Lessons Learnt 🡪 LL

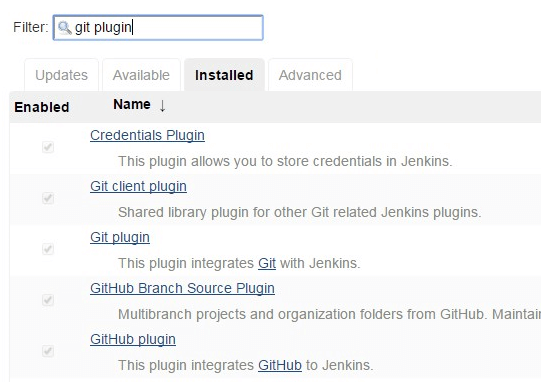
## **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

## **Install Jenkins Git Plugin**

In order to use the GIT in Jenkins, you should have “Git Plugin” installed in your Jenkins.

Login to Jenkins UI using your admin account, and go to “Manage Jenkins” -> Manage Plugins -> Click on “Installed” Tab -> From here, search for “git plugin” in the filter.

If the “Git plugin” is already installed, it will display it here as shown below.



During Jenkins installation, if you’ve chosen to install the “recommended plugin”, you’ll already have “git plugin” as shown above.

If you don’t see it, then install it. Go to “Available” tab, search for “git plugin”, and click on “Install without restart”.

From your plugins list, if you see that “Git Plugin” is an older version, then upgrade it to the latest version as explained here: [How to Upgrade Jenkins to New Version and Update Plugins](https://www.thegeekstuff.com/2016/06/upgrade-jenkins-and-plugins/)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**LL 1)** **If I right click on MavenProject or pom.xml 🡪 Run As 🡪 Maven Install**

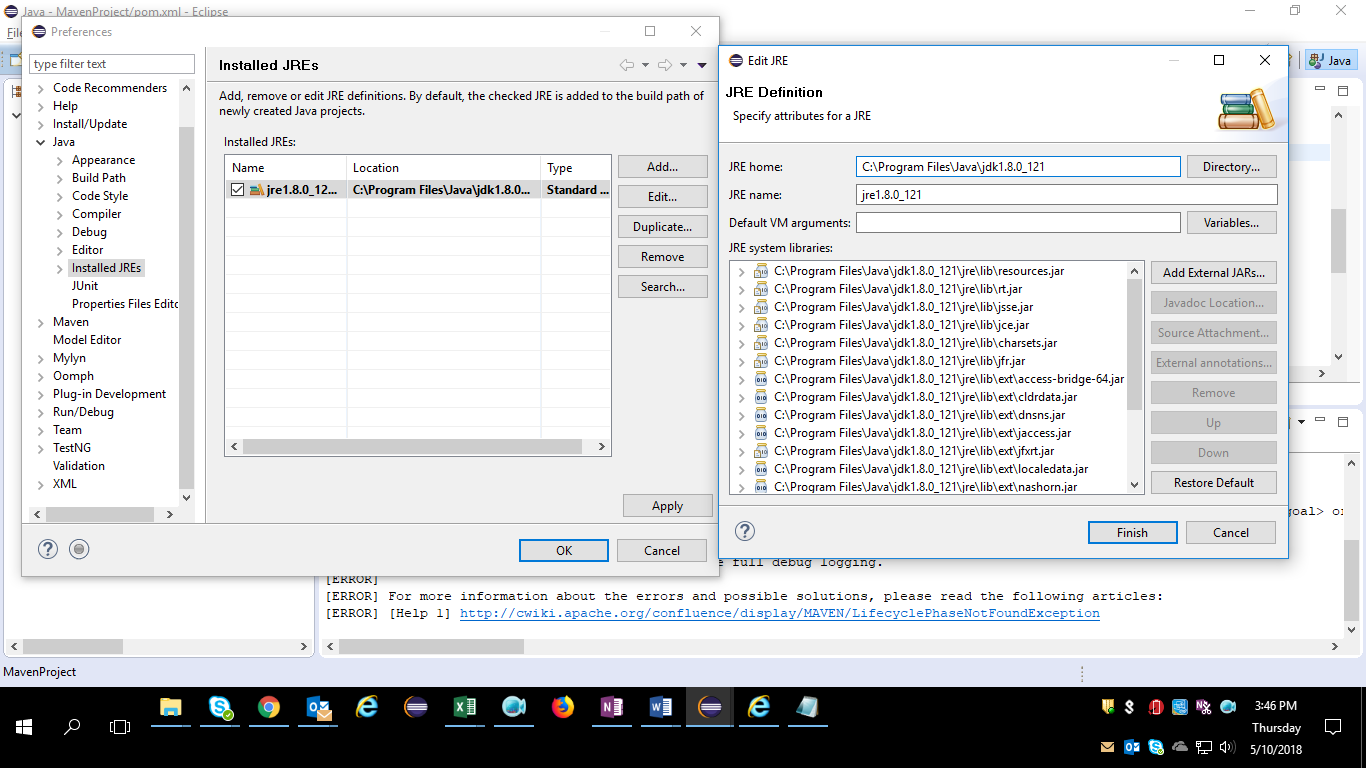
I got an **Error :**

No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?

**Solution** for this error :

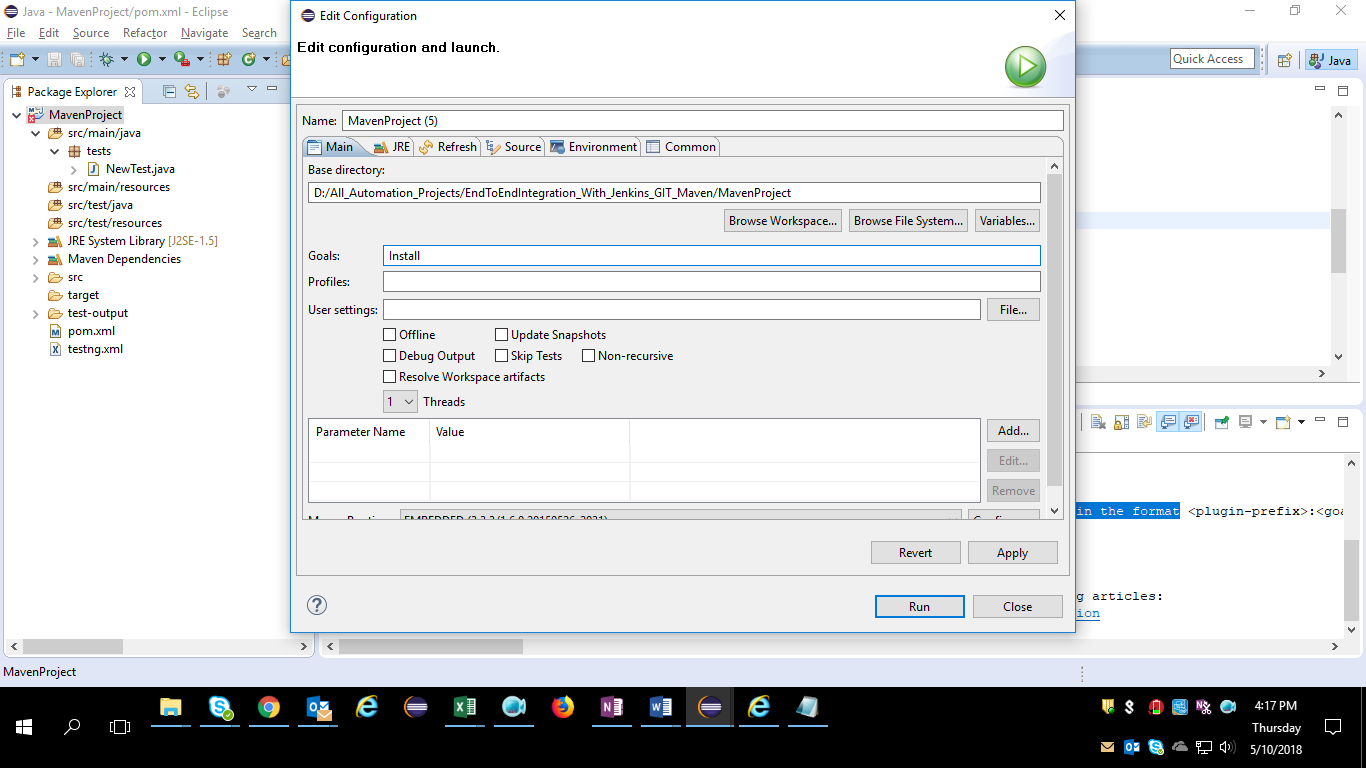
In Eclipse Go to Window --> Preferences --> Java --> Installed JREs.

And see if there is an entry pointing to your JDK path, and if not, click on Edit button and put the path you configured your JAVA\_HOME environment. (i.e., the path where in the JDK is present .Ex : C:\Program Files\Java\jdk1.8.0\_121 )



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 2)** **When I clicked on Run As 🡪 Maven Build I got below window :**



Then I got the BUILD FAILURE with error :

Unknown lifecycle phase "Install". You must specify a valid lifecycle phase or a goal in the format

**Solution:** This is because in maven we have specific build phases with predefined keywords, So under Goals field in eclipse we have to mention the proper keyword or else the BUILD will Fail.

In the above picture we can see, I entered as “Install” in Goals field , but I should enter it as “install” then it will Build

i.e., Keywords are Case sensitive

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 3) Sometimes when I try to do Run As 🡪 Maven Build (Goals = test), I use to get error like :**

[Maven Compilation error [package org.testng.annotations does not exist]](https://stackoverflow.com/questions/20345451/maven-compilation-error-package-org-testng-annotations-does-not-exist)

**Solution for this :** we need to remove the <scope>test</scope> tag from

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.1.1</version>

<scope>test</scope> 🡪 Remove this Tag.

