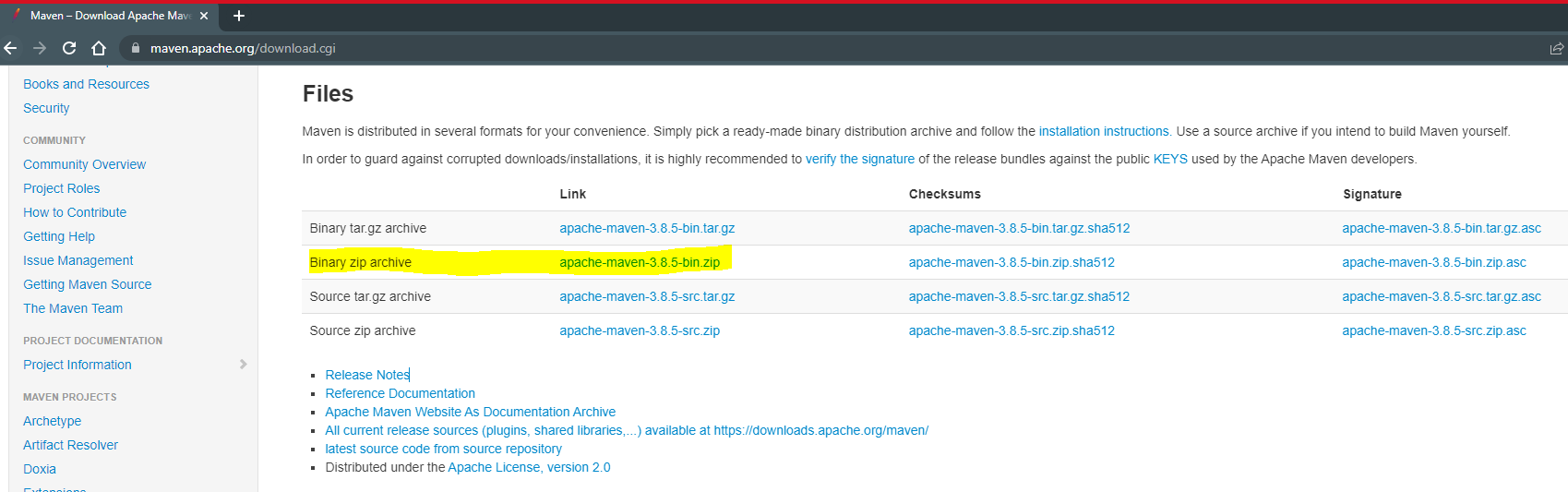
**Maven**

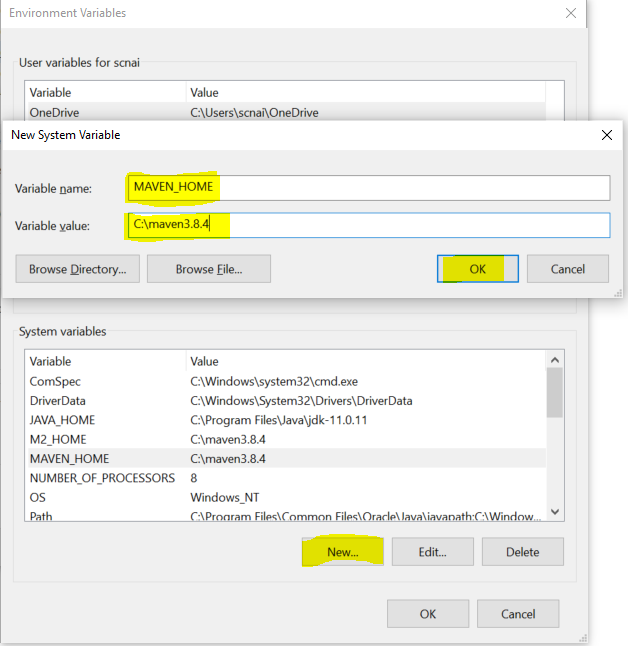
1. **Maven is a build tool**.
2. Gradle is another tool which is same as a maven and which is also a build tool.
3. Maven is use in the development process starting from the creating application till the execution of the application.
4. Maven **Archetype** is use to get the project structure of the application. It will provide you a project template.
5. Maven helps you to compiles the code (mvn compile).
6. Maven helps you to execute the unit test case of the code (mvn test).
7. Maven helps you to creating the project bundle (.jar, .war) (mvn package).
8. You can also run the project using Maven (mvn deploy)
9. Maven also helps to manage the dependencies (the jar files).

