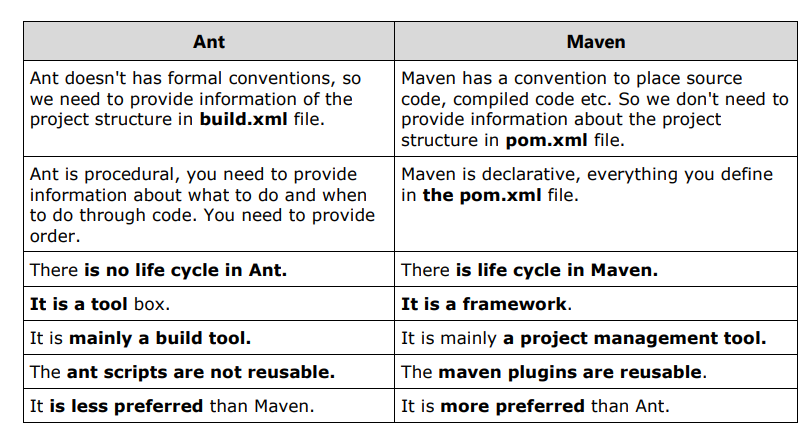
=========================MAVEN================================

Maven is basically a project management or build tool which can easily handle the entire lifecycle of a project that include code generation, compilation, testing, validation, packaging, installation and much more. It works in phases rather than tasks and makes the build management process much easier.

A [Selenium](https://www.simplilearn.com/tutorials/selenium-tutorial/selenium-ide) project for [test automation](https://www.simplilearn.com/what-is-automation-testing-article) requires you to have all the dependencies associated with it. These dependencies can be downloaded and upgraded manually. However, as the project gets bigger, it can be quite challenging to manage these dependencies. This is why you need to have build automation tools to handle them automatically.

Maven uses the concepts of the project object model (POM) and enables the user to cut down several steps followed in the build process.

* Maven is used to manage the dependencies. For example, if you are using selenium version 2.35 and any later point of time you have to use some other version, the same can be managed easily by Maven.
* It works very effectively when there is a huge number of Jar files with different versions.
* A Maven repository is a directory of packaged JAR file with **pom.xml** file. Maven searches for dependencies in the repositories.



* There are 3 types of Maven repository:-

**1) Local Repository :** Maven local repository is located in your local system. It is created by the Maven when you run any Maven command. By default, Maven local repository is **/home/luser/.m2** directory on Linux OS.

**2) Central Repository :** Maven central repository is located on the web. It has been created by the Apache Maven community itself.

**3) Remote Repository** : Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as **JBoss** library etc, so we need to define remote repository in **pom.xml** file.

Group ID : Organization name

**For example:** org.apache.maven.plugins is the designated Group-Id for all Maven plugins.

Artifact ID : Project name; The artifact for a project is typically a JAR file.

**For example:** A typical artifact produced by Maven would have the form:

<artifactId>-<version>.<extension> (for example, myapp-1.0.jar)

**Version:**  This element indicates the version of the artifact generated by the project. Maven goes a long way to help you with version management and you will often see the SNAPSHOT designator in a version.

**For example:** Version 1.0-SNAPSHOT is released as version 1.0, and the new development version is version 1.1-SNAPSHOT.

POM

* is the fundamental unit of work in Maven.
* It is an XML file that contains information about the project and configuration details used by Maven to build the project.
* Some of the configuration that can be specified in the POM are the project dependencies, the plugins(for config like complier, surefire for execution) or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

Dependencies

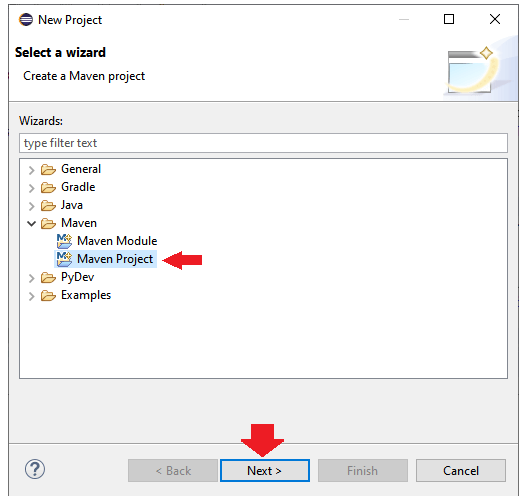
* When working on project, the external jars that we import for multiple functions are known as **project dependencies.**
* Example:- In eclipse to execute the selenium web driver program we need selenium jars.
* Maven has built-in dependency management. We can just specify in the POM file what external libraries the project depends on, and which version.
* Maven would download them and puts them in your local Maven repository.
* If these external libraries need other libraries, then Maven would download their dependency libraries into your local Maven repository as well.
* We can specify dependencies inside the dependencies element in the POM file.