</dependency>

Then BUILD will run successfully.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 4) How to set the Chrome Driver Executable file path whenever I am using Maven Dependency.**

**Solution :** Usually I have created the package named ‘driver’ inside the ‘src/java/resources/’ and kept chromedriver.exe inside that package and I used the path of that package in my code System.setProperty(“webdriver.chrom.driver”, “Path of Chrome driver inside src/java/resources/”);

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 5) How to tell the Maven to download the latest version of particular jar.**

**Solution :**

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.14</version>

It will download the log4j version 1.2.14 library automatically. If the “version” tag is ignozed, it will upgrade the library automatically when there is a newer version.

When Maven is compiling or building, the log4j jar will be downloaded automatically and put it into your Maven local repository.

See the difference? So what just happened in Maven? When you build a Maven’s project, the pom.xml file will be parsed, if it see the log4j Maven coordinate, then Maven search the log4j library in this order:

1. Search log4j in Maven local repository.
2. Search log4j in Maven central repository.
3. Search log4j in Maven remote repository (if defined in pom.xml).

This Maven dependency library management is a very nice tool, and save you a lot of work.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

# LL 6) Where is Maven local repository?

The maven local repository is a local folder that is used to store all your project’s dependencies (plugin jars and other files which are downloaded by Maven). In simple, when you build a Maven project, all dependency files will be stored in your Maven local repository.

By default, Maven local repository is default to .m2 folder :

Windows – Syntax 🡪 C:\Documents and Settings\{your-username}\.m2

My System Path 🡪 **C:\Users\likhid\.m2**

## **1. Update Maven Local Repository**

Normally, I will change the default local repository folder from default .m2 to another more meaningful name, for example, maven-repo.

Find **{M2\_HOME}\conf\setting.xml**, update localRepository to something else.

{M2\_HOME}\conf\setting.xml

In My Machine path is 🡪 D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Softwares\apache-maven-3.2.2\conf

<settings>

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

|

| Default: ~/.m2/repository

<localRepository>/path/to/local/repo</localRepository>

-->

<localRepository>D:/maven\_repo</localRepository>

## **2. Saved it**

Done, your new Maven local repository is now changed to D:/maven\_repo.

**Note :** Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

## **LL 7) Central Repository**

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL − <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows −

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

To browse the content of central maven repository, maven community has provided a URL − <https://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

## **LL 8) Remote Repository**

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository**, which is developer's own custom repository containing required libraries or other project jars.

For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.

<project xmlns = "http://maven.apache.org/POM/4.0.0"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>com.companyname.common-lib</groupId>

<artifactId>common-lib</artifactId>

<version>1.0.0</version>

</dependency>

<dependencies>

<repositories>

<repository>

<id>companyname.lib1</id>

<url>http://download.companyname.org/maven2/lib1</url>

</repository>

<repository>

<id>companyname.lib2</id>

<url>http://download.companyname.org/maven2/lib2</url>

</repository>

</repositories>

</project>

**LL 9) When I try to run the Maven Project from Jenkins I got below error :**

'mvn' is not recognized as an internal or external command,

Operable program or batch file.

**Solution : first we need to install the Maven from** [**http://redrockdigimark.com/apachemirror/maven/binaries/**](http://redrockdigimark.com/apachemirror/maven/binaries/)

**Then we need to set the environment Variables M2\_HOME=**D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Softwares\apache-maven-3.2.2

And **Path =** D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Softwares\apache-maven-3.2.2\bin

Also I observed that if we set **JAVA\_HOME** = C:\Program Files\Java\jdk1.8.0\_121\bin , then we will get an error like JAVA\_HOME is not pointing to a directory where it is present (something like this) . So we always have to set the JAVA\_HOME and M2\_HOME (maven path) to C:\Program Files\Java\jdk1.8.0\_121 and D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Softwares\apache-maven-3.2.2 respectively , not till bin folder (ex : C:\Program Files\Java\jdk1.8.0\_121\bin)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 10) If I want to know the Version of the plugin used in Maven then I use below command in Command Promt**

**mvn help:describe -DgroupId=org.apache.maven.plugins -DartifactId=maven-surefire-plugin**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 11) If I want to Know all the Dependency Jars Versions I use below Command in Command prompt**

**mvn versions:display-dependency-updates**

**Note : First I have to switch to the path where the pom.xml is present and then I have to use this command**

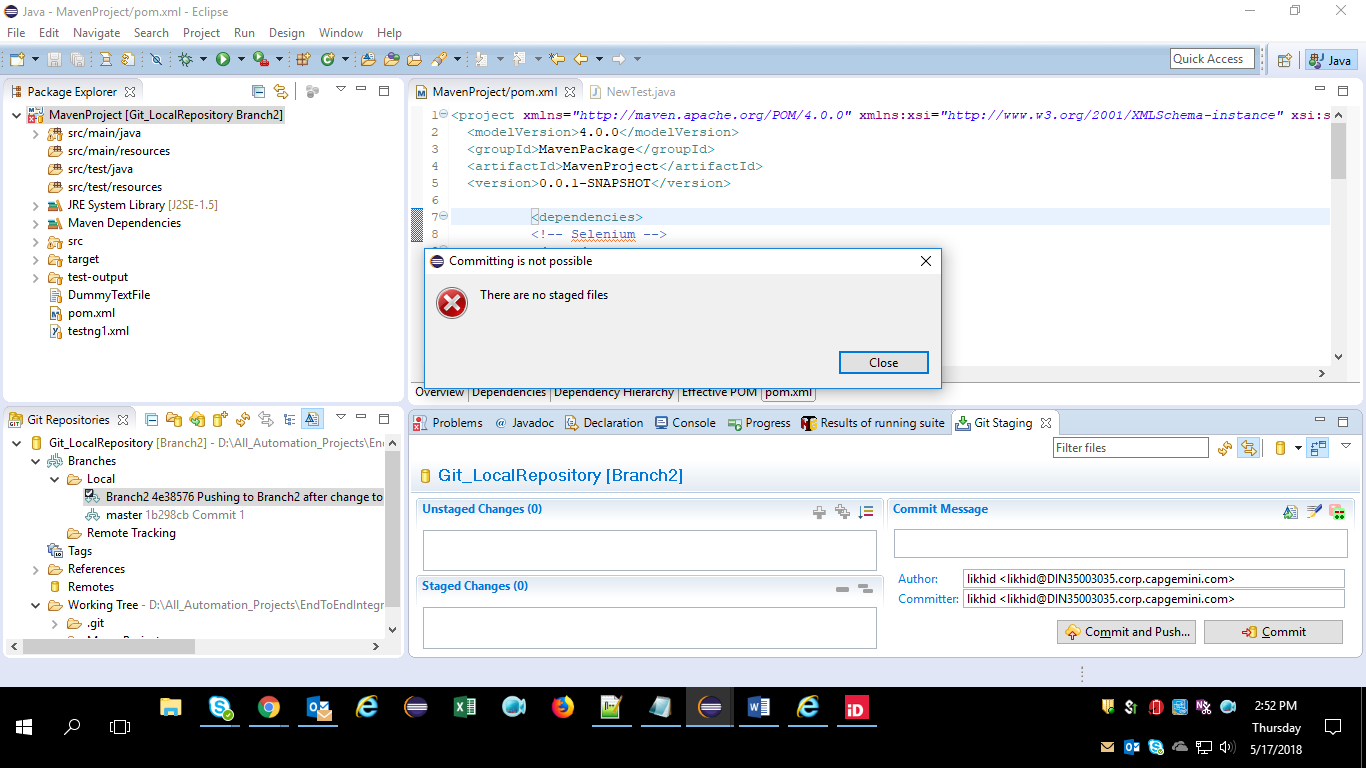
**Ex : >** 1) Open Command prompt

2) cd /d D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Git\_LocalRepository\MavenProject