**Maven Setup**

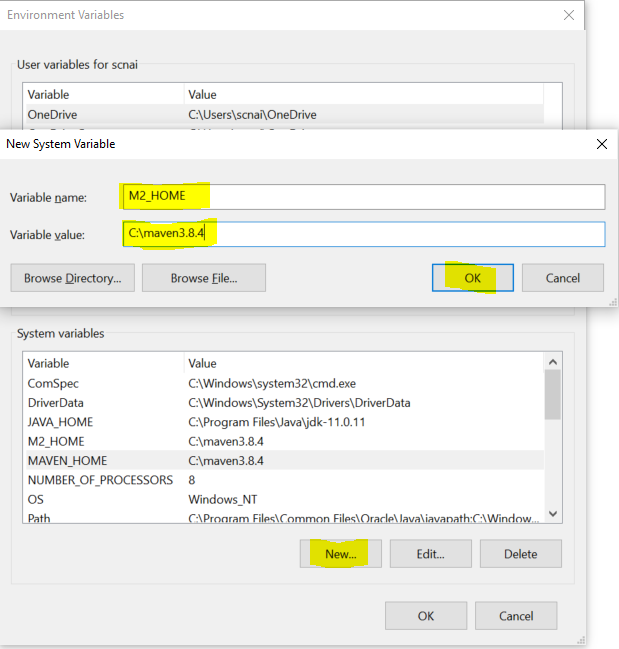
1. Download Maven Zip file (<https://maven.apache.org/download.cgi>)



1. Extract in into a specific location (prefer C dive)
2. Setting the environment variable for Maven
   1. **MAVEN\_HOME**

