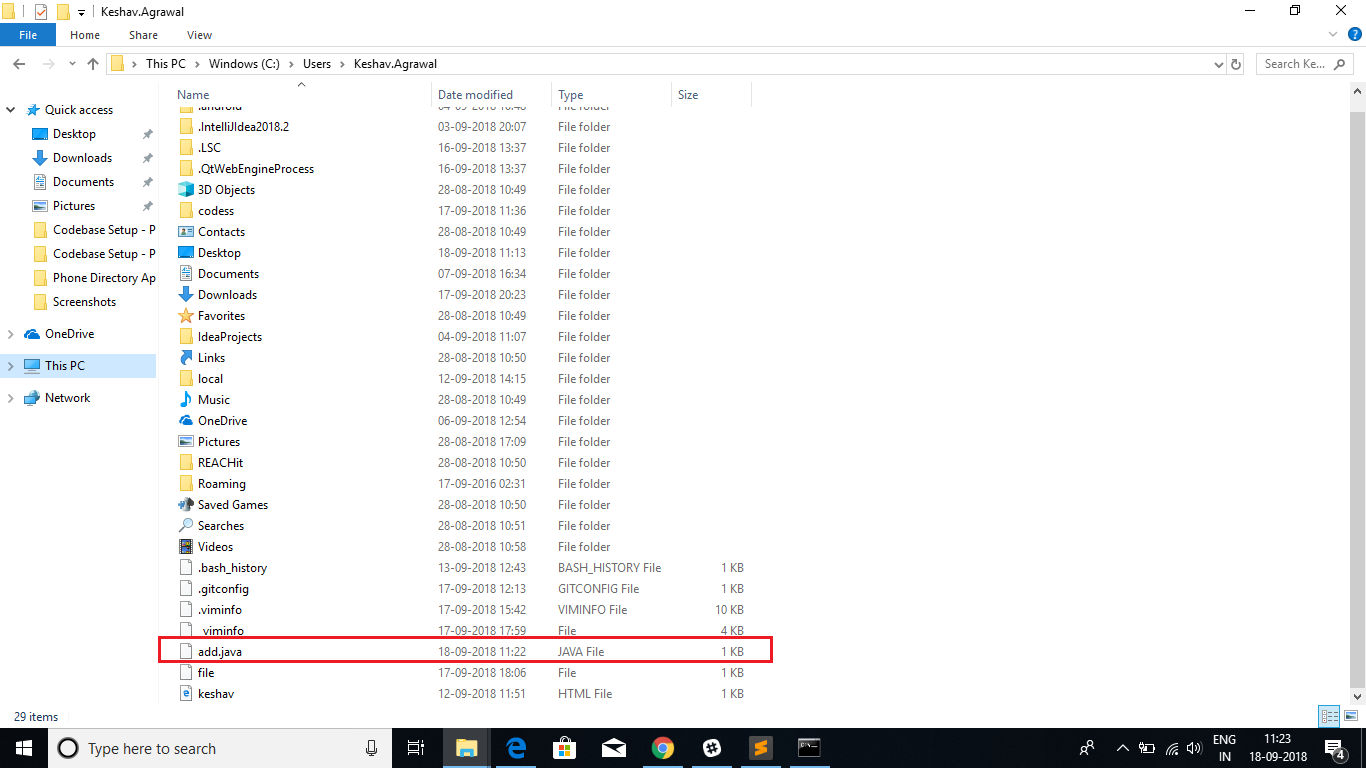
**git checkout**  it will show all the files available for checkout(in windows) then write the command