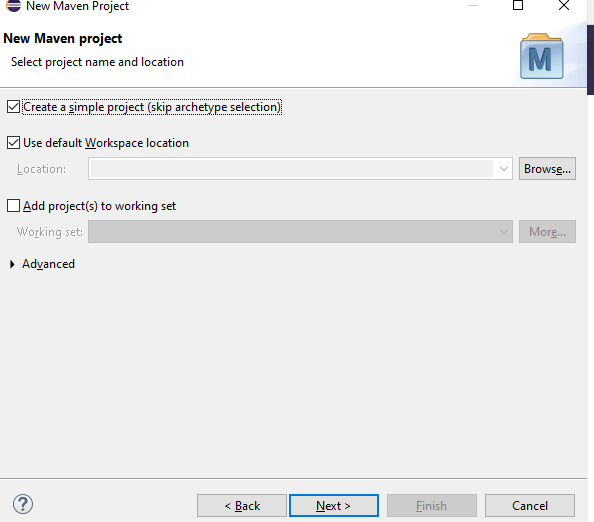


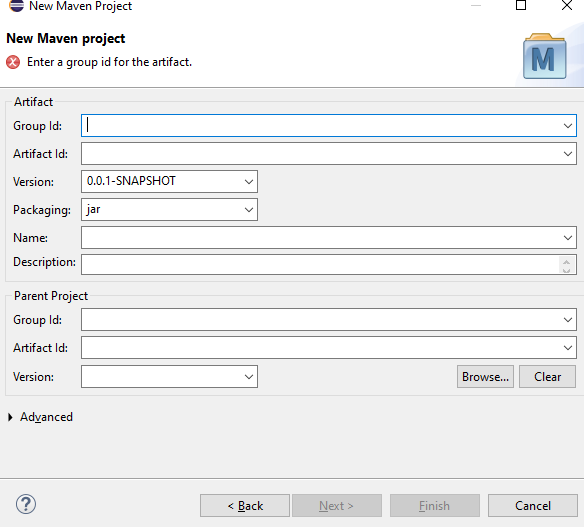
## Maven Eclipse plug-in

Maven plug-in is automatically installed in the Eclipse, but if it is not present, we will go to the **Eclipse Market Place** and search for **Maven** and download the **M2E integrated version** from there.

Create maven project

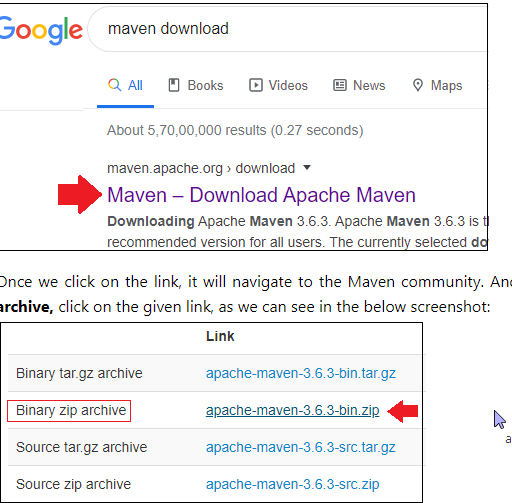






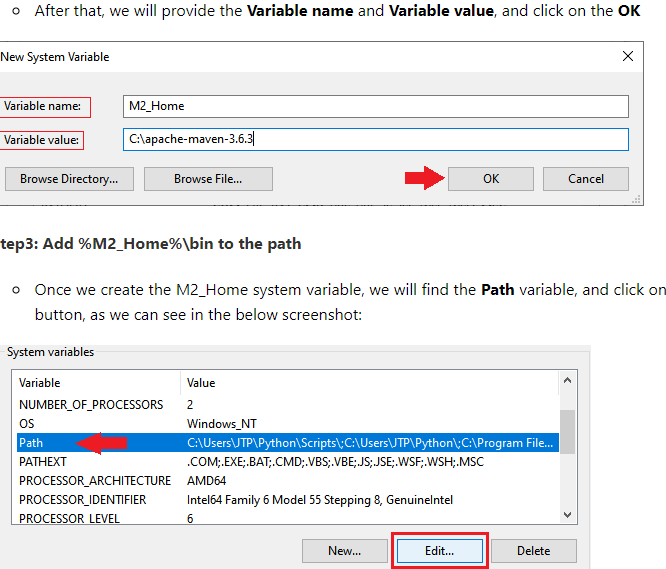
## Maven command line plug-in

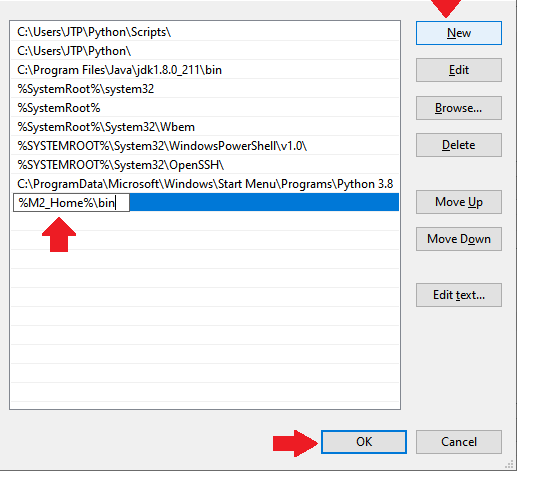
#### # Note : **Before we will start the installation process of Maven, we will make sure that Java is installed and the environment variable is also set.**