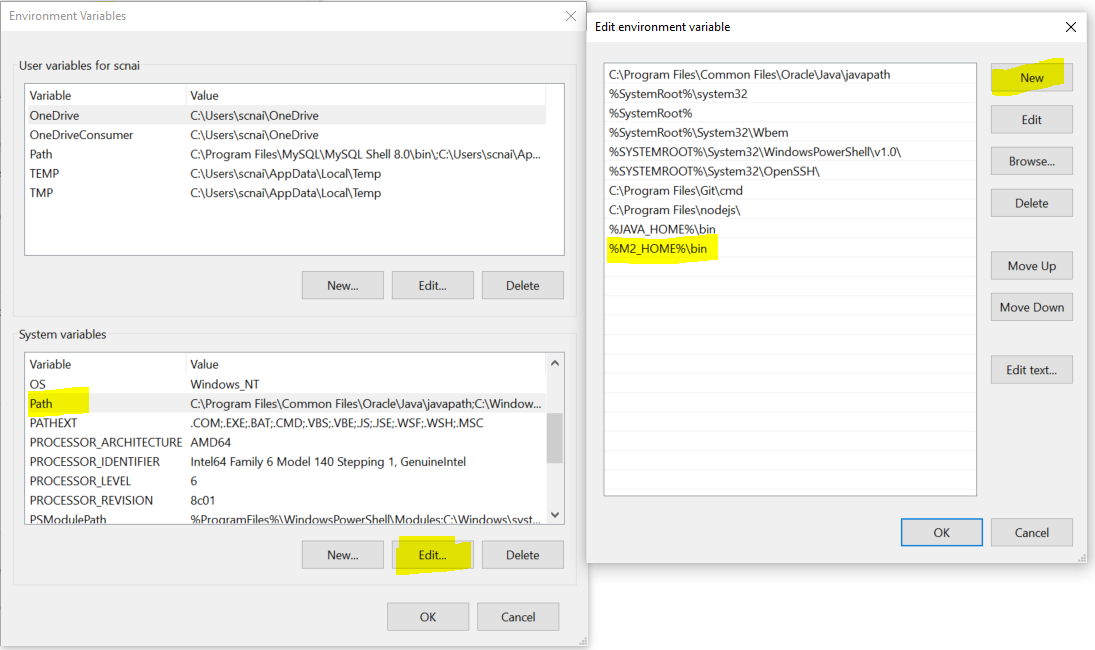


* 1. **M2\_HOME**



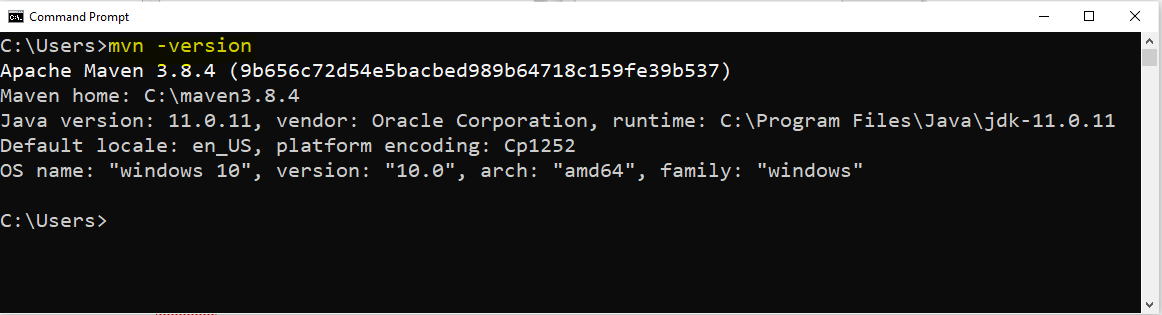
* 1. **Path**

Edit the existing Path Variable and add new value **“%M2\_HOME%\bin”**



1. **Verify Maven Setup**
   1. Open CMD
   2. Execute following command

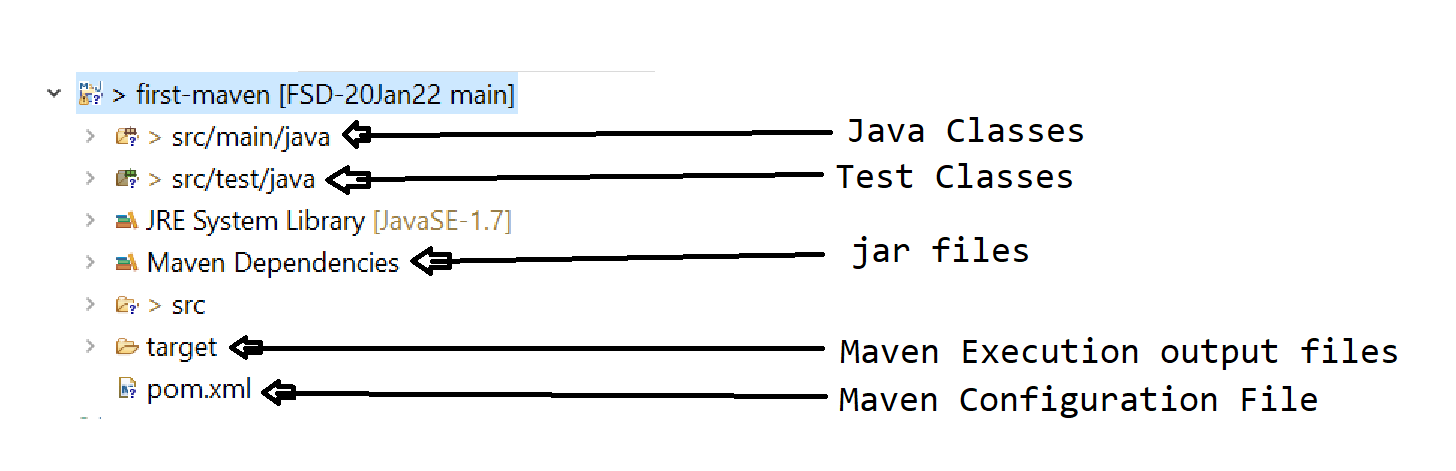
“**mvn -version**”