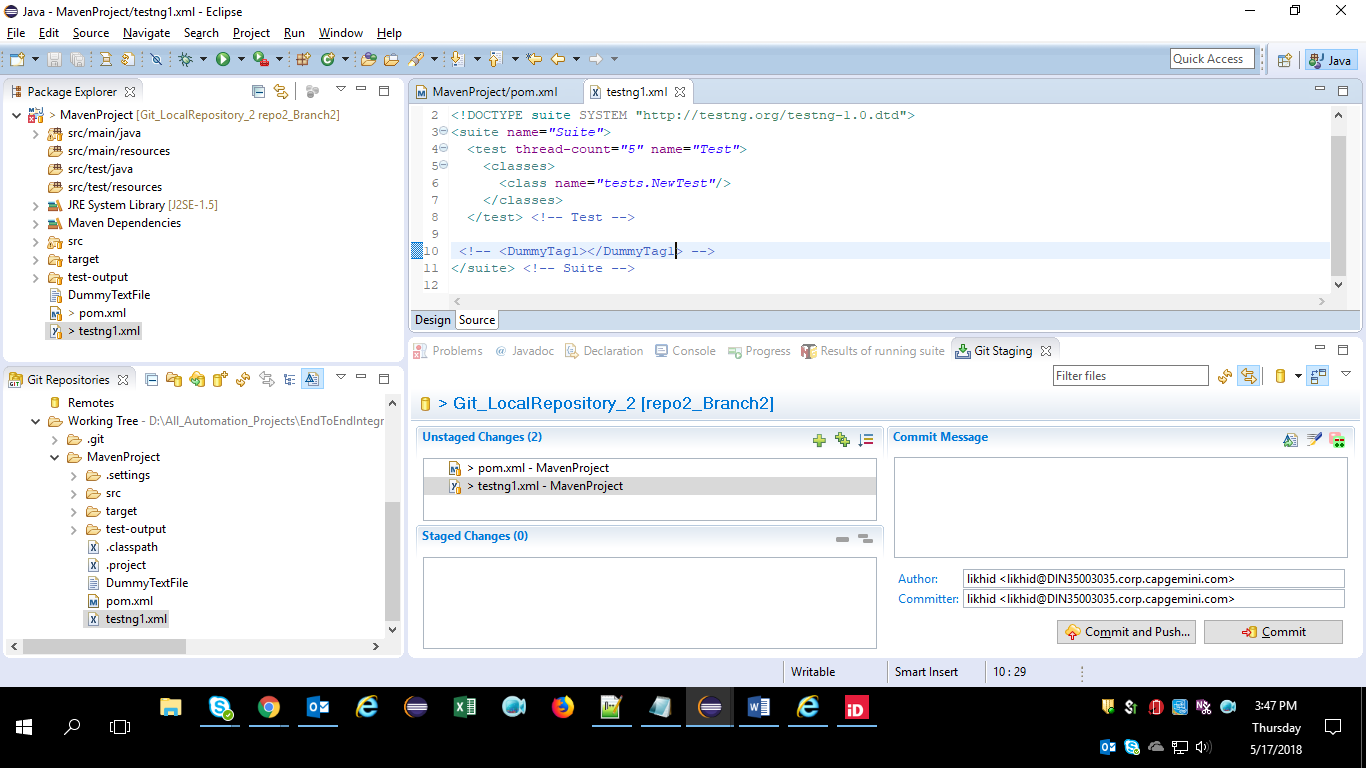
3) mvn versions:display-dependency-updates

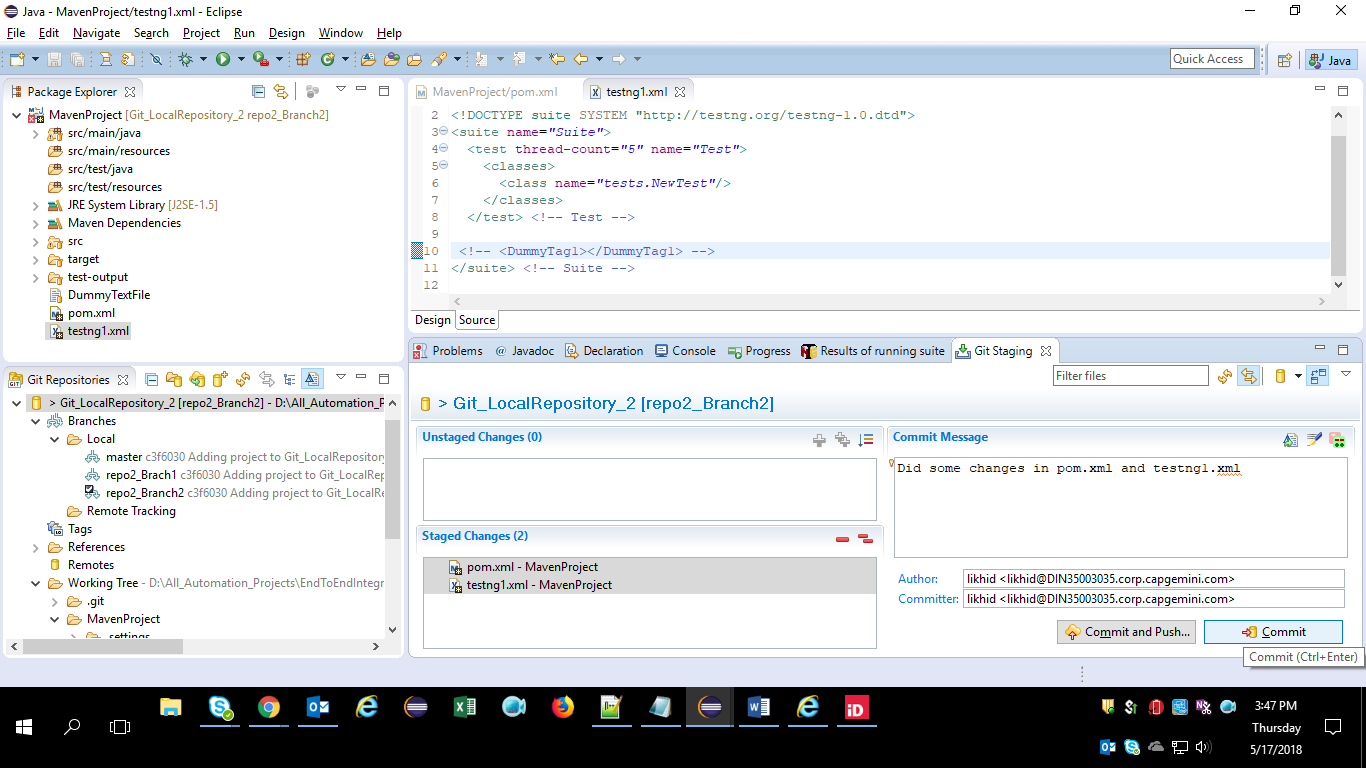
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 12) If we don’t do any changes in the code in eclipse and try to push the project from Eclipse to Git Repository we will get an error like below:**



**So If we change something in some file(ex: pom.xml and testing.xml) then that time we will get those file names under Unstages Changes section and we need to push them to Stages Changes section and we need to write the commit message , then we are good to Commit the changes to Git Repository.**