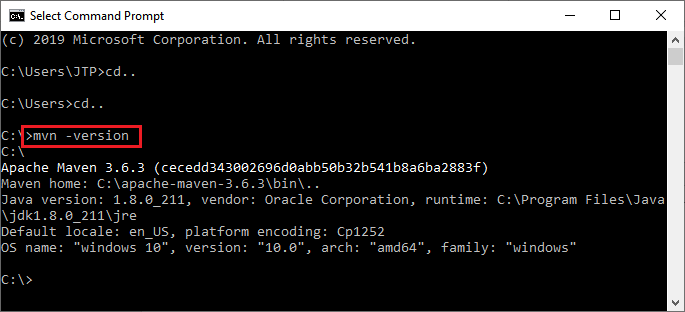


**Step2: Add M2\_Home in the System Variable**

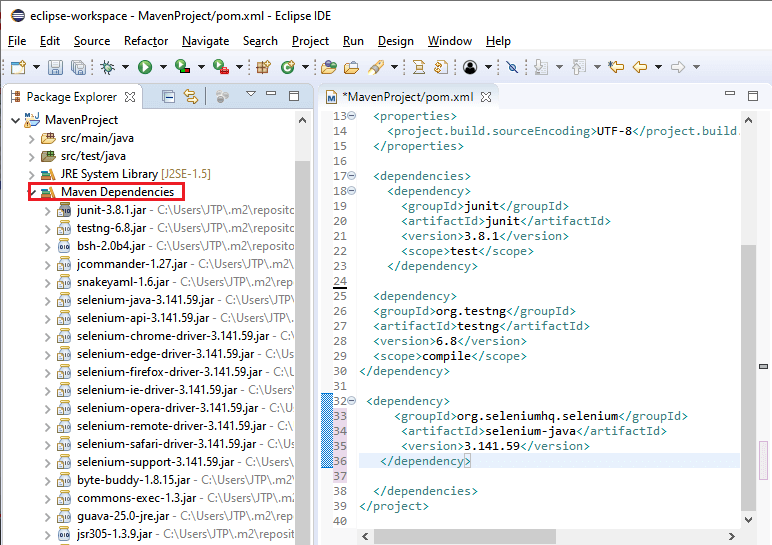
* Once we copy the location, we will go to the **Environment Variables** window and click on the **New** button on the **System variable** section, as we can see in the below image:



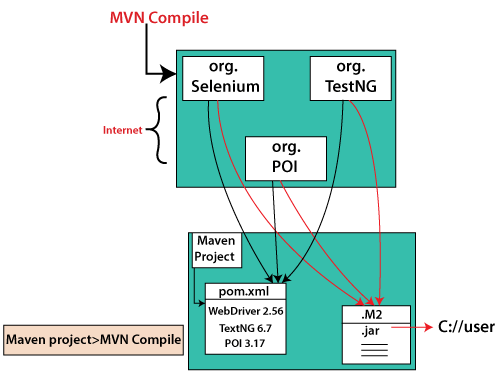




After adding dependency in pom.xml file, the jar files are added in maven dependencies folder and also in .m2 repository in system folder



The **Maven compiler plugin** is used to compile the source code of a Maven project. Maven test command will connect to the internet and download all the dependency jar into the .M2 folder local repository and then compile the entire selenium source code



The **Maven surefire plugin** is used when we have to run the unit tests of the application.

The **Maven source plugin** is used to build the jars files that were having the **.java** source

the most important directories:

- src

- main

- java

- resources

- webapp

- test

- java

- resources

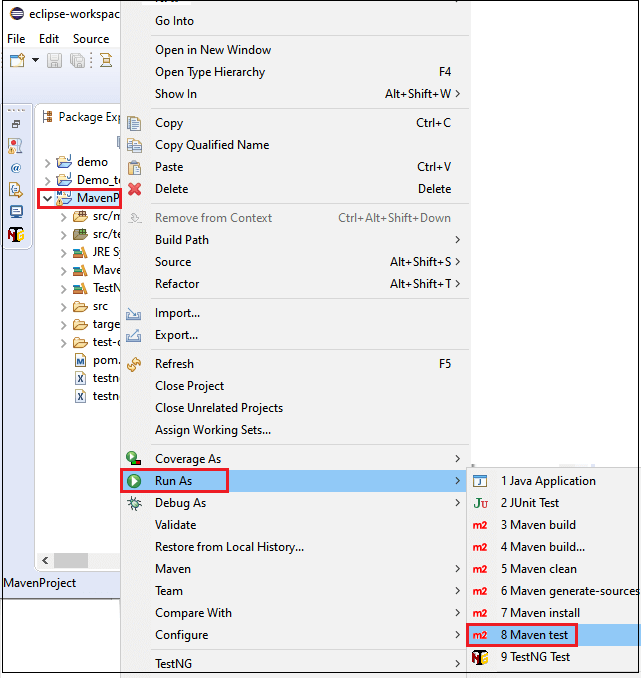
- target

The src directory is the root directory of your source code and test code. The main directory is the root directory for source code related to the application itself (not test code). The test directory contains the test source code. The java directories under main and test contains the Java code for the application itself (under main) and the Java code for the tests (under test).

The resources directory contains other resources needed by your project. This could be property files used for internationalization of an application, or something else.