***git checkout (file name)***

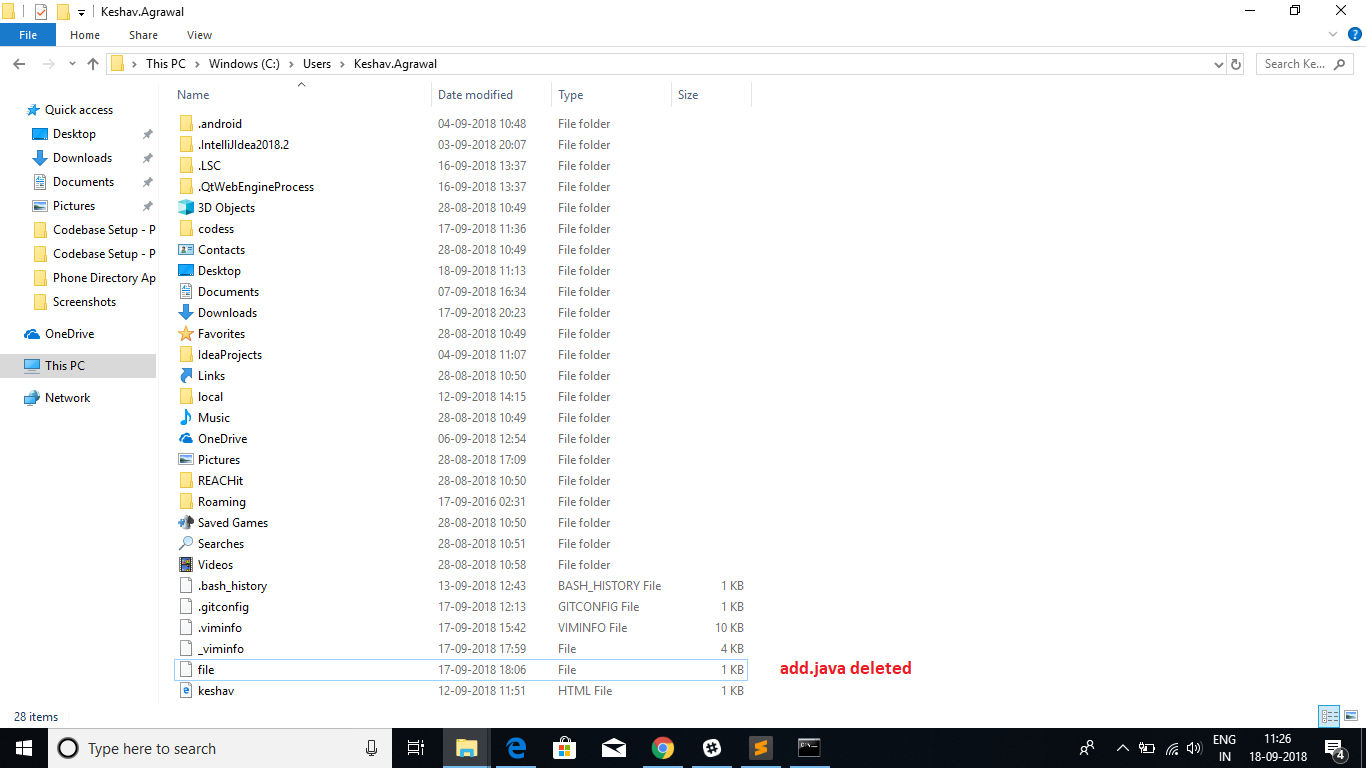
For eg. in our case, it will be

***Git checkout add.java***(Refer the screenshots below for better understanding).

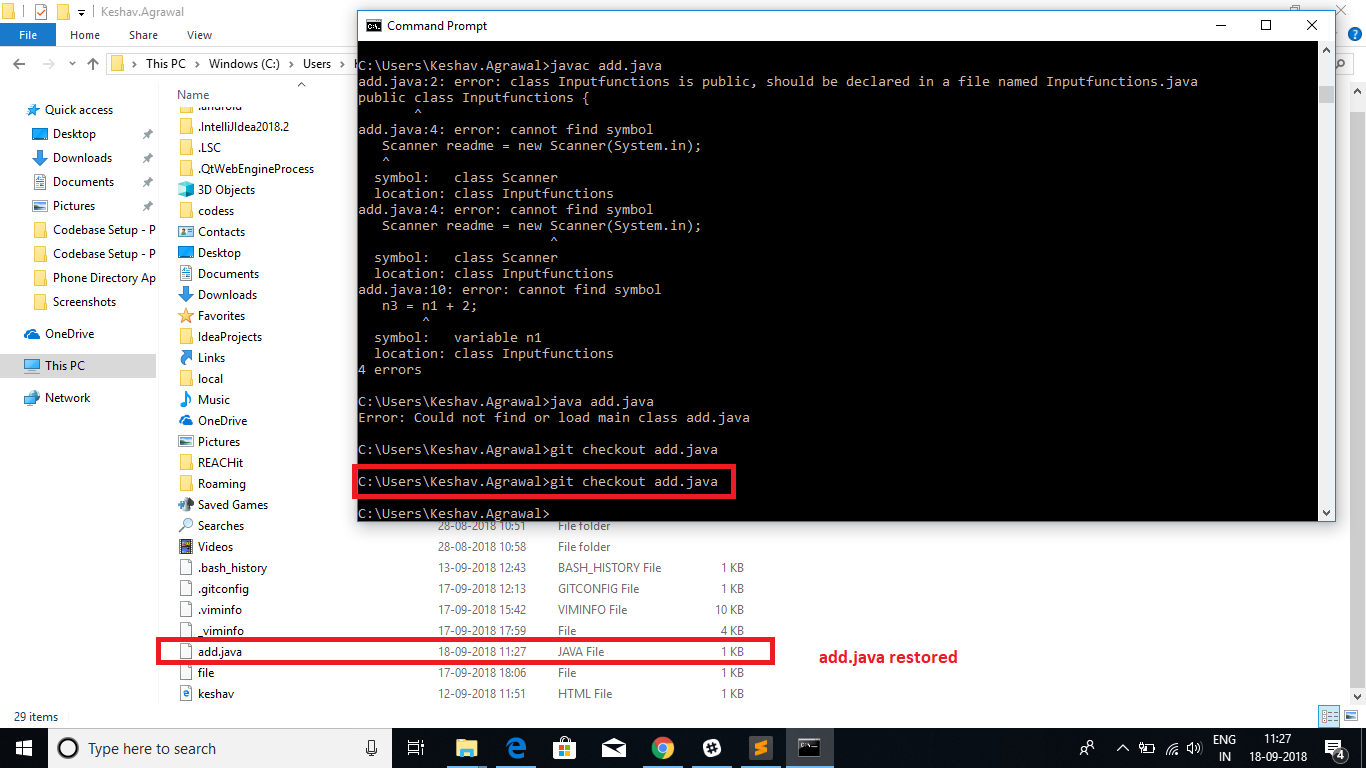
Delete the file **add.java**



File ***add.java*** deleted



Now run command ***Git checkout add.java***



Our file **add.java**is restored back in the folder.

# Summary

So, yes, git has true powers that can make your development life super easy. Git is a part of life for developers across the globe. It’s when you start using git that you realise that you’re addicted to it for all the good reasons.

So, in this session on the ‘Introduction to Version Control and Git’, you got the answers to the following questions:

* What issues would you face if you didn’t use version control to track the different changes and versions of your project files?
  + Some of the issues are —
    - Suppose that you are working on a code and after making many changes, you realize that you have messed up the code, and now you would like to revert to the last good version of your project. How would you do that without Version Control?
    - If you are to work with other developers, how would you and other developers coordinate the changes that you are all making to the project files?

You also learnt —

* How Version Control  comes to your rescue if you have a huge file and you want to keep track of all the changes in your file
* About the two types of Version Control Systems (VCS):
  + Centralized
  + Distributed
* About Git, which is a Distributed Version Control System, and why we prefer using it over all the other Distributed Version control systems
* About GitHub and how you can send your changes from your local system to your remote repository on GitHub
* How to use a set of commands on the command line to push all your code from your local system to your GitHub remote repository
* How to make changes to files and push those changes back to the GitHub remote repository again and what are the commands to do so
* Lastly, how you can undo bad changes and revert to previous versions of your project.
* *Note:*
* ***git add****: This command will add your file with a name, e.g. file1, to the staging area, that is, your file will now become a part of your development history.*
* ***git status****: This command will display the state of the working directory and the staging area. In other words, it lets you see the changes that have been staged and the changes that haven’t been added to the staging area.*

***git commit -m****“****my first commit****”: Git commit helps save files and its changes to git’s memory. Specifically, this means that all the  files in your staging area and its associated changes will be added to your development history. In other words, every time you make a commit to a git repository, git will take a snapshot of all the files added to the staging area (as well as the files’ changes) and save that snapshot to its memory.*

* *The command ‘-m’ is used for adding a message to your commit*

***git status****: This command will display the state of the working directory and the staging area. In other words, it lets you see the changes that have been staged and the changes that haven’t been added to the staging area.*

***Working tree clean****: This means that no modifications or changes have been made to the files that are being tracked by git*

# **Collaboration using Branching**

# Session Overview

## In this session

This session will cover the steps involved in Version Control and how it is a boon when —

* You want to work together with other developers as a team on the same project or the same piece of code
* You want to make additions/modifications to someone else’s project or code

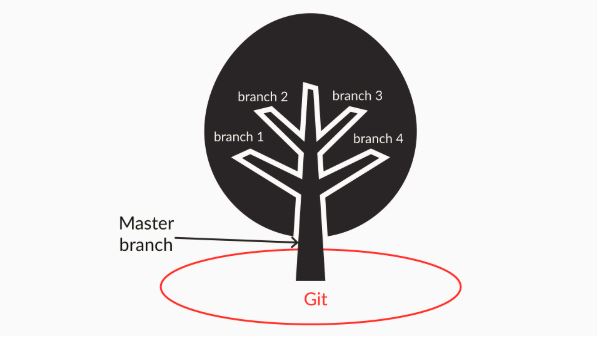
So, in this session, you will be basically introduced to the concepts of branching and collaboration.

For now, everything might sound ambiguous to you. But do not worry; you will have gained a fair idea of everything by the end of this session. Let’s jump in!

# Branching

When you come across the term branching, you might correlate it to branches of a tree.

Well, yes! Branching means exactly the same. Imagine the branches growing out of the trunk of a tree. This trunk represents git, as shown in the following illustration.



The trunk here plays the role of a master branch, and the branches coming out of the trunk represents the branches in git.

Let’s get started with the video to know more about branching.

What is the purpose of branching in Git?

**Suggested Answer**

It allows the user to create his own copy of the main branch(default branch in git), and work on it. Thus it enables the user to switch between his previous work(on his main branch) and his current work, keeping his current and previous work intact.

When you create different branches in git, does each branch get a copy of the original branch?

*As you’ve learnt in the previous session, git is a distributed version control system. So, when you create branches in git, each branch gets a copy of the original branch.*

Will git create a master branch for your project when you create a repository?

Top of Form



**True**

**Feedback :**

*Git will automatically create a master copy (or ‘branch’ in git terms) of your project when you create a repository. This master copy is called the ‘master’ branch.*

# Working with branches

You already know what branches in git are. Now it’s time to get some practical experience by creating and playing around with them.

**Branches in Git**

Which command would you use to create a branch named branch1 in git?

**Suggested Answer**

To create a branch named branch1 in git, the command git branch branch1 can be used.

If a team member is working on on branch x and he/she wants to move on to branch y, which command should he/she use?

**Suggested Answer**

To switch between branch x and branch y, he/she should use the command git checkout <branchname>, where ‘branchname’ would be y. You can also use the git command git checkout -b <branch\_name> <origin\_branch\_name> as it makes you aware of the branch you are making a copy of.

Summary: In this video, you learnt about the following steps:

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

In this video, you learnt about the following commands:

* **git branch <branchname>**: This command will create a branch with the given branch name
* **git branch**: This command will show you all the branches along with the HEAD pointing to the branch you are currently working on
* **git checkout <branchname>**: If you want to move from one branch to another, you can run this command

If you are working on a branch named ‘branch1’ and you want add a file named ‘caladd.java’ to this branch, will this file be added to the master branch as well?

*If you are working on a branch named ‘branch1’ and you want add a file named ‘caladd.java’ to this branch, this file will be added only to the branch named ‘branch1’, which you are currently working on. It won’t get added to the master branch as branch1 is a separate line of development.*

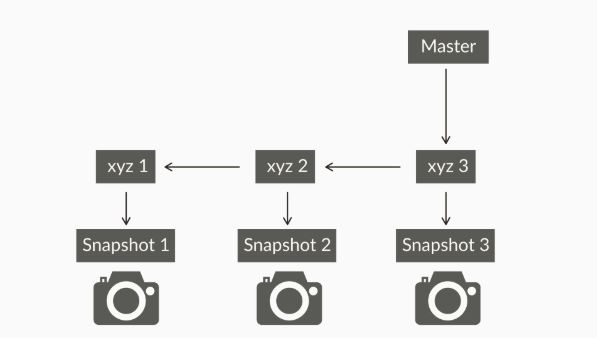
Why is it a bad practice to work directly on the master branch?