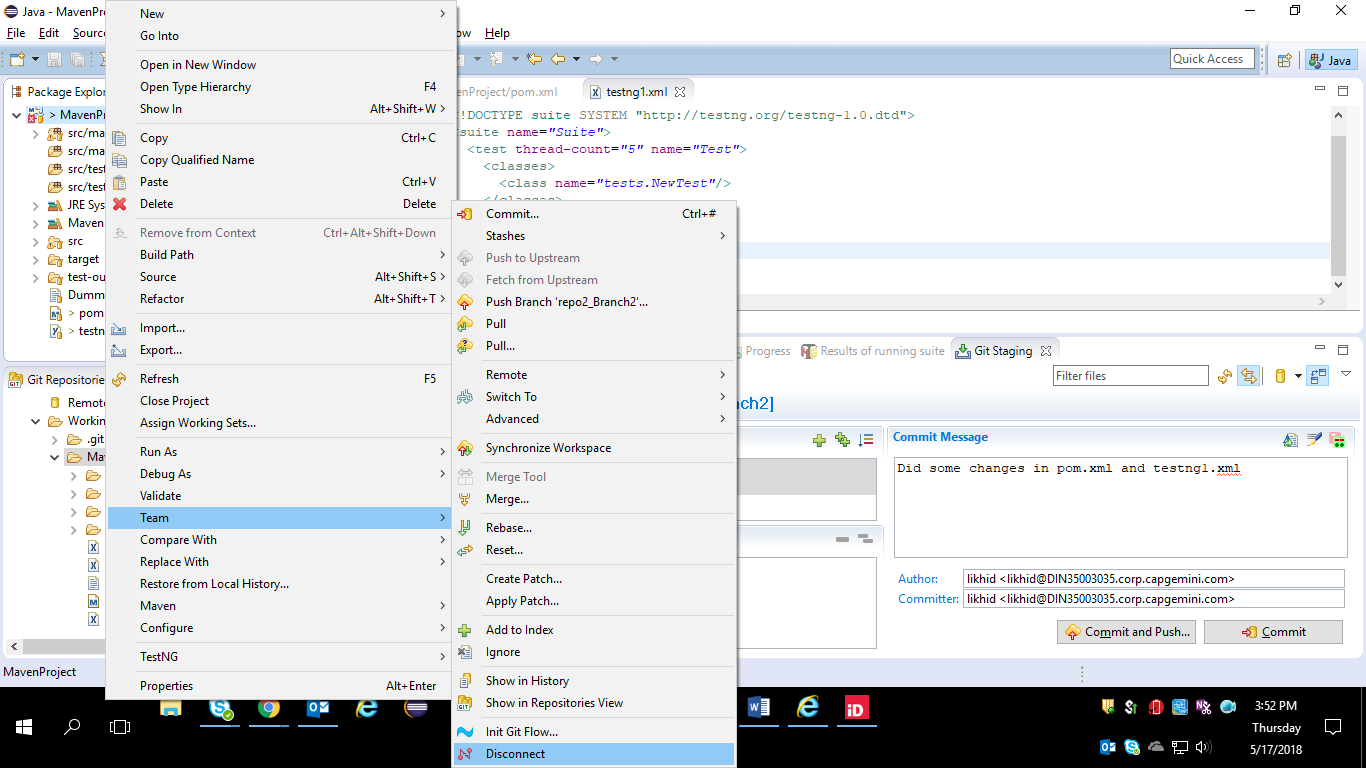


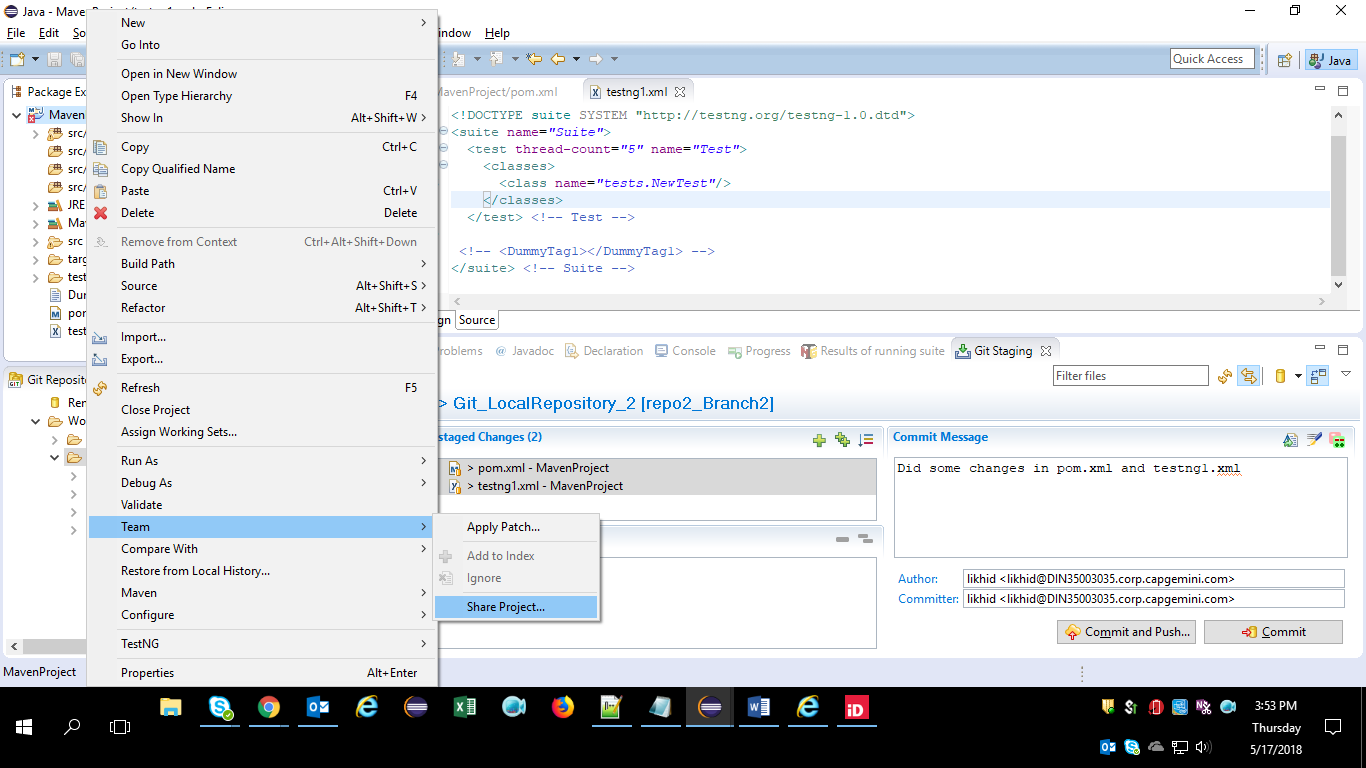
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

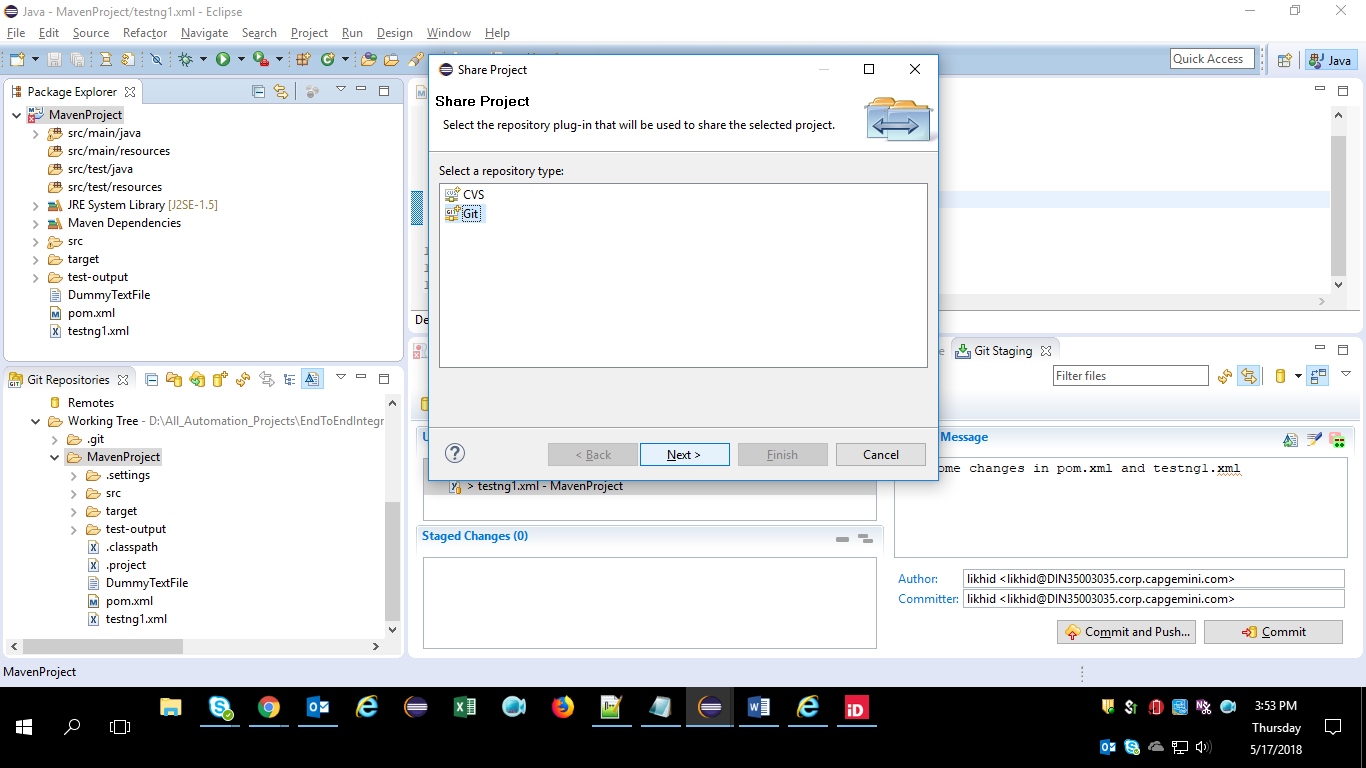
**LL 13) If we have 2 local repositories (say repo1 and repo2) and we want to commit our changes to second repository(repo2) but still still we are connected to first repository(repo1) , we have to follow the below steps:**

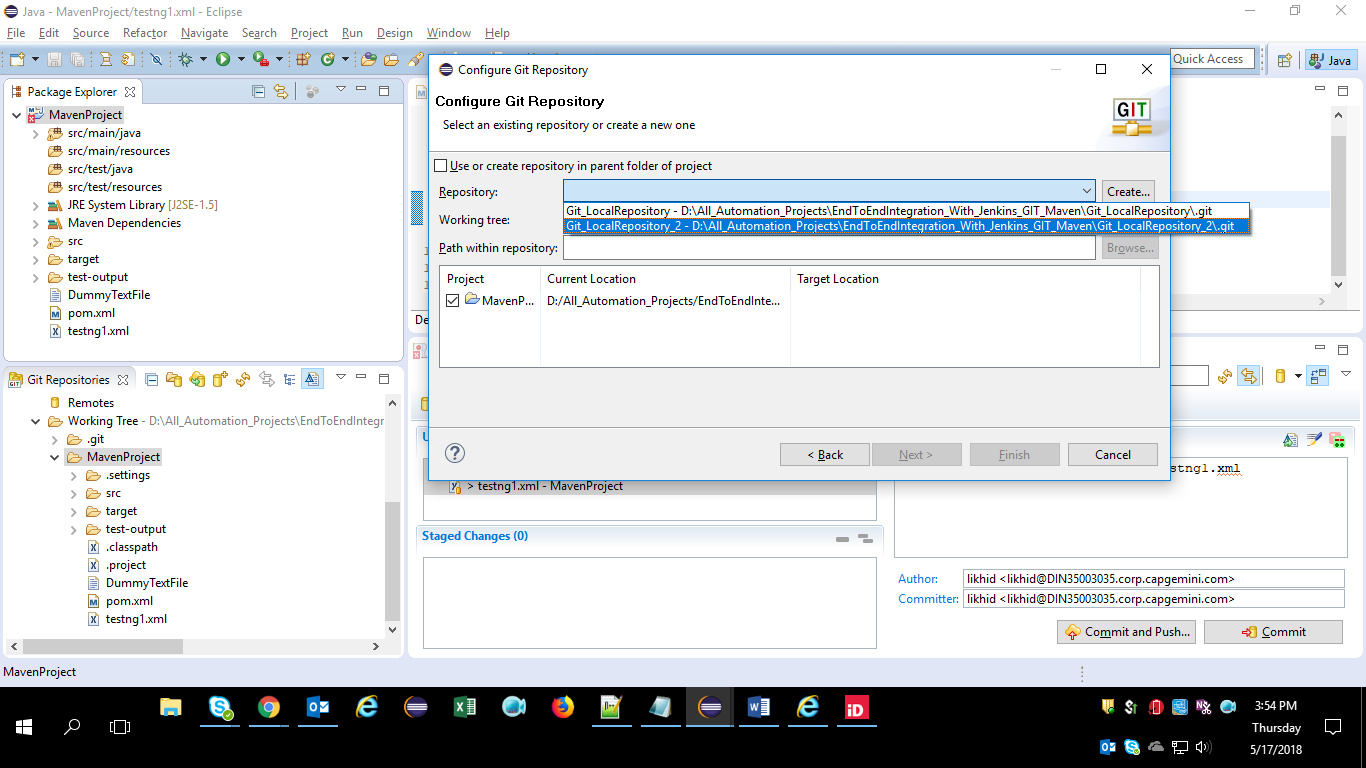
Right-click MavenProject project, Team->Disconnect

Right-click MavenProject project, Team->Share Project...









