**Session Overview**

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!

Need of Version Control:

* *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?*

# Git and GitHub:

**Suggested Answer**

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.

**Repository in Git**

A repository is a directory where you can store and track all your project work, code, docs, files, etc.

# *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.*

The three steps your files may be going through internally. These steps are —

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

**Suggested Answer**

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.

# Connecting a project on your local system to a repository on git hub:

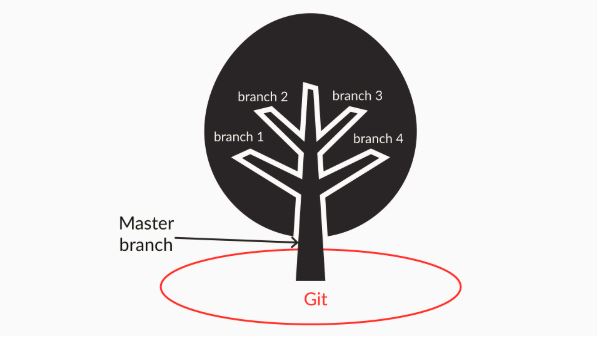
# 

# Branching:

# 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.

# 

Summary: Here is an important takeaway from this video:

* In this video, you learnt that there are situations where you may want to parallelly develop an existing project code, without making any changes to your initial/original branch. You can accomplish your goal by creating different branches based on your need (i.e. creating a branch per team member or a branch for every new feature) and each branch will have the same copy of the initial/original branch of the project source code.

**Glimpse of the Next Segment**

Don’t you think parallel development would make your life as a developer easy too? Yes, definitely!

In the next session, you will get to grips with making branches and playing around with them.

However, before moving on to the next segment, go through the questions below to verify your understanding.

**Branches in Git**

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?

Top of Form



Yes



**No**

**Feedback :**

*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.*

**Correct**

Bottom of Form

**Branches in git**

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

**!Note:**Once submitted, answer is not editable.



lightbulb\_outline

**Suggested Answer**

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.