The master branch of a project should represent the 'stable' history of your code, which you would deploy to production or present to the customers. Therefore, you should not modify the master branch or consider carefully before doing so. Alternatively, you can use branches to experiment with new features, implement them, and merge them back to the master when they have matured enough.

If you want to parallelly develop the existing project code without making any changes to your master branch, then you can create different branches based on your need, and each branch will have the same copy of the master branch project source code.

**In-Depth Study of the Concept of Branching**

A branch in git is simply a lightweight, movable pointer that points to one of the commits. The default branch name in git is master, which points to the last commit. Every time you make a commit, the master moves forward to that commit.



Now if you create another branch, what do you think will happen? When you create a new branch, git will create a new pointer at the same commit you’re currently on.

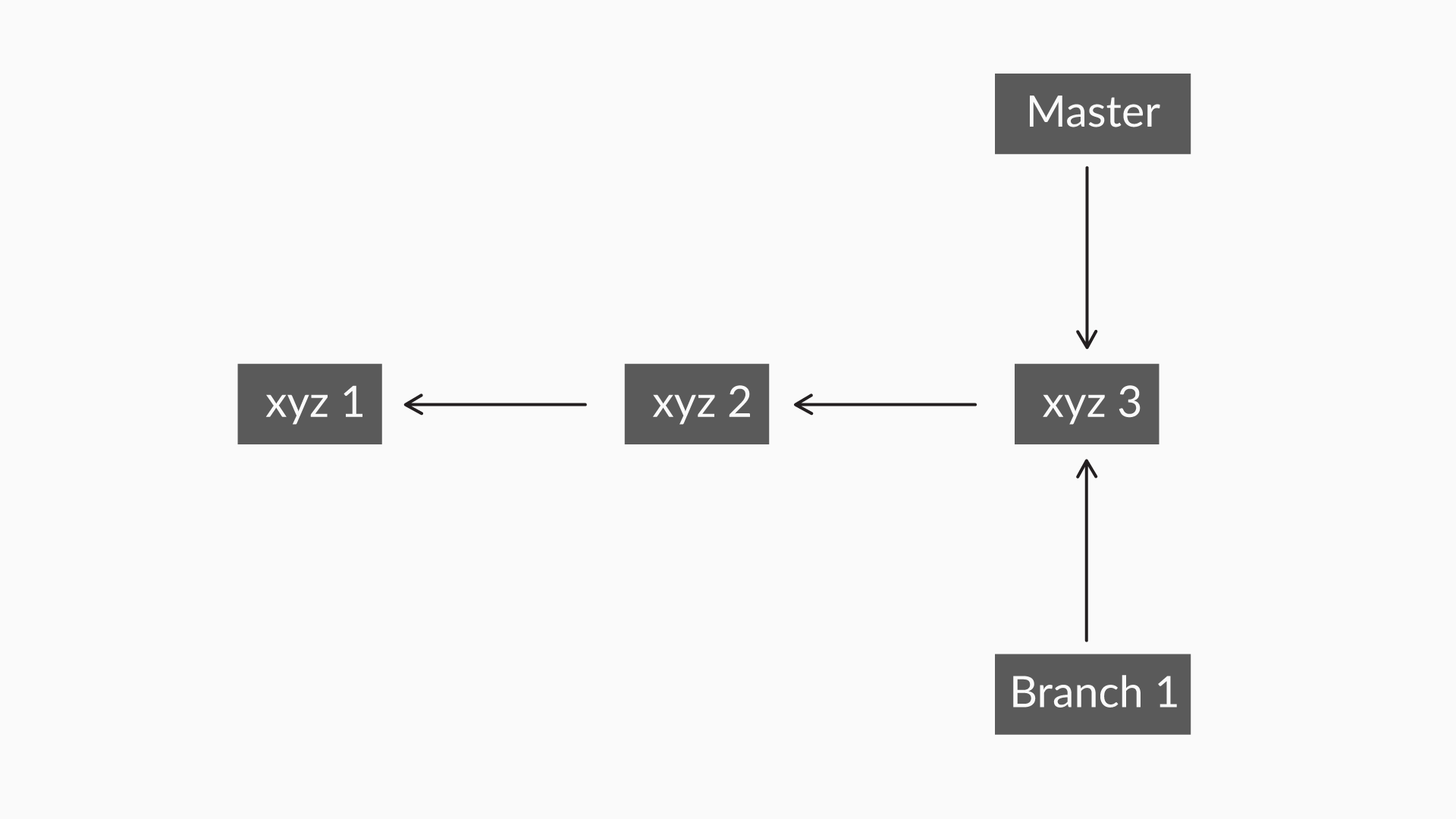


Figure 2: A new pointer at the same commit is created

Moving on, how do you think Git knows what branch you’re currently working on?

Git keeps a special pointer called the HEAD. It points to the branch you are currently working on. To make the HEAD point to the new branch, i.e. Branch1, you will have to switch to that branch — if it didn’t switch to that branch automatically.

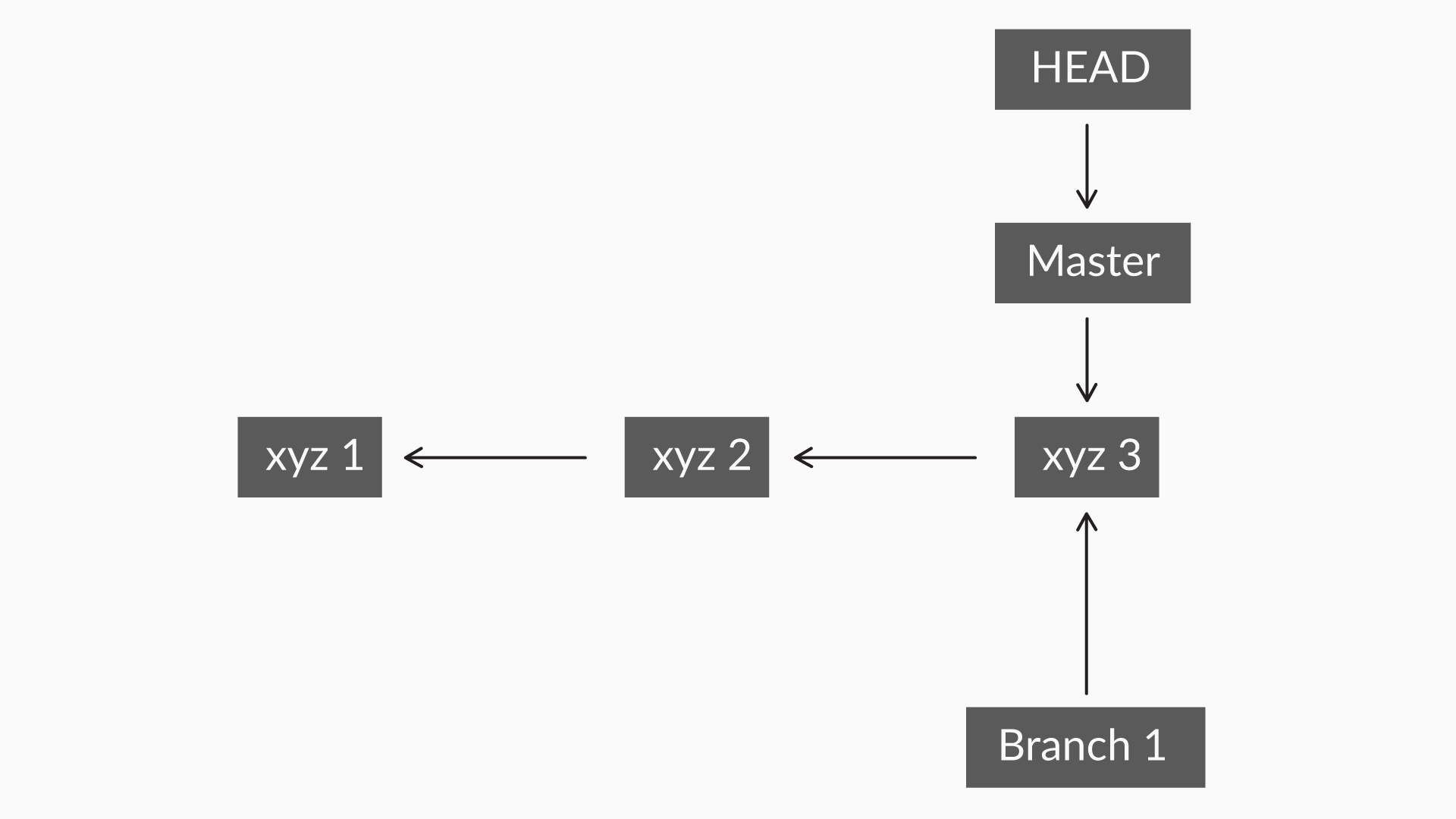


Figure 3: The HEAD points to the branch you’re working on

To switch to Branch1, you need to run the command:

**git checkout Branch1**

This will now move the HEAD to point to your newly created branch, i.e Branch1

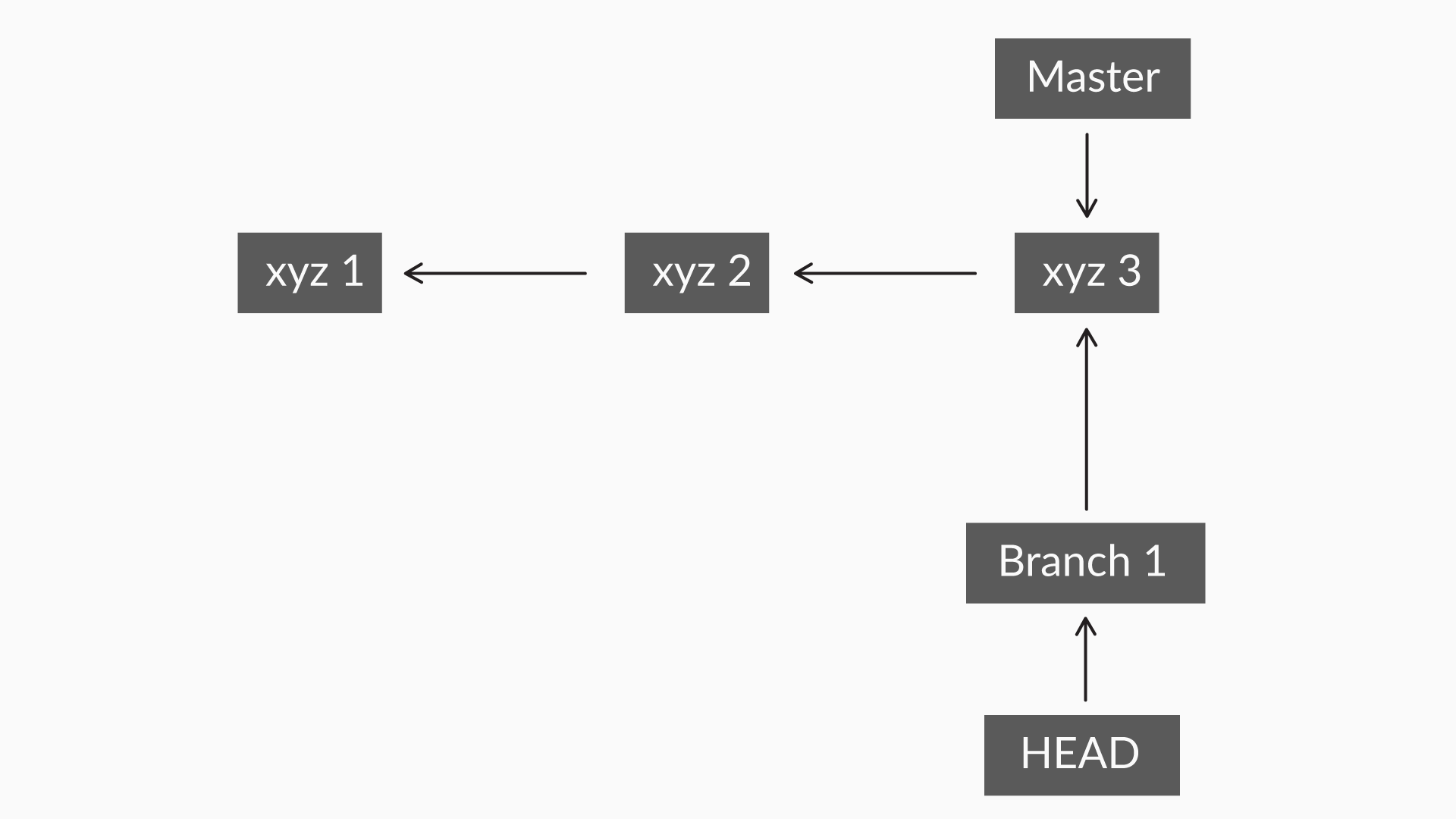


Figure 4: The HEAD points to branch1 when you switch the branches

Now that you know how to create, view, and switch between branches, you may ask yourself, “Is there anything else I should know?” Well, yes! How will you now insert all the changes you made to your new branch into your master branch?

To get the answer to this question, move on to the next video.

## Working with Branches - Part 2

Now that you have created your branch, you will get acquainted with the tools you need for synchronising your changes between your new branch and your master branch.

In this video, you learnt about —

* **Merging branches**: For merging, you can use the command **git merge <branchname>**
  + Note: This command will merge the changes in the branch <branchname> with the branch that you are currently working on. Merging can happen between all the branches. Imagine that you have three branches, namely —
    - Master
    - Branch1
    - Branch2

                    You can merge any one of the branches above with another one.

* **Deleting branches**: You can use the command **git branch -d <branchname>**

Let's learn more about the **git diff** command.

* The **git diff**command: This command is used to show the changes performed between commits. The main objective of version control is to enable you to work with different versions of the same file. Hence, git provides the command 'diff' to allow you to compare between the different versions of your files. The most common scenario where 'diff' is used would be when you need to see what changes you had made after your last commit. Ways in which we can use the 'git diff' command:
  + **git diff commitid1 commitid2**: To see the difference between two commits using their commit IDs
  + **git diff branch\_name1..branch\_name2**: To see the difference between two different branches. Here, 'branch\_name1' represents the branch you are currently working on
  + **git diff**: This will show you all the changes made to all the files and branches, all at once

**In-Depth Study of the Concept of Merging**

In the following image, notice that there are two branches connecting multiple boxes:

* The red line represents the master branch
* The grey line represents another branch named 'branch1'
* The rectangular boxes represent the commits and the different commit IDs