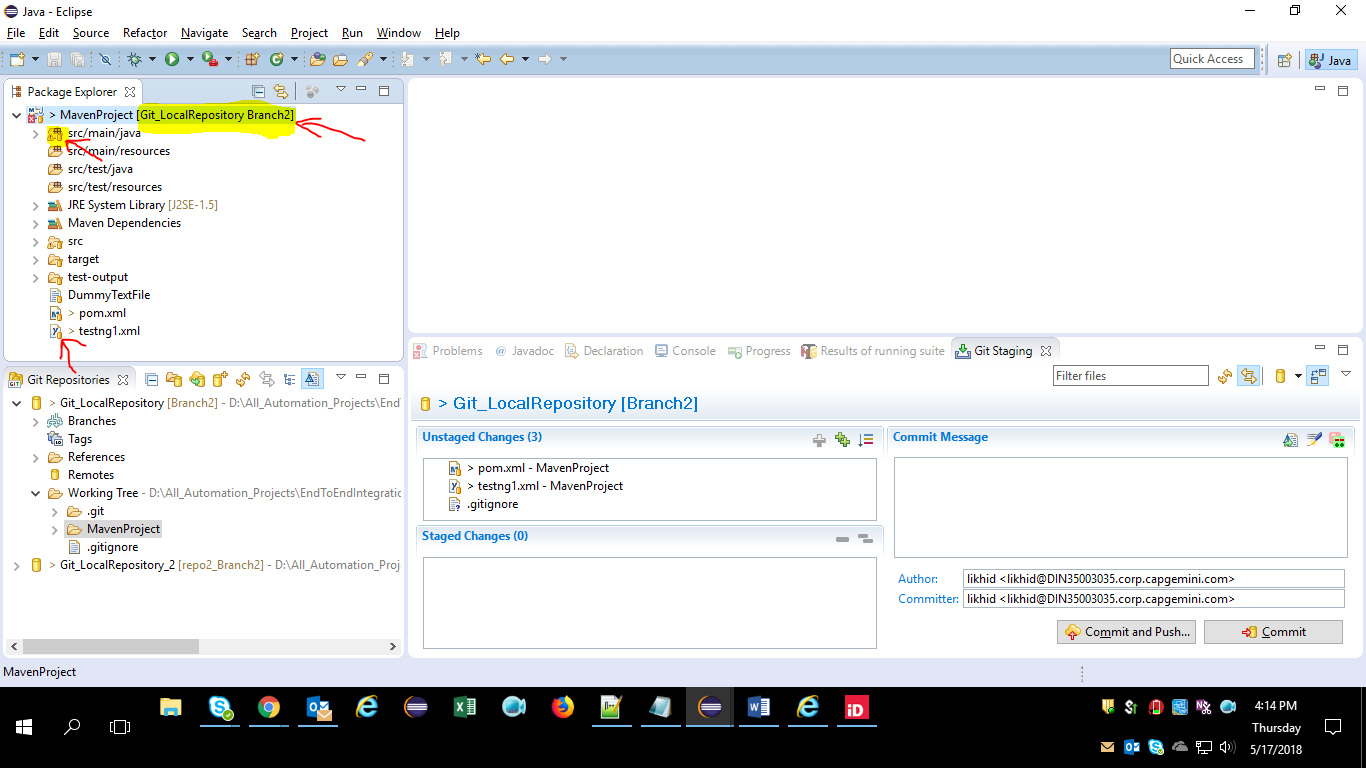
The wizard automatically finds the Git report relative to the project dir: ../.git

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

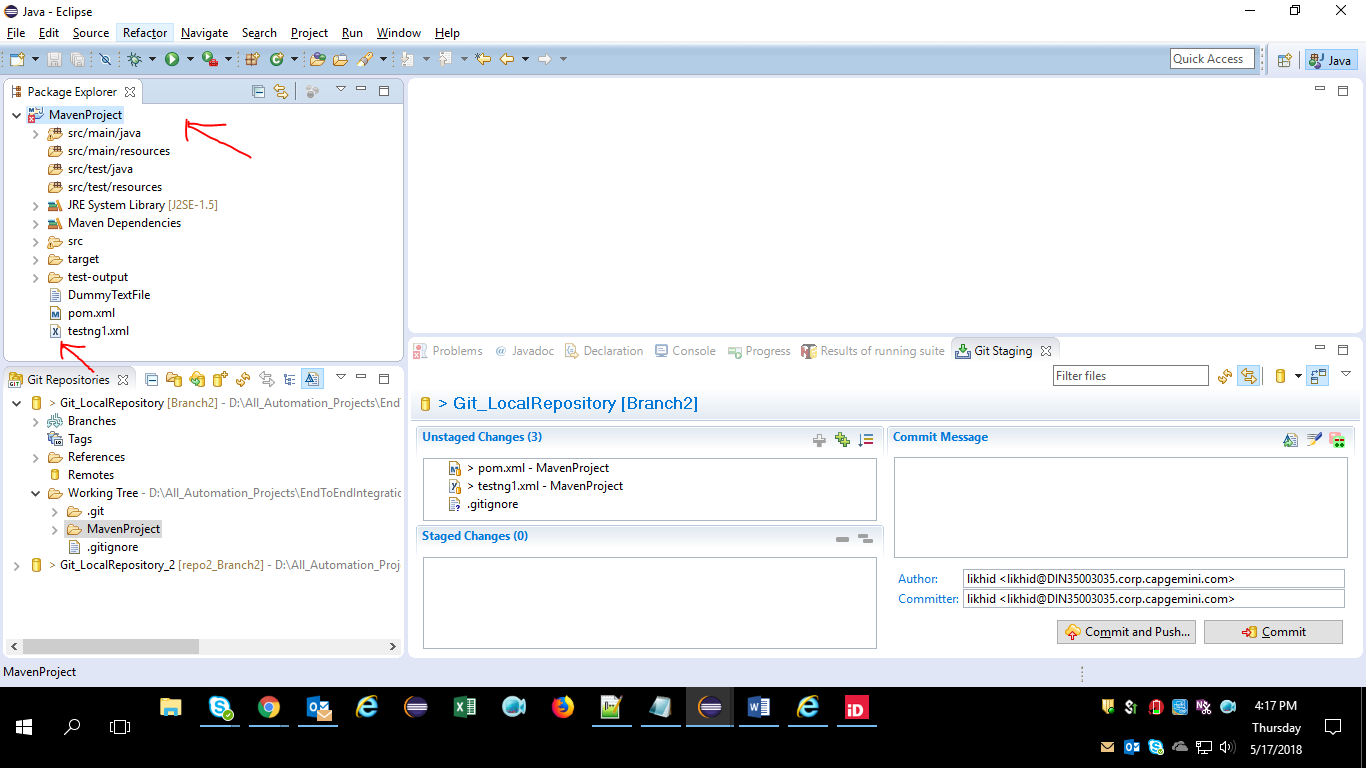
**LL 14)** As far as I know , we cannot share the maven project code from eclipse to GitHub Repository (remote repo) directly , first we have to create a Local Git Repository and then we have to Commit the changes to that local repository and only then we are able to Push our Code to Remote Github repository using **Right Click on Project -> Team -> Remote -> Push**

**Note :** we should be connected to Local Git repository inorder to move our code to Remote Github repository

**Below figure shows Maven project is connected to Local Repository**



**Below figure shows Maven project is NOT connected to Local Repository**



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

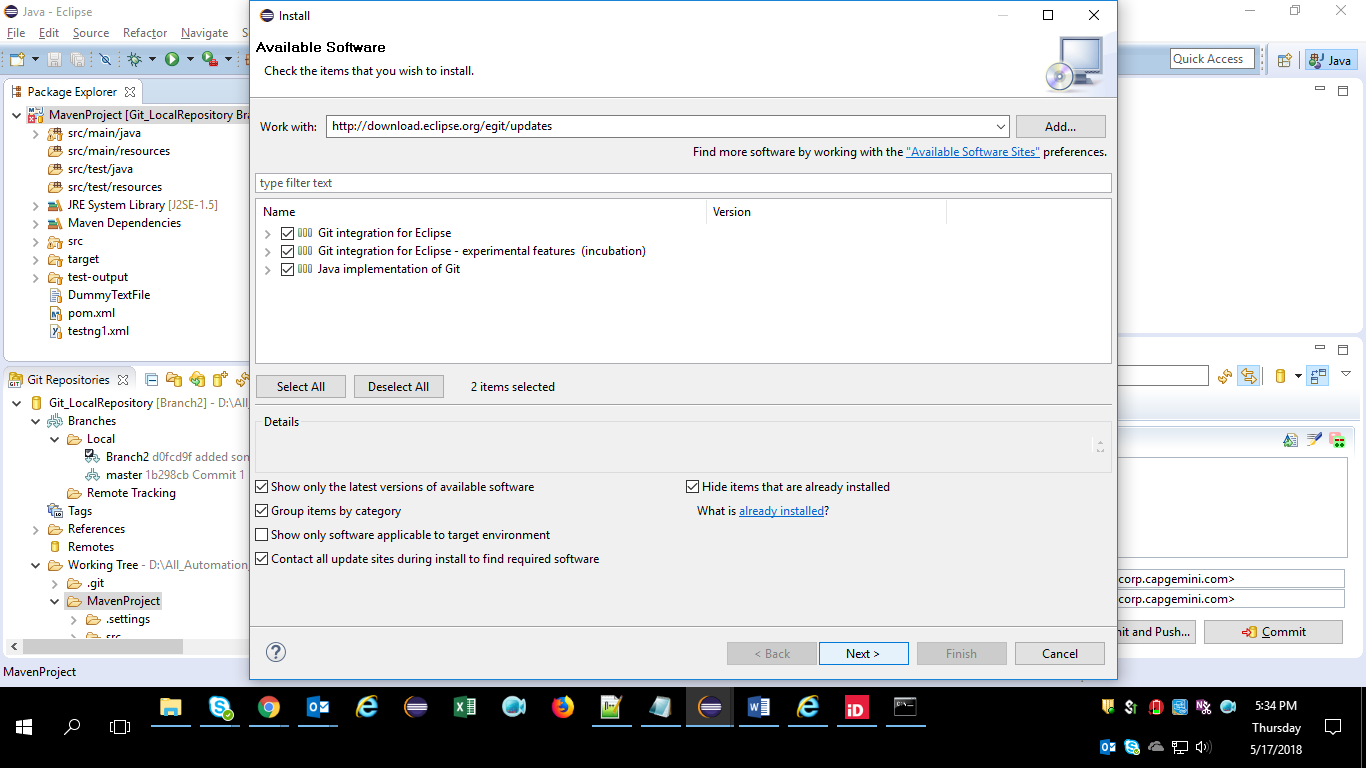
## **LL 15)** Installing EGit in Eclipse.