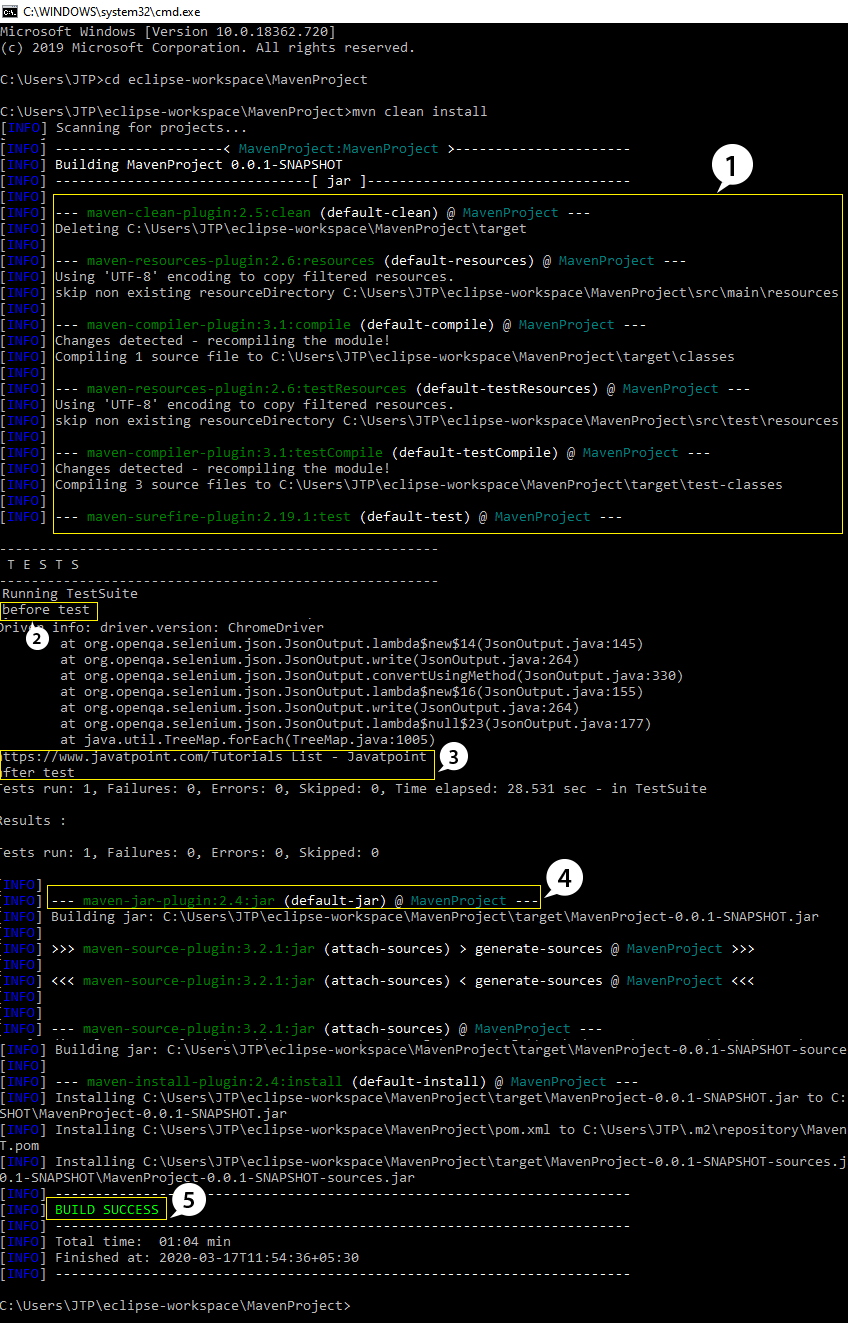
The webapp directory contains your Java web application, if your project is a web application. The webapp directory will then be the root directory of the web application. Thus the webapp directory contains the WEB-INF directory etc.

The target directory is created by Maven. It contains all the compiled classes, JAR files etc. produced by Maven. When executing the clean build phase, it is the target directory which is cleaned.



Run through cmd

tcopy the location of the **MavenProject** from properties optionand paste in the command prompt to reach the correct location of the file.



**Maven build lifecycle**

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

Here are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

For example, the default lifecycle comprises of the following phases (refer to the Lifecycle Reference)

• validate - validate the project is correct and all necessary information is available

• compile - compile the source code of the project

• test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

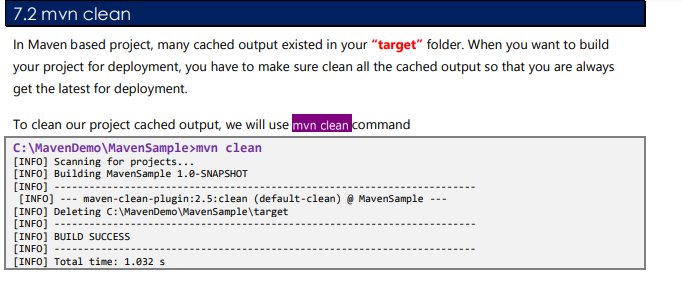
• package - take the compiled code and package it in its distributable format, such as a JAR.