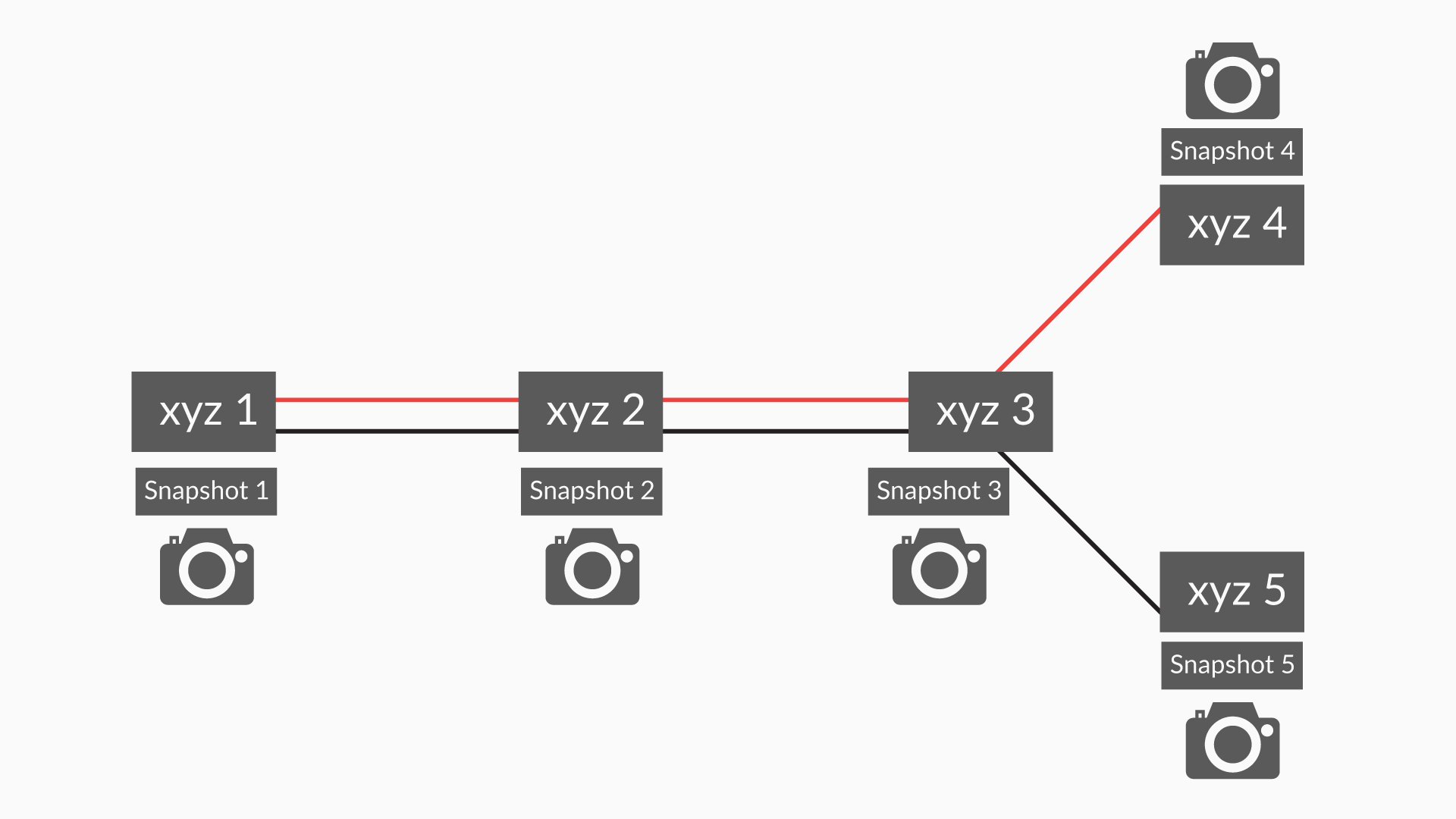


Figure 1: A depiction of git branches and commits

Now, suppose that you need to merge the commit 'xyz 5' on branch1 with the master branch. On merging of the two branches, it creates a new commit. At this point, git will take a snapshot of the changes made. After merging, the new commit would be named 'xyz 6', as shown in the following diagram:

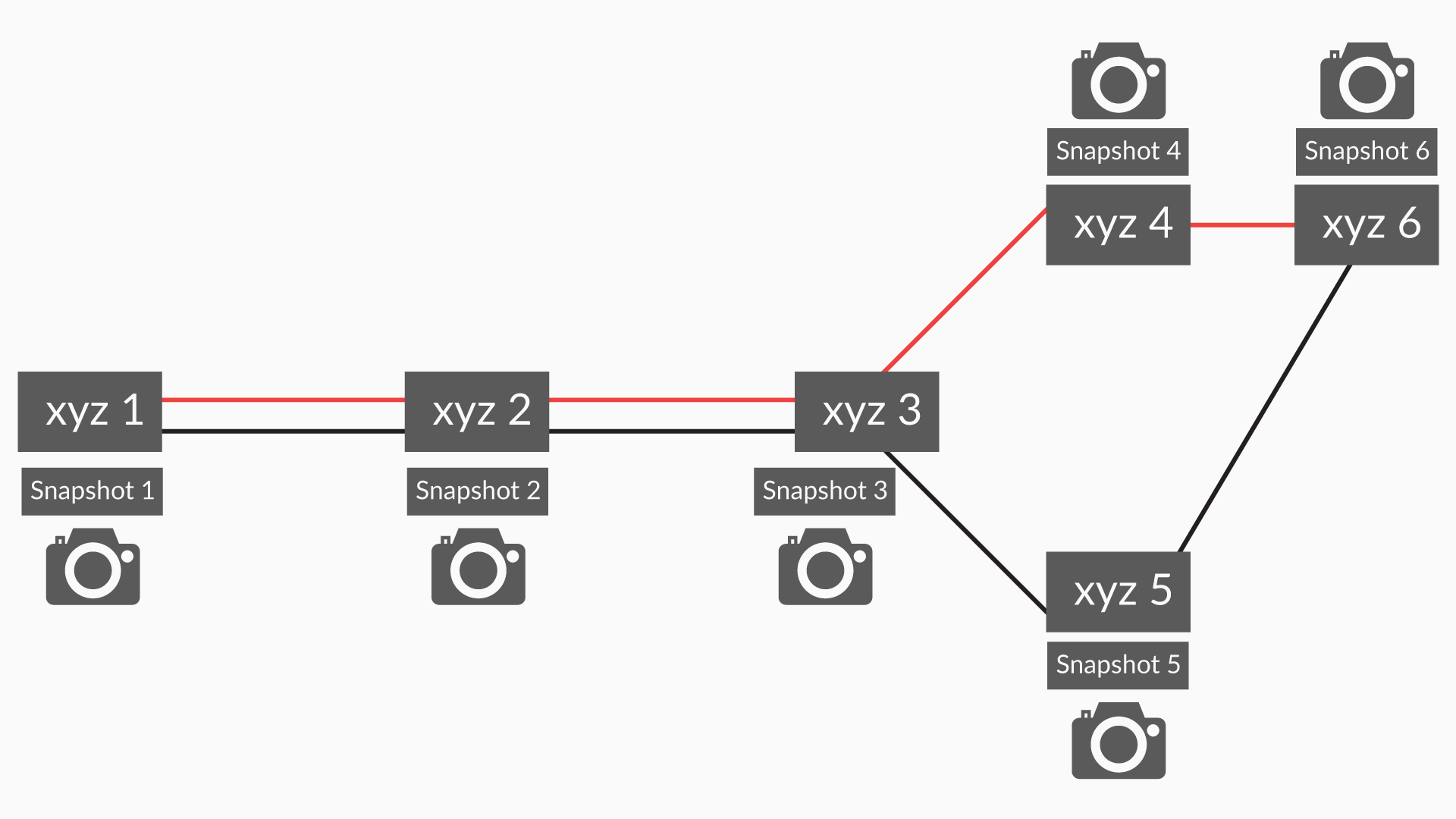


Figure 2: The git branches after merging

Now, once the commit 'xyz 5' is merged, if you want to work on 'branch1', you can keep making commits to it, e.g. commit 'xyz 7'.