You will need to install the git plugin for eclipse, EGit, in order to upload code from eclipse projects.

* In eclipse, go to Help / Install New Software
* Enter below URL Inside “work with” text field.

**http://download.eclipse.org/egit/updates**

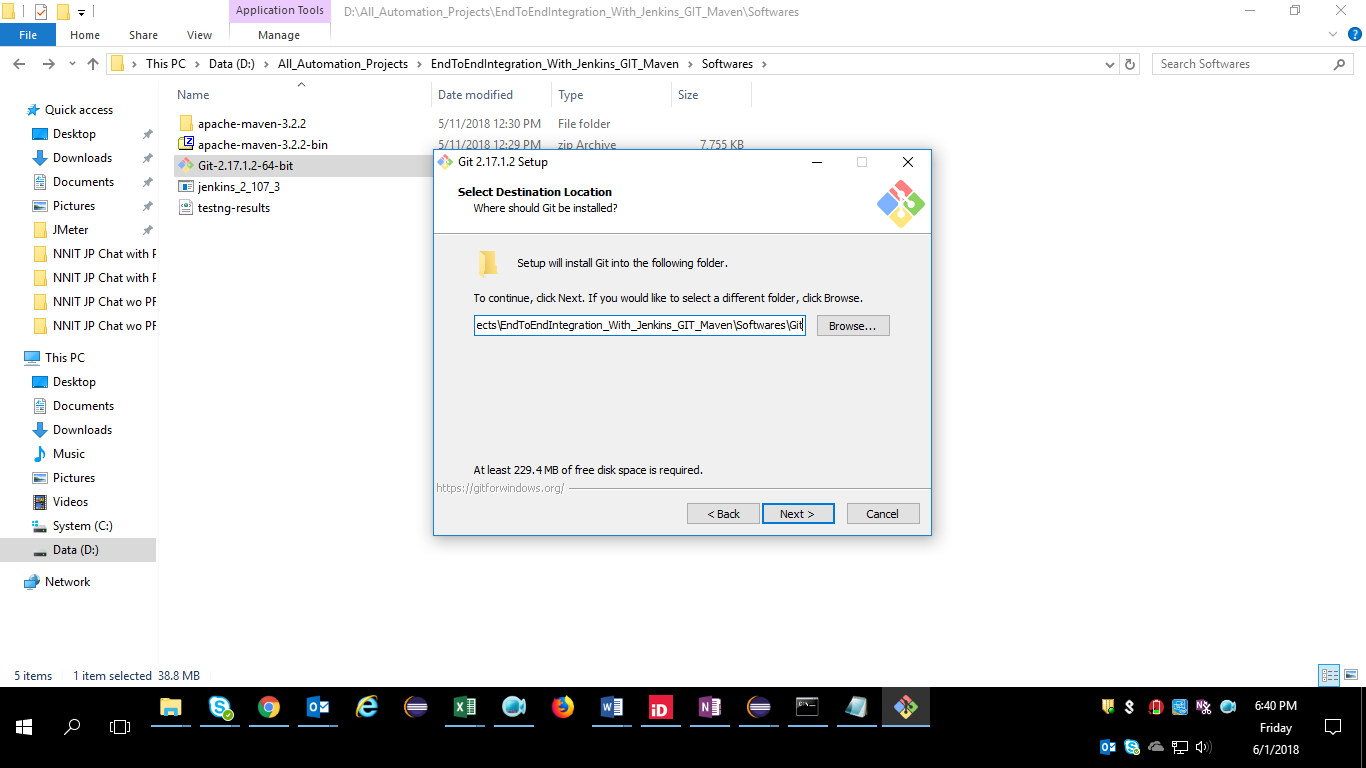
* List of plugins(Usually 3 plugins) will display
* Click on Select All button
* Click Next



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**LL 16)** In order to Download (we call it technically as “Clone”) and run the latest code from GitHub repository branches (Master or any branch) via Jenkins , we need to download and install Git in our local system first. For that follow the steps below:

1. Go to [**https://git-scm.com/download/win**](https://git-scm.com/download/win)
2. Download .exe file into your local. **Ex : Git-2.17.1.2-64-bit**
3. Install the Git by double clicking on that exe file.
4. And while installing mention the path where you want it to create Git folder(Shown in below figure)



1. It will create Git folder in the path specified

**Ex : D:\All\_Automation\_Projects\EndToEndIntegration\_With\_Jenkins\_GIT\_Maven\Softwares**

1. After you installed the GIT in your system , we need to set the git.exe file path in the Jenkins , for that follow the below steps
   * + - Go to Manage Jenkins
       - Go to Global Tool Configuration
       - In Git / Path to Git executable enter C:\<whatever the path is>\git.exe.
       - Click on Save.
2. Also you need to set the Environment Variables PATH = C:\<whatever the path is>\git.exe.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LL 17) How to add Git Credentials in Jenkins

When you create the repository, you might get the following error:

Repository URL error 1: ramesh@192.168.101.2:/home/git/devapp

Failed to connect to repository : Command "git ls-remote -h ramesh@192.168.101.2:/home/git/devapp HEAD" returned status code 128:

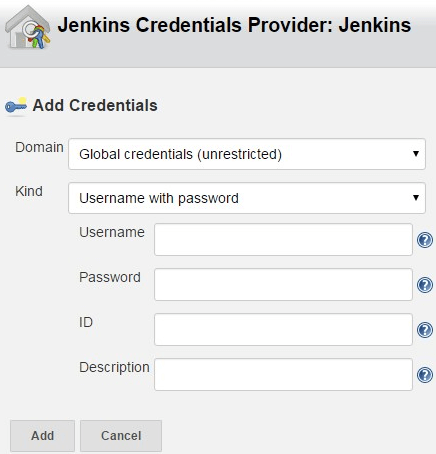
stdout:

stderr: Host key verification failed.

fatal: The remote end hung up unexpectedly

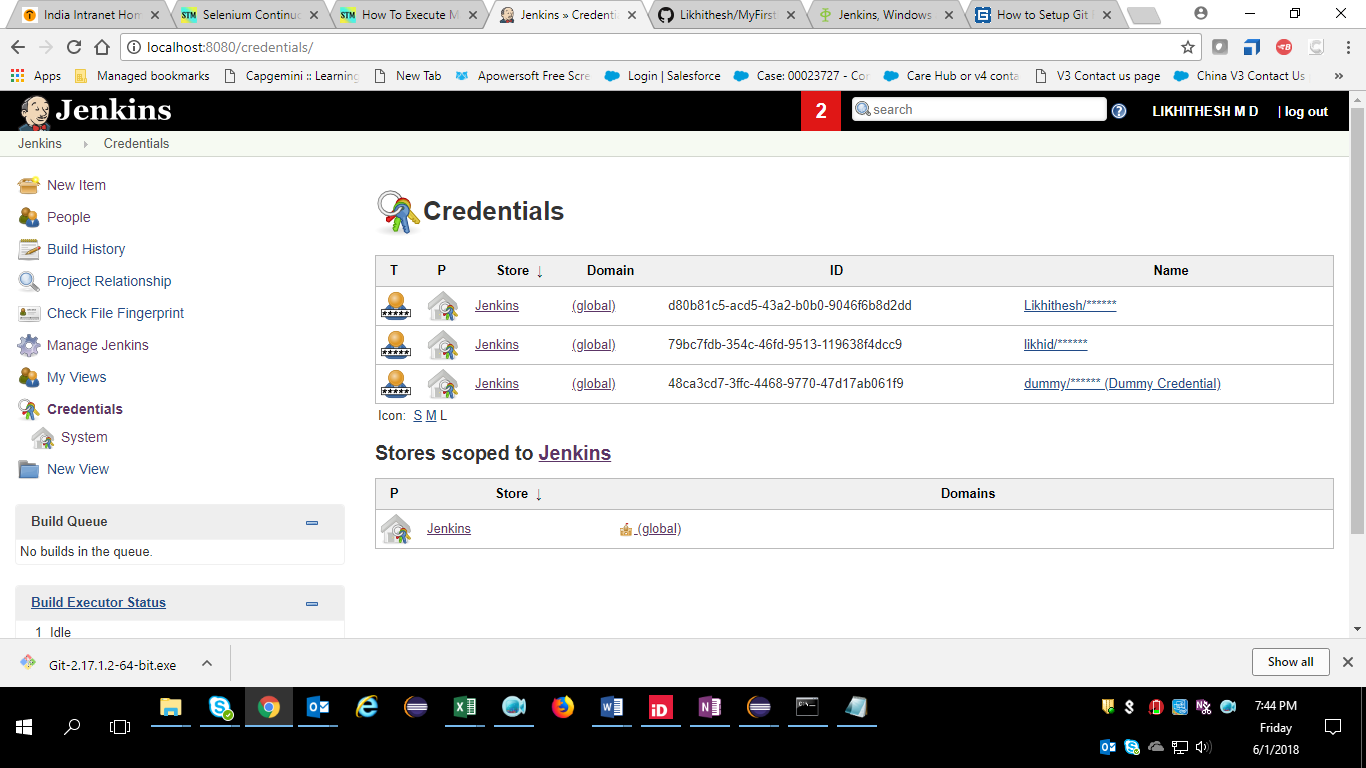
Typically, you’ll get the above error message, when your “Credentials” under “Git” is set to “None”.