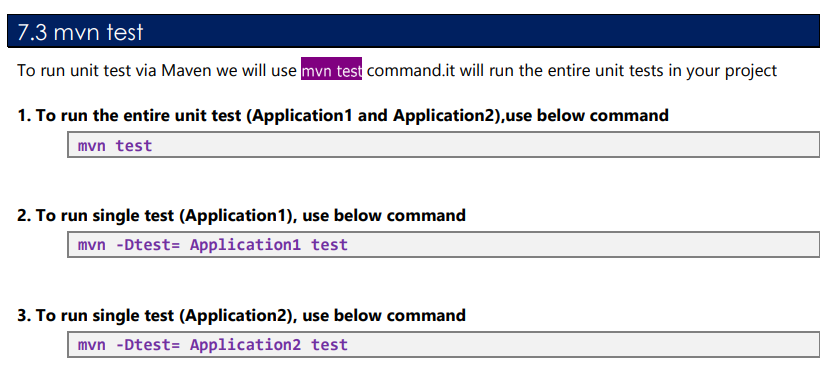
• verify - run any checks on results of integration tests to ensure quality criteria are met

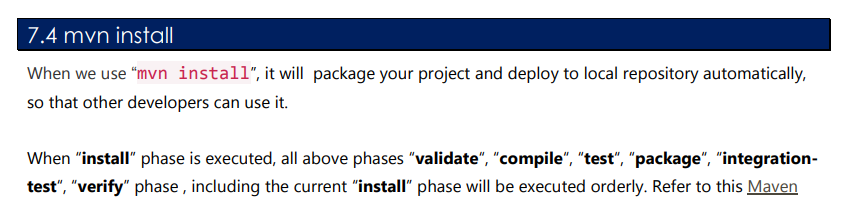
• install - install the package into the local repository, for use as a dependency in other projects locally

• deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

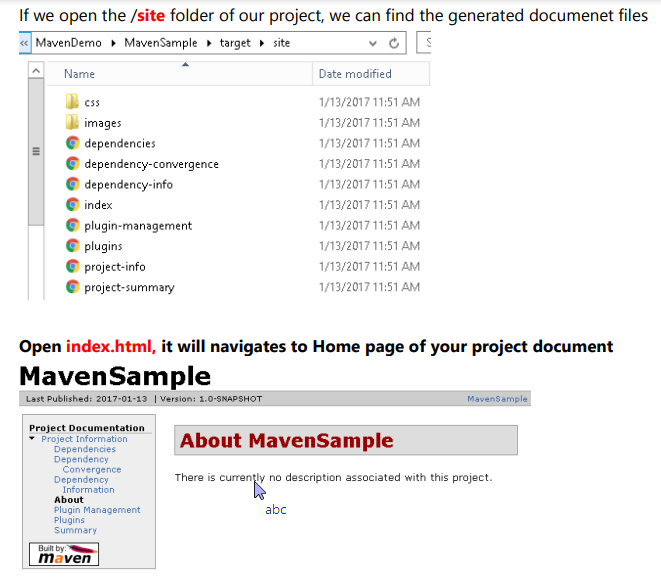
when the “package” phase is executed, all its above phases – “validate“, “compile” and “test“, including the current phase “package” will be executed orderly











What is WAR File?

The WAR file (Web Application Resource or Web Application ARchive) is a container for JAR files, JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static sites (HTML and associated files), and other resources that make up an online application

Jenkins

Continuous Integration (CI) is a development practice in which the developers are needs to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built and further this built is tested to find any kind of errors in it. The general practice is that whenever a code commit occurs, a build should be triggered. This entire process is automated thereby allowing continuous Integration

DevOps team : main purpose integration, create build(Maven s/w), Test the build with using test scripts(Jenkins s/w), Perform sanity(to check main functionalities working fine or not) and smoke testing, after build successful will get mail to both QA and Developer and final build will be ready in Jenkins itself.

Installation process

Step 1: Install java 11

Go to oracle website : <https://www.oracle.com/in/java/technologies/downloads/#java11>

Select the .exe file, enter oracle username and password to download exe file

Username :anirudhamumbarkar@gmail.com Password :Anirudha@777

To install jdk 🡪 double click on downloaded exe file (After complete installation java folder will create in c drive program files folder)