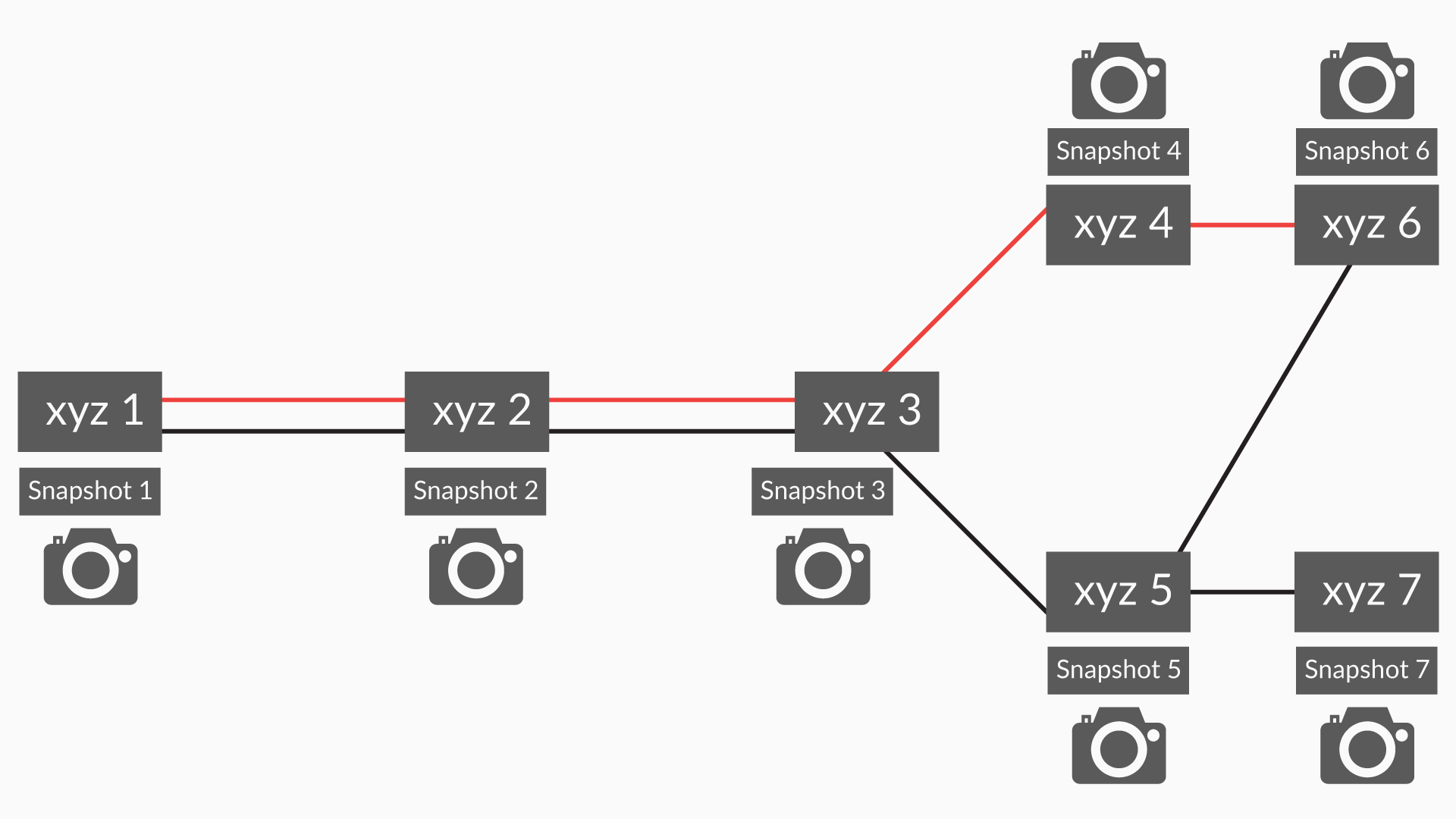


Figure 3: The git branches after the commit 'xyz 7'

**Glimpse of the next segment**

Happy merging! Oh, wait! Do you think merging would be as easy as what you just saw? You need to consider the fact that you may be working in a team, and two people might end up changing the same line of a code snippet. What happens then?

To know the solution to the above problem, move on to the next segment.

However, before moving on to the next segment, go through the questions below and answer them to test your understanding of the concepts explained in this segment through text, images, and videos.

What is the HEAD in Git? You can write your response in the box below.

*Imagine that you are working on the master branch and made a commit, say, x. The HEAD will now point to commit 'x' in the master branch. But, let's say that you made a new branch named 'branch1', switched to it, and made a commit on this branch; let's call the new commit 'y'. Then HEAD will now point to commit 'y' on branch1 instead. This is because the HEAD always points to the last commit on the branch you are working on and not the other commits made before that.*

What does the command 'git branch' do?

*The 'git branch' command will show you —*

1. *The number of branches in your repository and*
2. *The branch you are on. (The HEAD will point to the branch you are currently on.)*

# Managing Conflicts

Merging branches in git isn't always as easy as it may look. If two people change the same line of a code at the same time, don’t you think git will get confused?

**Summary**

In this video, you learnt how you could help git resolve merge conflicts. The best workaround to deal with a merge conflict is to use your best decision, that is —

* You can keep all the changes by making necessary modifications to the files where the merge conflict happens
* Or, simply discard some changes that cause the merge conflict

**Glimpse of the next segment**

In the next segment, you will go through someone else’s project code to improve it and make it even better with your ideas. Don’t you think that would be great?

What should you do in case of a merge conflict?

*In case of a merge conflict, you should use your best judgement and resolve the conflict by changing the specific lines of the code in a way that no two conflicting changes are made to the same line of the code. In case of a merge conflict, you should use your best judgement and resolve the conflict by changing the specific lines of the code in a way that no two conflicting changes are made to the same line of the code.*

How would you identify which lines of code in a file are part of a conflict?

**The conflicted lines are marked with <<<<<<<< HEAD**

**Feedback :**

*Right answer! You can identify conflicted lines by spotting them with****<<<<<<<< HEAD****.*

# Collaboration

Collaboration helps you work with the people around you and produce something even better than before. You can add features or improvise some of the features in someone else’s project who is sitting miles away from you.

**Repository in Git**

What is the difference between a local and a remote repository?

*Local repository is only available on the computer that it resides, i.e. its files, changes, commits, etc. cannot be seen by people using other computers. On the other hand, a remote repository is stored on Github (or any other Git hosting service), and its files, changes, commits, etc. in a repository can be accessed by other people with an internet connection. In other words, you would make changes to the project files, and commit those changes in your local repository. After which, you would push the commits in your local directory to a remote directory when you want to share those commits with other people.*

A fork is a copy of a repository hosted on a server like Github.

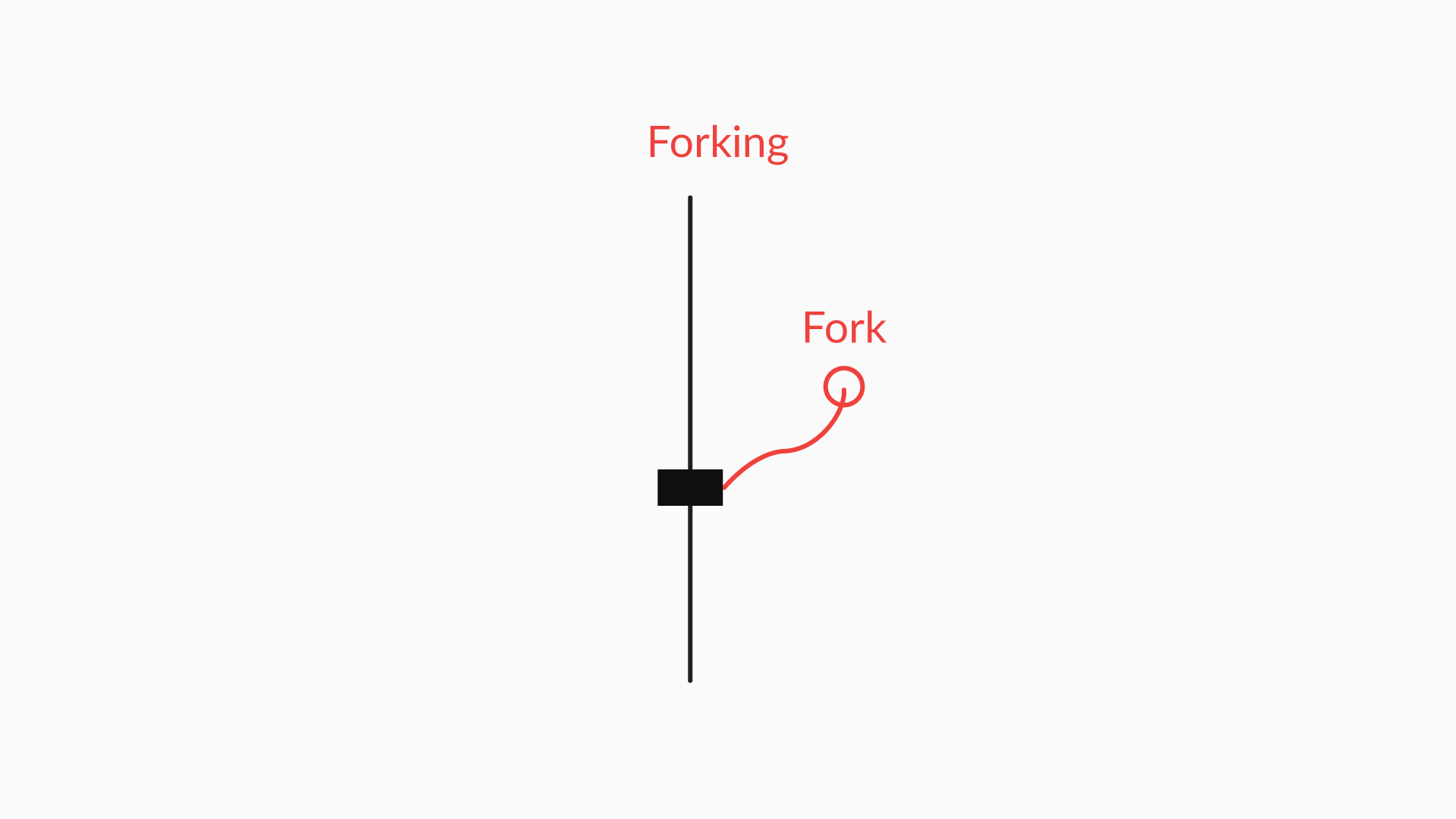
*A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.*

*Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.*

Here are the major takeaways from the video above:

* You learnt about a new term — **forking**
* To contribute to someone else’s repository on GitHub, you must first fork it. Forking will create a copy of that person’s repository, which is hosted on GitHub. Subsequently, GitHub will link that copy to your GitHub account as a remote repository. Once this is done, the copy will be owned by you.