**Create Maven Project using IDE (Eclipse)**

1. File Menu -> “New” Option -> “Maven Project” option
2. Click on the “Next” button
3. Select an Archetype (Project Structure)
   1. There are Apache maven official and third party archetype available.
   2. You can use the official archetype provided by Apache by filter “org.apache.maven”
   3. **maven-archetype-quickstart**: You can create a Core java application
   4. **maven-archetype-webapp**: You can create a java web application
4. set Group Id (Package structure)
5. Artifact ID (Project Name)
6. Set the version
7. Click on Finish

**Maven Project Structure**



**Update Maven Project forcefully**

Follow the septs only if you are not able to get the changes after modifying the pom.xml or getting error on the project

1. Right click on Project
2. Go to “Maven” -> click on “Update Project”
3. Select the check box in the new window (Force Update of Snapshot/Release)
4. Click on “Ok” button

**Pom.xml file**

1. POM stands for Project Object Model
2. This is the maven configuration file
3. In this file maven setting, configurations and dependency management can done.
4. The basic configurations are as follows
   1. You can configure a groupId, artifactId, Version
   2. Using Properties tag you can set the java version or dependencies version.
   3. Dependencies tag
      1. This tag is use to manage the dependencies (jar files) into a maven project
      2. Can add, remove or update the version of the jar file using this tag
5. Maven has make all jar files available into a central repository.

<https://mvnrepository.com/>

**Dependency Management by Maven**



**Maven Life Cycle Stages**

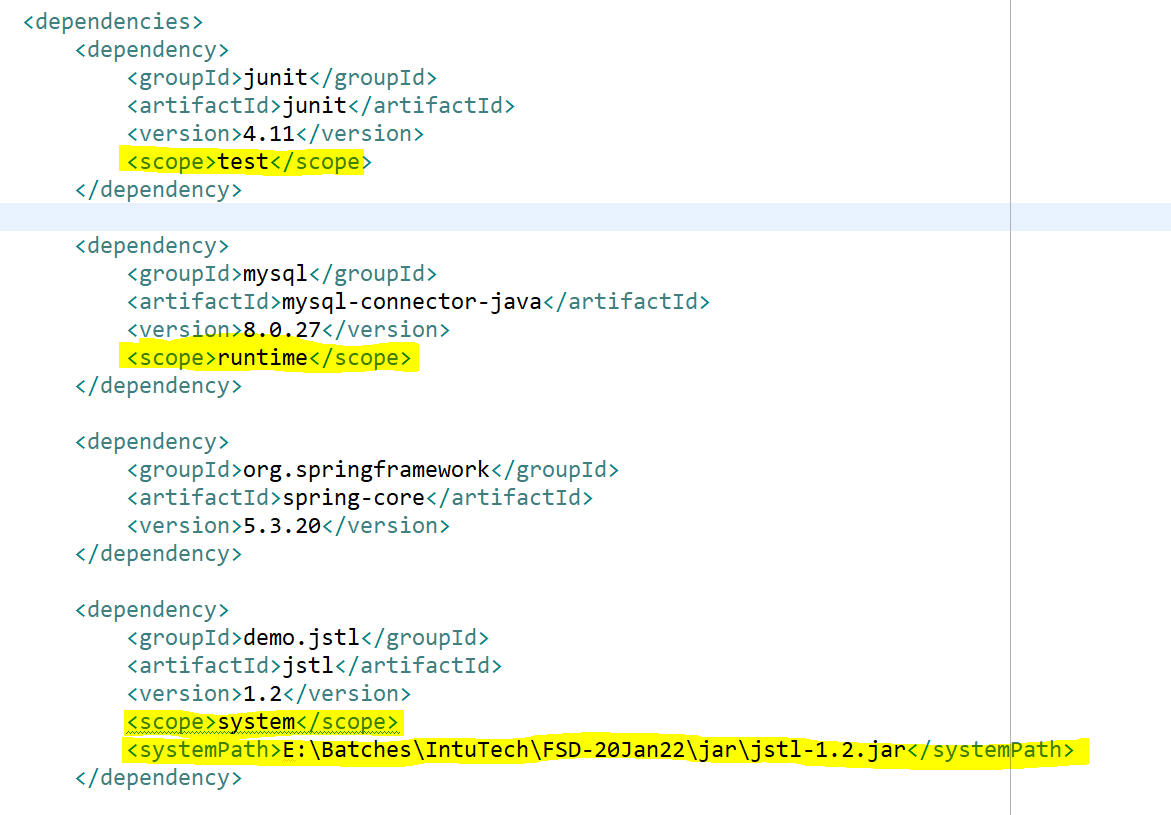
1. **Clean**: is use to clean the previous result executed by Maven.
2. **Validate**: This step is use to check the correctness, syntax and dependency of the project.
3. **Compile**: the java classes will be compile in this stage.
4. **Test**: The test cases will be executed in this stage
5. **Package**: In this stage the code will be packaged inside a jar or war files.
6. **Verify**: it will verify the project package whether it is correctly created or not.
7. **Install**: Prepare a package for the execution.
8. **Deploy**: Execute the application.