To verify the java version, open cmd java -version

Step2 : <https://www.jenkins.io/download/>

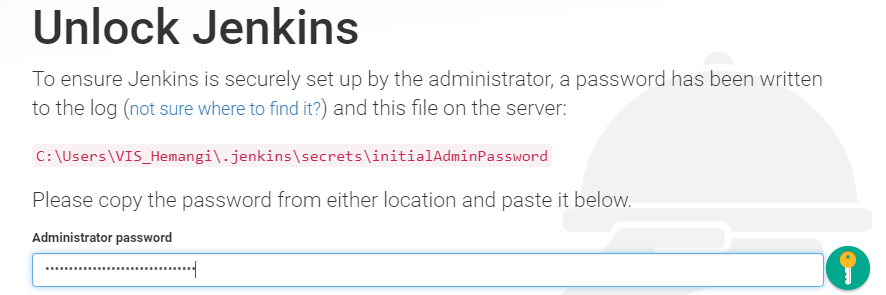


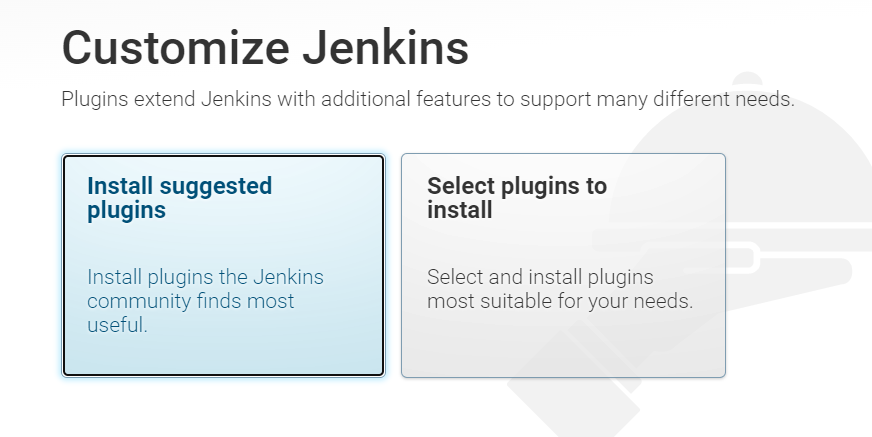
Create folder in c drive and place this war file in that

Open up a terminal/command prompt window to the download directory

Run the command java -jar jenkins.war

Browse to <http://localhost:8080> ; Enter password(.Jenkins root folder in user> Secret > Initial password)





Select plugins to install and continue; Set username and password

Step 3 : Manage jenkin > global config > set path for jdk, Git and maven thru env variables

Step 4: For assigning job to jenkin > New item > enter item name > Free style prj > ok

Step 5: under general > click advance button > click use custom workspace > copy n paste ur project workspace path in directory

Step 6: under build > add build step > execute windows batch command > in command section u can write any maven cmnds eg mvn test > save > build now

Step 7: To execute test cases periodically after specific time

* + click prj > configure > buid trigger > build periodically > Schedule (clk help ? to chk how to set schedule)
  + use \*\*\*\*\* to set periodic time as per ur requirement
  + to stop this periodic execution untic build periodically chkbox and save

Git setup with Jenkins

Step 1 : need to download git plugins as Jenkins > Manage Jenkins > Manage plugins> search Git plugins

Step 2 : Create free style project > source code management> git> Enter github Url> branch if any> save

Java n Maven with Jenkin

Step 1 : Manage Jenkins> Global config> Set Java JDK and Maven ENV variables name and path > save

Step 2 : Create maven project > under build > pom.xml path and mvn command

(mvn clean install)

# Jenkins Pipeline

In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

It is a combination of plugins that support the integration and implementation of **continuous delivery pipelines** using Jenkins.

**Pipeline:** This is the user-defined block, which contains all the processes such as build, test, deploy, etc. it is a group of all the stages in a JenkinsFile. All the stages and steps are defined in this block.

How to create the pipeline?

Step 1 : New item> Create 4 free style project> Select execute shell under build> enter script> save

Step 2 : Under build trigger deploy test project > select build project as to run initial project

Similarly for next project🡪 to make connectivity between them

Step 3 : Open all 4 projects in new tab and execute build project (another projects executes one after other)

Step 4 : add build pipeline plugins : manage Jenkins> manage plugins> build pipeline plugin in available tab> install

Step 5 : click on add button > view name > enter name and select pipeline view> select build project under upstream/downstream > no od build displayed > apply

Step 6 : run and validate

Operation:

DevOps team takes diff. codes from diff. developers which are placed in common shared repository and create an integrated build

They will take automated scripts / test cases from QA team and will execute to cases against that build

perform basic sanity and smoke testing and then after success the build is given to QA team for further testing

Again, after making the changes in the scripts (QA) and codes (dev) that will be uploaded to the common repo. and again, devops will check n so on.... this is referred as CI

Devops uses Maven to create the build and to run it Jenkin is used. QA team use limited features of MAVEN like pom.xml, and some other test related commands. But devops uses max. features of MAVEN.

Execution in Jenkin is headless ie no UI and so it is faster.

QA download the build from Jenkin into their env. For example :

* Dev will work 9am to 5pm > Devops will build n test in eve. > Then next day QA will work on that build > and at the same time dev will work further and at eod will store their codes in common repo n so on...referred as CI
* QA team don’t us MAVEN n JENKIN 100p, but as DevOps team uses our automated scripts using mostly Maven n Jenkin, so some knowledge of maven n Jenkin is must
* So, QA team must do certain configuration on Jenkins related to their scripts n verify that they r working fine on Jenkin for the first-time n then later on devops team will handle.

QA don’t install Jenkin in their env. Devops will give Jenkin Server Url to QA; Qa will only configure their test cases n all in Jenkins.

So Devops team basically helps to integrate all the process which in turn enhance the agile process fast.

===============================================================

GIT, GITHUB

we can also perform same operations on other operating systems like **Linux (Ubuntu)** and **MacOS**.