Take a look at the image below to get a visual understanding of how forking works:



* However, a fork is a remote repository that resides on GitHub only. Now, to bring the fork back to your local system, you need to **clone** it to your computer. In other words, you need to download the fork from GitHub to your computer so that you can make changes to the forked repository that you created. You will learn more about cloning in the next segment.

**Glimpse of the next video**

Don't know how to fork? Jump to the next video.

## Making a Pull Request

In this video, the professor will teach you how to fork someone else's GitHub repository. You will also get familiar with how you can make changes to a forked repository, in addition to making contributions through pull requests.

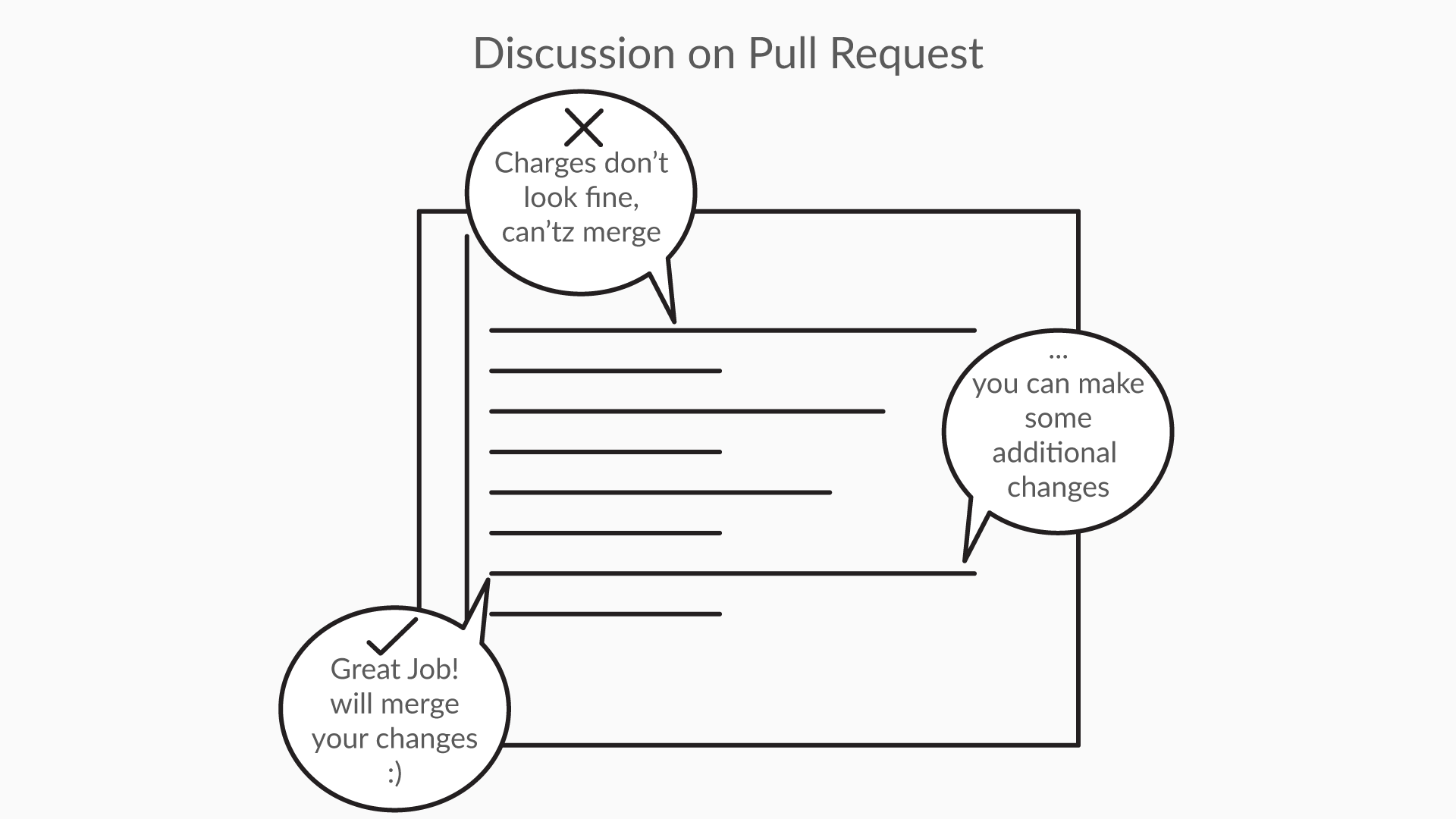
**Pull Request**

What is a pull request?

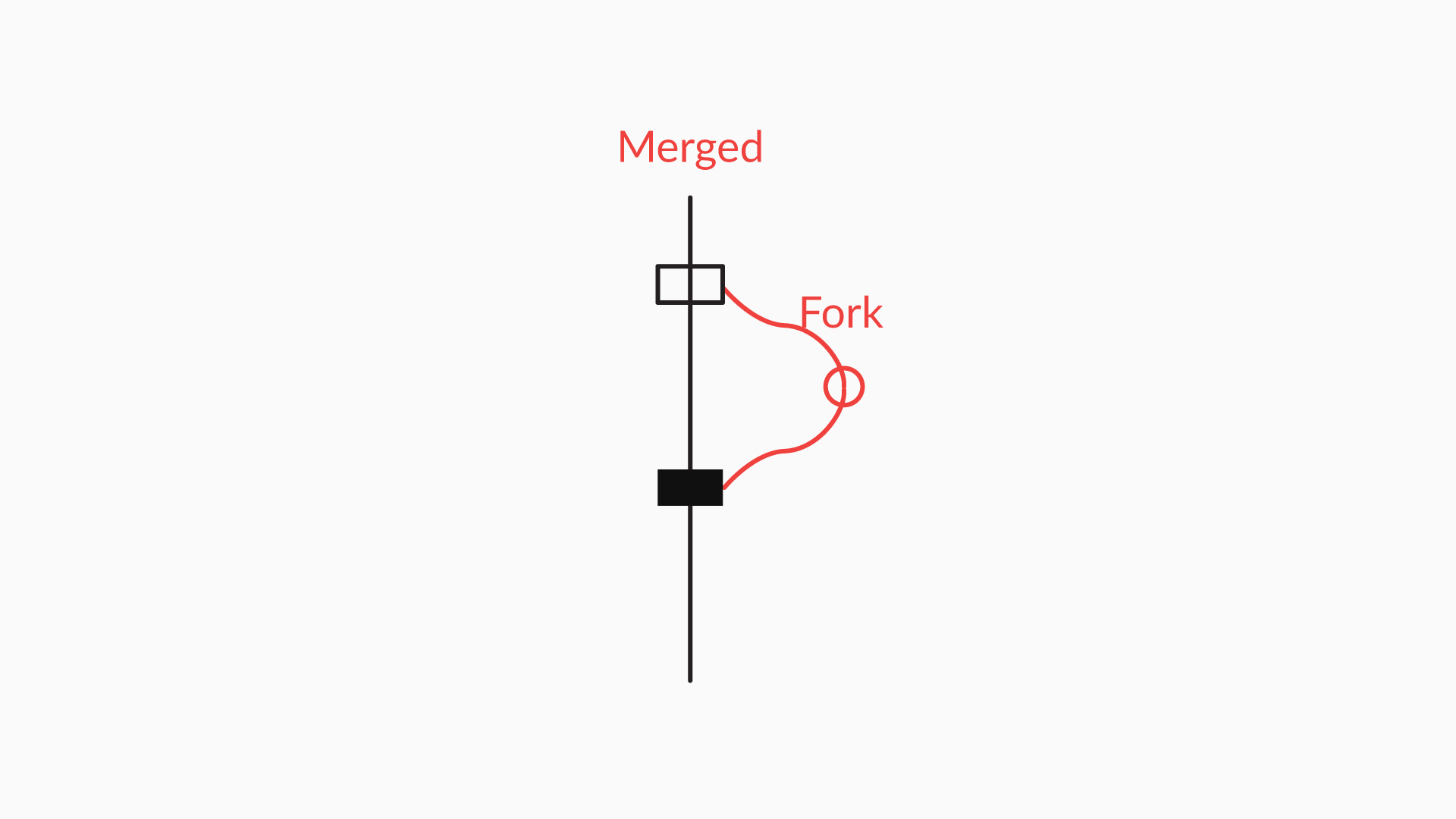
*A pull request informs the repository owner that you have forked their repository, and you have made some changes to the repository in your fork In addition, you would like for repository owner to merge the changes that you have made in your fork to their repository. Now It is up to owner of the original repository  to decide whether or not to “pull” changes from your fork and merge the changes back to their repository.*

Here are the major takeaways from this video:

* You will be able to see your desired repository on your GitHub account only after forking it on GitHub
* After a repository has been forked, you can open it and make changes to it from the GitHub site, as illustrated by the professor
* You can commit those changes from the GitHub site and then make **pull requests**
* A**pull request**tells the repository owner about the changes you've made
* In the next step, there may be discussions on your changes, and the owner of the original repository can accept or reject your pull request



* If your changes are valid, the owner will merge your changes



**Glimpse of the next segment**

Heard a lot about cloning already, right? Want to see how you can make a clone? Then the next segment is for you!