To add a credential, click on “Add” next to “Credentials” -> Select “Jenkins Credential Provider”, this will display the following add credentials screen.



* Domain: By default “Global credentials (unrestricted)” is selected. Other option is: “Username and password”. Use default.
* Kind: The following are the possible options:
  + Username and password (This is shown in the above screenshot).
  + SSH Username with private key
  + Secret file
  + Secret text
  + Certificate
* Username – Specify the username that will be used to connect to the remote git repository
* Password – Specify the password for the above username.
* ID – Leave this empty. This is an internal unique ID that are used to identify this credentials in the job. When you leave this empty, Jenkins will auto generate an id.
* Description – Descriptive notes about this credentials

After this, if you want to view all the credentials that you’ve created so far in Jenkins, go to “Jenkins” -> Click on “Credentials” on the menu, which will list all the credentials that you’ve created inside Jenkins so far as shown below.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LL 18) Detail Steps to delete GitHub's credentials from Jenkins:

1. Go to **Jenkins's Dashboard**
2. Click on "**Credentials**" [Located at left side menu]
3. You will now able to see :
   * **Store**
   * **Domain**
   * **ID**
   * **Name**
4. Click on on "**Name**", you will get options "**Update**", "**Delete**" & "**Move**". **Choose your option.!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LL 19) In Jenkins when I am trying to download and run the latest code from GITHUB ,

I am getting an error like below :

**[ERROR] The project (C:\Users\likhid\.jenkins\workspace\MavenProject\pom.xml) has 1 error**

**[ERROR] Non-readable POM C:\Users\likhid\.jenkins\workspace\MavenProject\pom.xml: C:\Users\likhid\.jenkins\workspace\MavenProject\pom.xml (The system cannot find the file specified)**

**Solution :** This is because

Every Time when you start the execution (new Build) in Jenkins , it will fetch whatever the code present in Github branch (Master or any other) you specified in Jenkins.

When the Jenkins fetch the latest code from GitHub , it will store that in your local

**C:\Users\likhid\.jenkins\workspace\{Jenkins Project Name}**

If your pom.xml file exists in the root folder of workspace , then there is no need for you to mention the path of pom.xml file. Just write it as pom.xml

If your pom.xml file exists inside the root folder , then the easy way to access that would be

$workspace\subfolder\pom.xml

So In my case

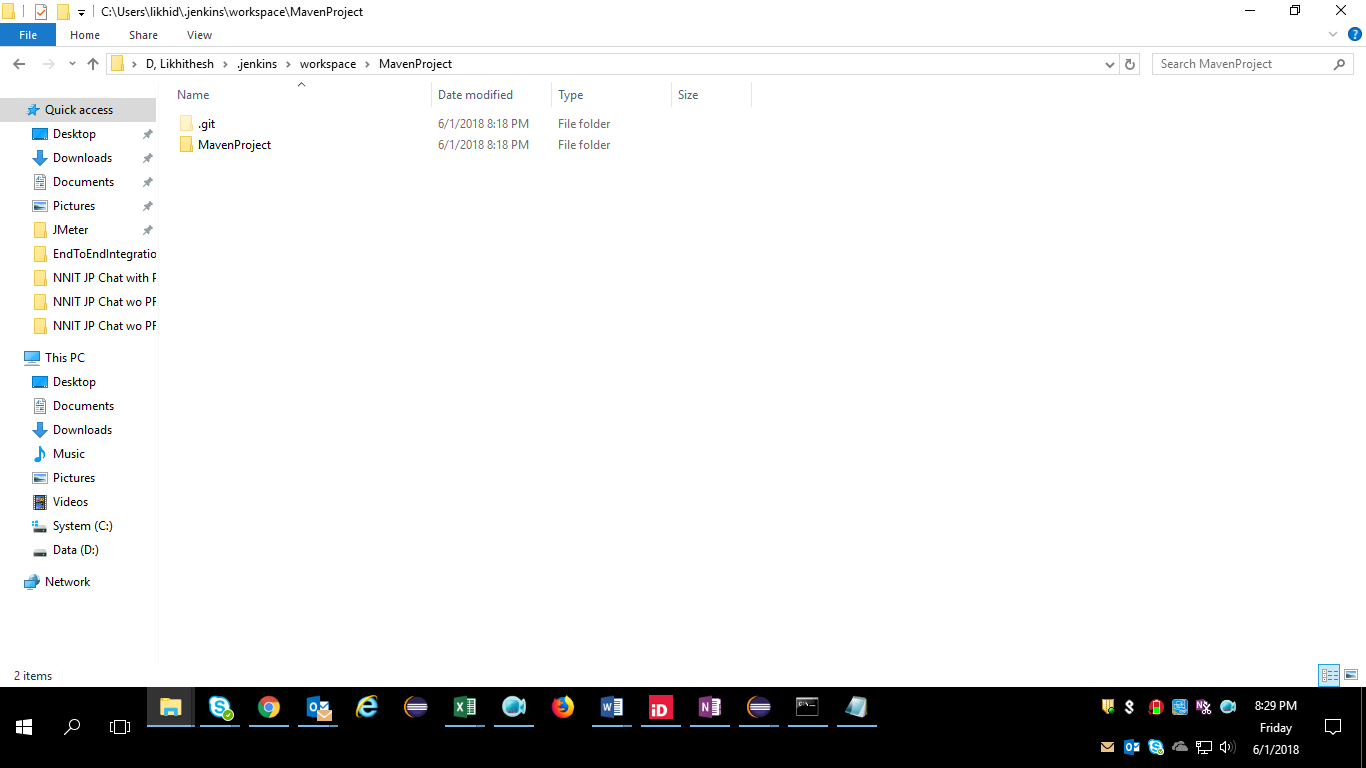
1. root folder of workspace is = **C:\Users\likhid\.jenkins\workspace\MavenProject**

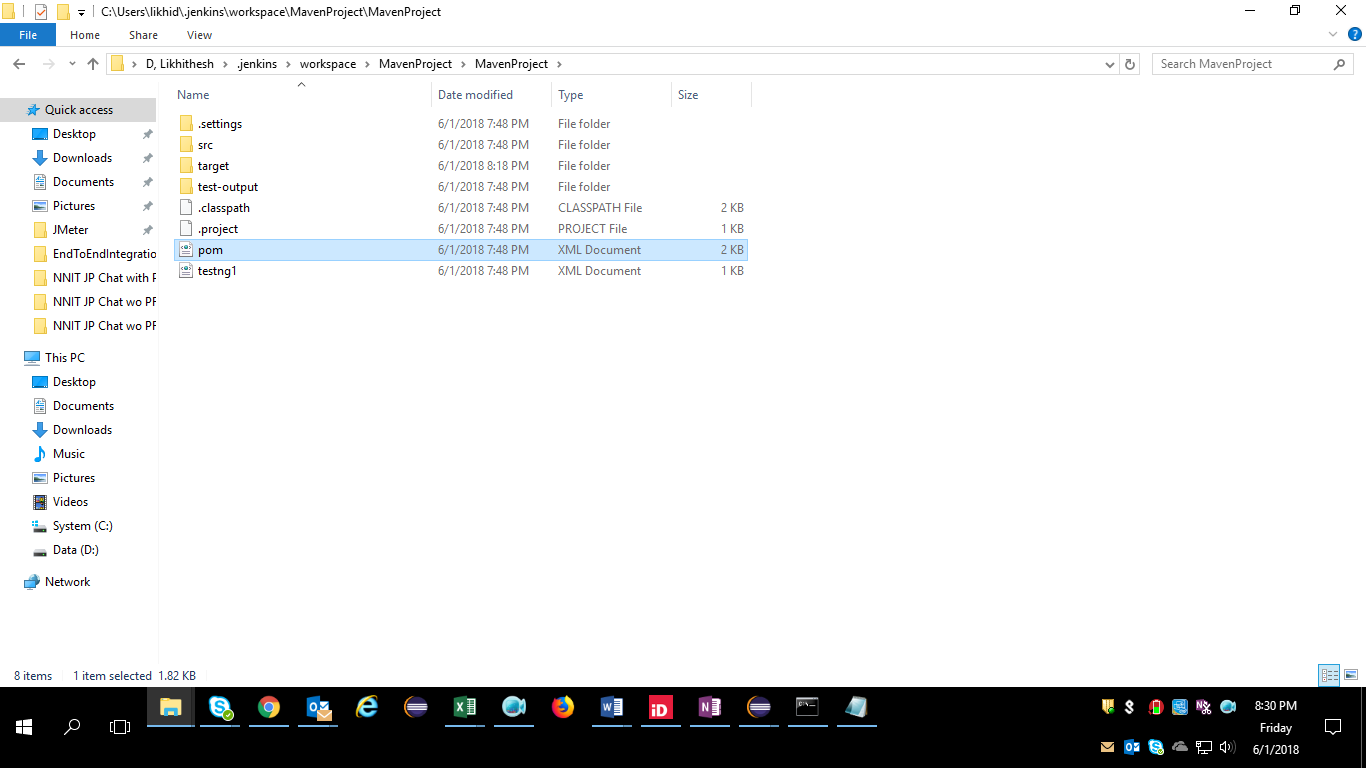
and the pom.xml is not present here , but it is present inside one more subfolder named **MavenProject** as shown in the below diagram