**Maven Scope**

This is the Dependencies (jar file) scope. Using scope you can specify till which maven lifecycle you need the specific dependency(jar).

There are 5 types of maven scope

1. **Compile:** the dependency added using this scope is available only till the compilation stage
2. **Runtime**: is a default scope if it is not explicitly set. These dependencies available in all the stage like code compilation and execution.
3. **Provided**: the dependencies with this scope will be search internally inside the project in jre or inside server.
4. **Test**: the dependencies available till the test stage of the application
5. **System**: the dependencies are present inside a system but not inside the project, so you can specify a path of your dependency manually.



**Project Versioning**

Version are Given in the following Format

**000.000.000**

Example :

Eclipse: 4.42.0

Java: 11.0.11

Major Release: First digits

Minor Release: Middle Digits

Bug/Defect fixes release**:** last digits

**Git and GitHub**

1. Git is a desktop tool which is use to maintain the files and its version etc. on local.
2. GitHub is a web application which is use to maintain the files and its version etc. on cloud.

Git Setup

1. Git has to install in local system
2. Download Git from : <https://git-scm.com/downloads>
3. Install Git on local system. <https://phoenixnap.com/kb/how-to-install-git-windows>
4. Verify Installation
   1. Right Click on any folder and check if you are getting **Git Batch** and **Git GUI** options.

GitHub

1. Create a Free Account on GitHub.
2. <https://github.com/signup>

**VCS tool (Version Controlling System)**

1. **Git is a VCS tool**.
2. VCS is use to manage the version of the files.
3. Using VCS tools you can maintain the version of the file.
4. Code can be integrated into a single location.
5. You can switch between different version easily.
6. Along with the version it will also maintain the metadata of the files.
7. There are 2 types of VCS tools
   1. Central VCS
      1. There is a single system there all the files and the MetaData (file details along with version)
      2. All the users use the same VCS tool to interact.
      3. TortoiseSVN is an example of Central VCS
   2. Distributes VCS
      1. There is a single cloud/on premises repository where all Metadata will be store
      2. This metadata will also be clone by the all the uses of repository.
      3. In this case if main system crash the you can recover the metadata from client system.
      4. Git is a distributed VCS

**Git**

1. Git can be use as a client and server both.
2. It has to be install on every client system.
3. You can create a local repository and also connect with cloud repository using Git.
4. Repository is a common location where all can store the files. The file Metadata will be maintain on repository.
5. To use Git you have to use command or GUI