However, before moving on to the next segment, go through the questions below and answer them to test your understanding of the concepts explained in this segment.

Which of the following commands will create a new local Git repository on your computer, copy all the files, and commit the histories, messages, and branches from a desired remote repository to your computer?

*The 'git clone' command creates a new local Git repository on your computer, copies all the files, and commits histories, messages, and branches from a desired remote repository. There is no 'git fork' command in git. In the context of this course, forking is a GitHub-specific functionality that creates an exact copy of the remote repository and stores the copy as a remote repository on your GitHub account. In other words, the remote repository will be stored on GitHub and not on your local computers. Therefore, you need to clone the fork in order to create a local repository on your computer so that you can work on the fork.*

After making a pull request, which of the following might happen?

* A pull request can be merged if the changes are valid, acceptable, and appropriate
* A pull request can be rejected if the changes are invalid or cannot be merged with the original repository
* Users can give suggestions on the changes proposed by your pull request if additional modifications are needed, increasing the chances of your pull request getting accepted

# Cloning

By definition, cloning means making an identical copy of a remote repository on your local computer. This video will help you clone a GitHub project and bring it to your local machine.

What is a clone?

**Suggested Answer**

A clone is a duplicate copy of a remote repository  on our github account. When we clone a remote repository  we get a duplicate copy of it on our local machine as a local repository

A clone is same as a fork

**False**

**Feedback :**

*A fork is a copy of a repository. Forking brings a copy of repository from owners github account to your github account.*

*While clone is also a copy of a repository. Cloning will bring the copy of repository from your github account to your local computer. Or, on a high-level, a fork is a remote repository that resides only on Github, and a clone is a local repository that resides in your computer.*

**Summary**

In this video, you learnt —

* What cloning is
* What the two ways of cloning are:
  + Cloning from your GitHub account
  + Cloning from the command line using commands such as —
    - 'git clone url' (of the git repository)

**Glimpse of the next segment**

The next segment will walk you through some of the most important and high-level concepts in git.

However, before moving onto the next segment, attempt the assessment below.

**The 'git log' Command**

Use the command 'git log --oneline' and write down what do you see on your command-line interface?

Note: If you want to get out of the 'git log --oneline' screen, hit **:q** (type ':' and 'q')

**All of the above**

**Feedback :**

*On using the command 'git log --oneline', you should see —*

* *One commit per line*
* *The first seven characters of the commit SHA*
* *The commits message*

*Basically, you may use the command 'git log --oneline' to see the commit details in one line along with all other mandatory details of the commit.*

**The 'git log' Command**

1) Create a folder on your desktop named 'assessment'

(Note: You can create the folder anywhere on your system and not necessarily on the desktop only. However, you need to remember where you saved it to proceed with the following steps.)

2) Move into the folder named 'assessment' using the command 'cd Desktop/assessment/' from your command line

(Note: If your 'assessment' folder isn't located on the desktop, type in the address of the assessment folder after 'cd' on the command line instead of 'Desktop/assessment/'.)

3) Open the following GitHub account:

'https://github.com/upgrad-edu/Upgrad-experience'

4) Next, you will clone the project titled 'upgrad-experience' using the URL 'https://github.com/upgrad-edu/Upgrad-experience.git'

5) To proceed, copy the URL and go back to your command line

6) You should make sure that you are inside your 'assessment' folder

6) Run the command 'git clone [https://github.com/upgrad-edu/Upgrad-experience.git](https://github.com/upgrad-edu/experience-upgrad.git)'

7) Now, run the command 'ls'. You will be able to see the name of your cloned repository from GitHub — 'upgrad-experience'

8) Now, you have to move inside the folder named 'upgrad-experience'. To do this, run the command 'cd upgrad-experience/'

9) Finally, run the command 'git log'

Note: If you want to get out of the git log screen, type 'q' and run. To load all the commits press space till you see END on command line screen at the last

*Questions:*

**Filenames**

1. Move into the folder named 'Upgrad-experience' (this is the same folder that you cloned in the previous segment)
2. Create a branch with your own name using the command 'git branch'.  
   For example, 'git branch shreya'
3. Checkout to your branch using the command 'git checkout shreya' (replace 'shreya' with your own branch name)

       4. Now create a file named 'file1' using the command 'touch file1' (for Windows use echo. > file1.txt )

       5. Move into the file using the command 'vim file1'

       6. On the next screen, press 'i' once to enable writing options; now, type any text you want. For example, 'Hey this is my new file!'

       7. When you are done editing the file, hit the Esc key; then, you can type a colon (':'), which will appear at the bottom of the screen. Now, you can type in 'x' to save and exit from the file. To execute a command, press the Enter key.

      8. Run the command 'git checkout master' to switch to the master branch

      9. Run the command 'git merge shreya' to merge all the changes you made to the branch named shreya with the master branch.

    10. Now use the 'ls' command to list the files in the master branch

What do you see when you execute step 10 on the command line?

**Files named 'README.md' and 'file1' both**

**Feedback :**

*On executing all the steps mentioned above, you should be able to see two files on your master branch, namely, 'README.md' and 'file1'.*

**Q. The 'git branch' Command**

Now run the command*'*git branch'and write down the names of the branches that you see?

**Suggested Answer**

You should be able to see two branches, namely —

* master
* shreya

(Instead of 'shreya', you should be able to see the branch with your name on it.)

**The 'git branch' Command**

Now run the command 'git branch -d shreya' (here, instead of 'shreya' you should type in the name of the branch you had created). Next, run the command 'git branch'. From what you see on your screen, which of the following do you conclude?

**Suggested Answer**

The command 'git branch -d' should delete the branch you specified. And now you will be left with only one branch, that is, the master branch.

*Q.* **Pushing Changes to Your Fork**

When you are finished making changes to your local repository, which sequence of steps will you follow to push those changes to your fork on GitHub?

For example, imagine that you have created a fork of another person’s repo, and you have cloned the fork to your computer as a local repository. Now, you have made changes to the files in the cloned repository on your computer, and you want to push those changes back to your fork. Which of the following sequence of steps should you follow?

1) Commit the change

2) Add the file to the staging area

3) Push the changes to a remote repository

Order-2,1,3

A*fter making changes to the file, you need to first add the file to the staging area; then, you need to commit the file so as to make a record of it. Only after that, you can finally go ahead and push the file from your local repository to the remote repository representing your fork.*

*q.* **Branching in git**

Imagine that you are working on a branch named 'branch1' and you want to create a new branch named 'branch2'. From 'branch1', you want to navigate to 'branch2' and add the file named 'caladd.java' to 'branch2'. What would be the sequence of commands that you will have to follow?

1) git checkout branch2    
2) git add caladd.java  
3) git branch branch2

**3, 1, 2**

**Feedback :**

* *You need to first create a branch named 'branch2' using the command 'git branch branch2'*
* *Next, you need to move to the branch you just created using 'git checkout branch2'*
* *Lastly, you need to add your file on this branch using 'git add caladd.java'*

q. **Making a Pull Request**

What are the steps that you need to follow before making a pull request?

1. Clone the fork to your local computer' local repository
2. Fork the repository
3. Push the changes back to the fork
4. Make changes to the files in the local repository

**2, 1, 4, 3**

**Feedback :**

* *The first step would be to fork the remote repository you want to contribute to. A fork is a copy of a repository. Forking a repository allows you to freely experiment on it by making changes to it, without affecting the original project. Commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.*
* *Next, you need to clone the fork to your computer's local repository*
* *Then, you need to make the desired changes to your local repository*
* *Lastly, you must push the changes from your local repository back to the fork*

Bottom of Form