Git 🡺 Local repository, Windows based application;

developed to manage projects with high speed and efficiency. The version control system (changes can be reverted or roll back to previous version of the build and manage the builds version) allows us to monitor and work together with our team members at the same workspace.



When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.



**Maintain the clean history:** It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

GitHub 🡺 Remote Repository, Web Application

is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration.

Pre-Requisites:

Step 1 : Create new account in Github : <https://github.com/>

Create new repository in Github and capture the link

Step 2 : Download and install the git repository: <https://git-scm.com/downloads>

Commands:

**Setting username**

The username is used by the Git for each commit.

1. $ git config --global user.name "Himanshu Dubey"

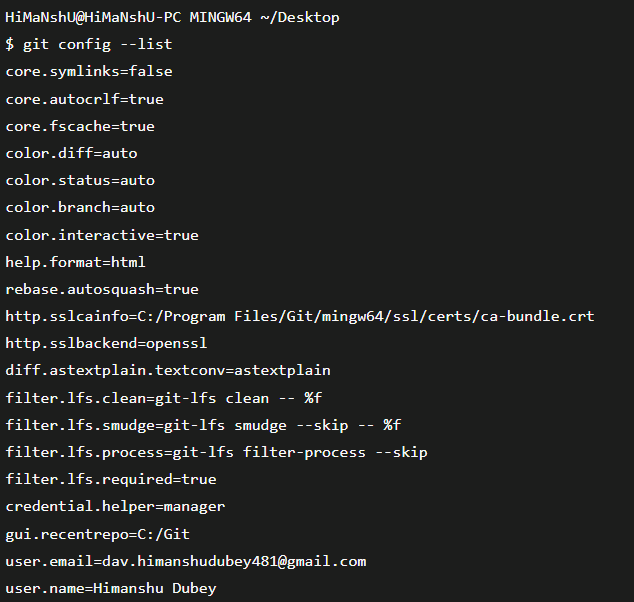
**Setting email id**

The Git uses this email id for each commit.

1. $ git config --global user.email  "himanshudubey481@gmail.com"

**Checking Your Settings**

You can check your configuration settings; you can use the **git config --list** command to list all the settings that Git can find at that point.



GIT > LOCAL REPO. (everyone shld install this git exe in their sys) . Chk sc for configuration n installation steps. Also add git bin locn path in ur env. variables

(GIT software is mandatory on the remote system where Jenkin is running)

GITHUB > REMOTE REPO. (it is a webappl type)

Codes r 1st stored in individual git repo n at the eod all codes frm diff. gits r pushed to github repo.

QA works n store codes as > eclipse workspace > git repo > github repo.

Each QA member can access another members code via github

Script is written in wrkspc > which is 1st passed to staging area > then from there it is passed to git repo > frm git repo it is pushed to github

Below Commands r used in bash.exe (D:\Program Files\Git\bin):

1.Status = It show the chngs u made in wrkspc and which files r ready to be sent to staging area & git repo

2.Add = to add the ready files to staging area.

The git add command is used to add file contents to the [Index (Staging Area)](https://www.javatpoint.com/git-index). The git add command can be run many times before making a commit($ git add newfile.txt / $ git add -A)

3.Commit = to send files to git repo frm staging area

4.Push = to send files to github frm git repo (here we pass generated URL of our github/remote repo along with UN n PW . NOTE THIS URL IS COMMON FOR ALL MEMBERS).

(WHATEVER CHNGS U WILL DO IN WRKSPACE , then after that u hv to execute those 4 cmnds everytime)

5.git init = to create a new local git repo.

The git init command creates a .git subdirectory in the current working directory. This newly created subdirectory contains all of the necessary metadata. These metadata can be categorized into objects, refs, and temp files. It also initializes a HEAD pointer for the master branch of the repository.

6.git remote add orgin = To create a connection betn local git repo n github repo. (this two git cmnds r used only once in the beginning)

7.B4 DOING COMMIT we need to execute below cmnds only for 1st time.

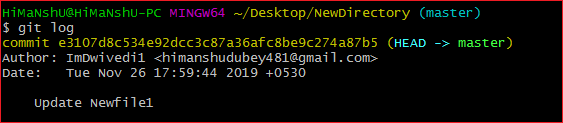
7.1 git config --global user.name "rudra" (global indicates, this un and email will be applied for all ur repos. so u frm next time onwards u no need to specify this for other repos)

7.2 git config --global user.email "ENTER THE EMAIL ID THAT U HV USED TO LOGIN GITHUB REMOTE REPO."

8.clear = to clr the screen

9.PULL = to get files (usually added by others) frm github to ur local system

10. commit : Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area.  Every commit is recorded in the master branch of the repository. We can recall the commits or revert it to the older version. Two different commits will never overwrite because each commit has its own commit-id



steps to txfr file to github:

1.Go to ur prj in eclipse n copy its wrkspc path in open that in explorer.

2.ryt clk anywhere in that locn n select git bash here option to open cmd

3.git init (enter)

4.git remote add orgin "GENERATED URL OF GITHUB" (entr) > (no msgs will b displayed on cmd console)

5.git status (UNTRACKED FILES shown in red colour = files which r not part of staging/git repo, TRACKED FILES = this files r presnt in staging /git repo.)