So in Jenkins I wrote it like :

$workspace\MavenProject\pom.xml

Then My Code Executed Successfully ☺





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**LL 20)** Always when you did new changes to your code and committing your changes to GitHub repository, in the comments field always try to include current date along with your proper comment, so that you can track your changes in GitHub easily like, when you did the changes exactly ☺ and to what file you did the changes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

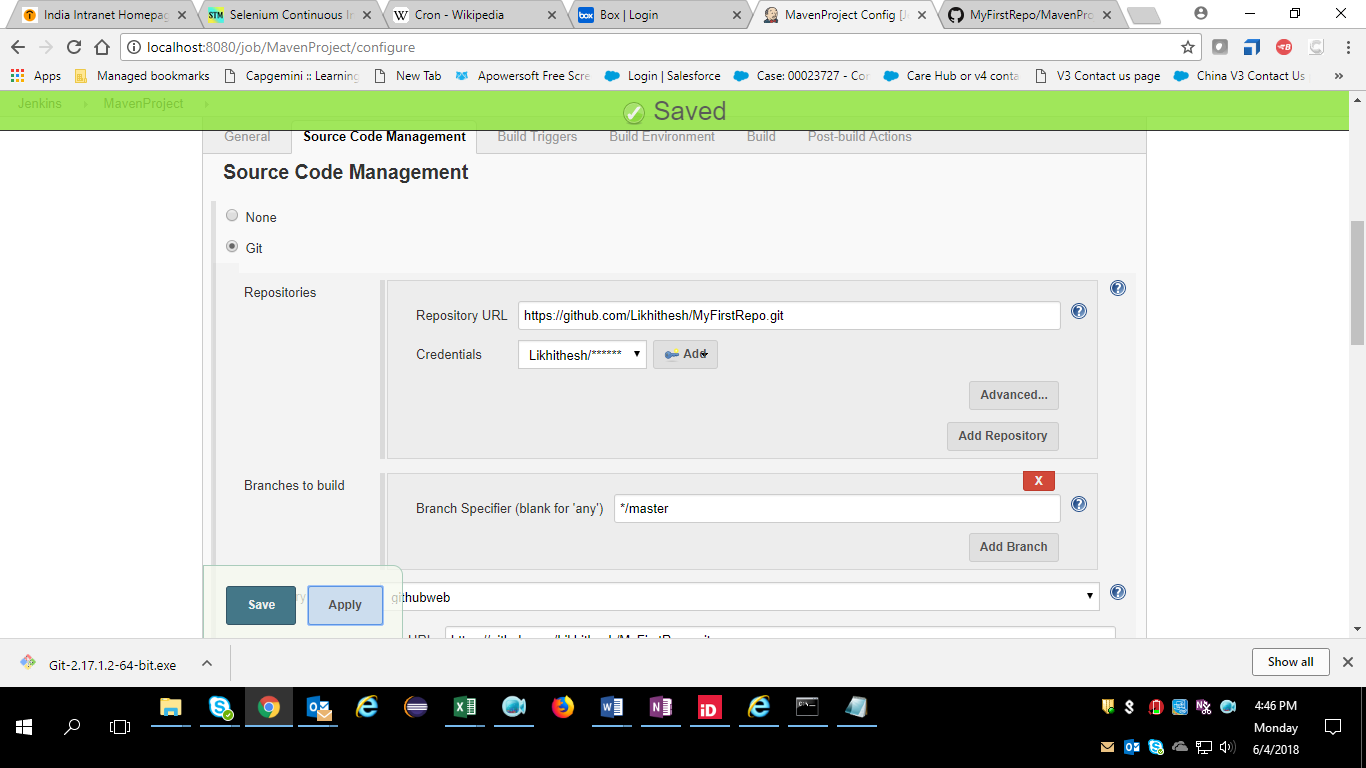
**LL 21) If you want to see the build status in Green Color instead of Blue , install “Green Ball” Plugin in Jenkins**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LL 22) This is how I need to write in Jenkins to fetch the code from GitHub**

1. **To fetch the code from Master branch of GitHub**

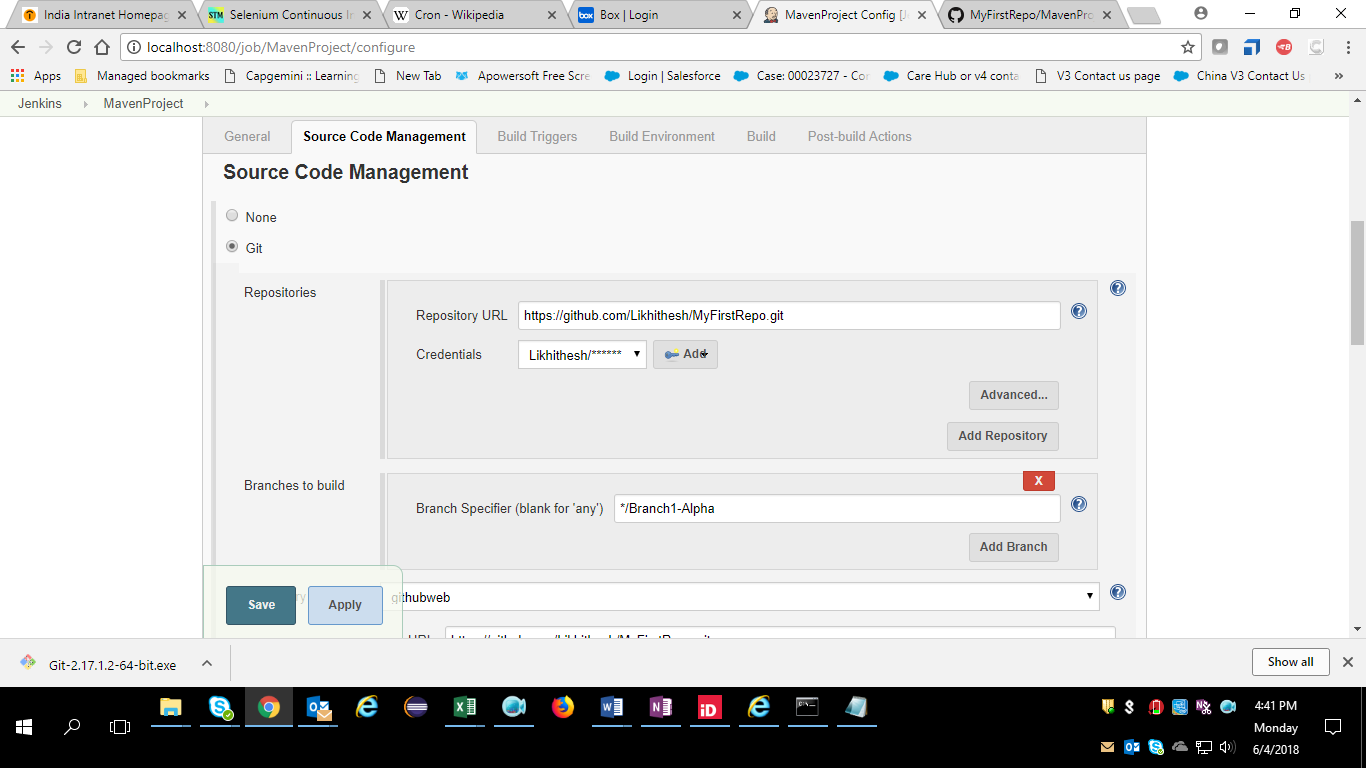
**\*/master**



1. **To fetch the code from any Other branch**

**\*/{Branch Name}**

**Ex : \*/Branch1-Alpha**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## **LL 23) Configuring Jenkins to Run a Build Automatically on Code Push to GitHub Repo**

**Solution : WAY 1 : Using Git Hooks**

Git hooks are scripts that run automatically every time a particular event occurs in a Git repository.

Common use cases for Git hooks include encouraging a commit policy, altering the project environment depending on the state of the repository, and implementing continuous integration workflows. But, since scripts are infinitely customizable, you can use Git hooks to automate or optimize virtually any aspect of your development workflow.

### **Installing Hooks**

Hooks reside in the .git/hooks **(But in my System - \Git\mingw64\share\git-core\templates\hooks)** directory of every Git repository. Git automatically populates this directory with example scripts when you initialize a repository. If you take a look inside .git/hooks **(\Git\mingw64\share\git-core\templates\hooks)**, you’ll find the following files:

applypatch-msg.sample pre-push.sample

commit-msg.sample pre-rebase.sample

post-update.sample prepare-commit-msg.sample

pre-applypatch.sample update.sample

pre-commit.sample

These represent most of the available hooks, but the .sampleextension prevents them from executing by default. To “install” a hook, all you have to do is remove the .sample extension. Or, if you’re writing a new script from scratch, you can simply add a new file matching one of the above filenames, minus the .sampleextension.