**Git Command**

1. Create a local repository

**git init**

after this command the empty local repository will be created and .git folder will be created which is by default hidden

1. To check the git status

**git status**

Git maintain the file status

**Untrack File**

1. The files which is unknown to the git.
2. This file version and metadata will not be manage by Git

**Track File**

1. The file will be known to git
2. Git maintains the version and metadata of file.
3. Git maintains the different state of track files.
   1. New State
   2. Modified state
   3. Delete/Remove state
   4. Rename state

3. To add new or existing file

**git add <filename>** : can add single file at a time

**git add .** : can add all the files at a time

1. To get the file version and metadata

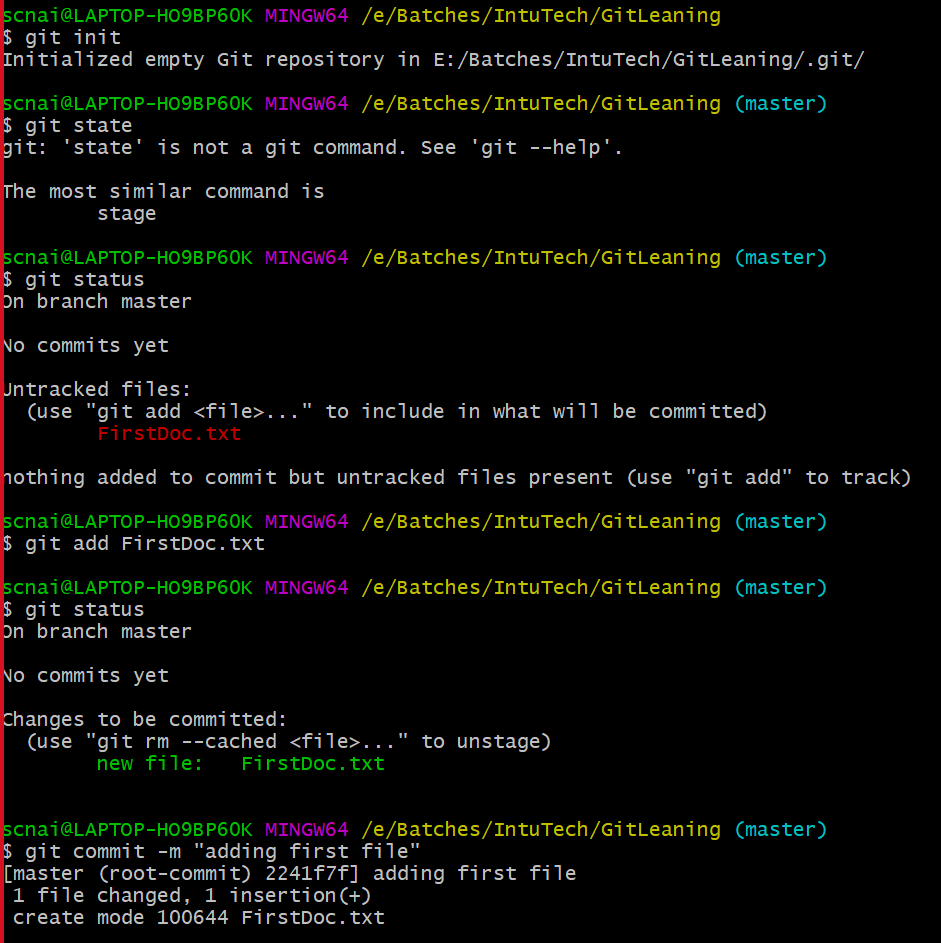
**git commit -m “Message”**

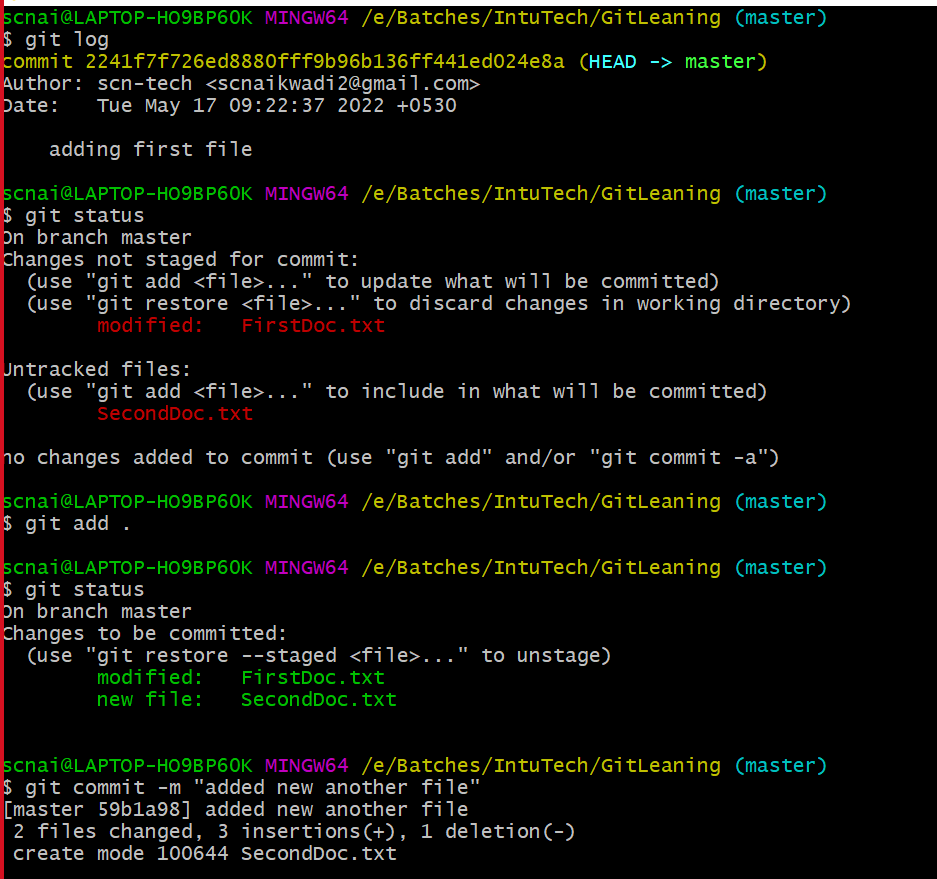
Using this command you can start maintain the file versions and metadata of the added files.

One commit can have more than one files. Every commit will get a unique Commit ID which is also known as version.

1. To view all the commits

**git log**





**Git Configuration:**

**Git config --list** : to list out the all configurations

**git config --global user.name "<User\_Name>"**

Command is use to set the user name at global level

**git config --global user.email "<Email>"**

Command is use to set the email at global level

**Git checkout**

Is use to switch between one commit to another or one branch to another.

git checkout <commit-id>

git checkout <branch\_name>

**git branch**

it is use to sperate out the code changes from the original code. After creating branch you will get the copy of the original code into a branch. You can independently work on branch and commit the changes without affecting the original code

**git branch**: to get the list of all the branches

**git branch <branch\_Name>**: to create a new branch

**git checkout <branch\_name> :** to switch between branch

**Git Merge**

To merge the changes of one branch to another

Steps:

1. Witch to a branch where you wants to merge the changes using git checkout command.
2. Then use a merge command followed with the branch name

**git merge <branch\_name>**

**GitHub**

1. It is a cloud repository.
2. It is a central location where you can save you files/code.
3. This files/code can be access by anyone from the network if it is public.
4. You can create a private repository also where only the owner or the collaborators can see the files/code (can add up to 3 for free account).

Git clone:

To get clone the remote repository in to a local system. This is the first time activity.

**git clone <Repo\_URL>**

git push

To Push the committed file from the local repository to remote/cloud repository

**git push**

git pull

to get the cloud changes into the local repository

**git pull**