6.git add -A (-A = means add all files to staging area, to add any specific file specify its name after -)

7.git status = to chk files r added or not > if added successfully to staging area all files will be turned green.

8.git config --global user.name "rudra" // un n email of individual member using local git repos.

9.git config --global user.email "ENTER THE EMAIL ID THAT U HV USED TO LOGIN GITHUB REMOTE REPO."

10.git commit -m "write some comments" > (comments r mandate to show what all chngs u hv done) > if added succesfully to git repo. all files will be in white colour n

a msg showing no. of files chngd will be displayed.

11.git push -u origin master > used to send all files from git local repo to github remote repo.

12.Now it will ask for username n pw, so for un & pw = enter ur email id n pw n press entr

13.Now after 100% txfr > refresh ur github to see all ur files

How to acess other members files frm github

1.Create a dummy test file (assuming it is the file created by someone else in github) as > under Code tab select Create new file option >

> give file name n write dummy comments in edit new file section > click commit new file.

2.Execute PULL cmnd as :

git pull origin master > this will pull only those files frm github which r not present in ur local repo.

3.Refresh ur prj in eclipse ,there under src/test/java > u will see this new test script file dwnlded frm github.

(File in git is directly found in wrkspc)

From 2nd round onwards ie when u do modifcation in existing file or when u add new file to ur wrkspc. execute below cmnds

1.git status

2.git add - A

3.git status

4.git commit -m "added serach\_custm file and modified edit\_cust file"

5.git push -u origin master

provide email id n pw

6.refresh ur github to chk chngs.

How to execute our prj frm github using jenkin

1.Login jenkin

2.Create a maven prj n give name to it.> ok

3.Under Source code management select GIT > provide ur generated github url > dont do anything else

4.Under Build > Root POM = pom.xml > Goals n options > clean install > apply n save

5.Make sure u hv this config setup b4 u run ur build > Under manager jenkin > gobal tool config >

1.Jdk locn

2.Git locn > C:\program files\git\bin....(set this path in ur environment system variable)

(Also note that although Jenkin pick script n code files frm github, still it internally uses GIT software),

(the installation of GIT software is mandatory on the remote systaem where Jenkin is running(this is taken care by devops team)).

3.Set ur GIT locn path in > "Path to git executable" section in jenkin as > C:\program files\git\bin\git.exe

4.appl n save

6.go to dashboard > select ur prj > Build now > u can click on progress bar to chk the logs

## Git Undo Add >>>>to remove files from staging area

We can undo a git add operation. However, it is not a part of git add command, but we can do it through git reset command.

To undo an add operation, run the below command:

**$ git reset <filename>  or $ git reset -A**

**To Undo ur last commit > Undo the last commit with git reset --soft HEAD~1, move the mistakenly committed files back to the staging area.**

$ git log

commit cf8bf132ab2a85fa8bc33854629dcba785cd5d7e (HEAD -> master)

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:47:02 2023 +0530

hj

commit 4da980ebad6e6857d1857ca349367c5b588bea3b

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:42:24 2023 +0530

1stcommi

commit 318b9e4ad168583edb586ba5f37f2d03b8a0bf35

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:38:28 2023 +0530

1stcommit

Here we hv made 3 commits , out of which the last commit is treated as head master n the content for this commit will be pushed to github. Now suppose u hv made any wrong chngs in code n hv commited the same n now u want to roll back to ur previous commit here “1stcommi” then make it as a head master by using follwiong git reset cmnd

**$ git reset -–soft “commit id”**

abc@DESKTOP-U8N41C8 MINGW64 /d/Users/Public/ANU BACKUP/LAST REVISION NOTES (master)

$ git reset --soft 4da980ebad6e6857d1857ca349367c5b588bea3b

Now u can see “1stcommi” has became the head master.

abc@DESKTOP-U8N41C8 MINGW64 /d/Users/Public/ANU BACKUP/LAST REVISION NOTES (master)

$ git log

commit 4da980ebad6e6857d1857ca349367c5b588bea3b (HEAD -> master)

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:42:24 2023 +0530

1stcommi

commit 318b9e4ad168583edb586ba5f37f2d03b8a0bf35

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:38:28 2023 +0530

1stcommit

**>>>>>>>>>>.to chk commited files >>>>>>>>>>>>>>>>>>>>>>>**

**Use $ git show 4da980ebad6e6857d1857ca349367c5b588bea3b --name-only**

$ git log

commit 4da980ebad6e6857d1857ca349367c5b588bea3b (HEAD -> master)

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:42:24 2023 +0530

1stcommi

commit 318b9e4ad168583edb586ba5f37f2d03b8a0bf35

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:38:28 2023 +0530

1stcommit

abc@DESKTOP-U8N41C8 MINGW64 /d/Users/Public/ANU BACKUP/LAST REVISION NOTES (master)

$ git show 4da980ebad6e6857d1857ca349367c5b588bea3b --name-only

commit 4da980ebad6e6857d1857ca349367c5b588bea3b (HEAD -> master)

Author: rudra <rudra.rao@gmail.com>

Date: Wed Mar 1 17:42:24 2023 +0530

1stcommi

New Text Document.txt