1. nJDK 1.8 : <https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html>

https://docs.oracle.com/javase/8/docs/technotes/guides/install/install\_overview.html

1. Eclipse for J2EE : <https://www.eclipse.org/downloads/packages/release/indigo/sr2/eclipse-ide-java-ee-developers>
2. GIT: <https://git-scm.com/downloads>
3. Maven: <https://maven.apache.org/download.cgi>
4. Jenkins: <https://www.jenkins.io/download/>

**DAY1 Configuring Jobs**

# Starting Jenkins

* Open command prompt
* Go to war file location
* Now, fire the commond,
* Java -jar Jenkins\_new.war
* It will run by default on 8080
* Now go to Jenkins home as, C:\Users\admin\.jenkins\secrets and find the file initialAdminPassword file which contains initial password
* Go to browser hit the URL localhost:8080
* Enter username = admin and password = string copied from password file.

**Installing the Jenkins setup, create a simple Jenkins job, delete, fail, disable the job**

# Demo1 : Freestyle Project

A)

Builds for Non-Source Control Projects

There is sometimes a need to build a project simply for demonstration purposes or access to SVN/CVS repository is unavailable. By choosing to configure the project as "None" under "Source Code Management" you will have to:

* Build the Project at least once, (it will fail), but Jenkins will create the structure jenkins/workspace/PROJECTNAME/
* Copy the project files to jenkins/workspace/PROJECTNAME/
* Build again and configure appropriately

## Demo :

* Create a new job Freestyle1
* Goto Build section and add ---🡪

cd Project\_freestyle

javac MyClass.java

java MyClass

* Now build the project but it will fail.
* Now, go to drive and create a Project with any name e.g. Project\_freestyle

Add MyClass.java inside it

* Now build the job from Jenkins and everything will work

B)Using Custom workspace

Project is not using any SVN for versioning but we want it under CI/CD. In this case using custom workspace is the best choice.

## Demo

* Create a new job Demo1
* Goto general -🡪 Advanced--🡪 Use Custom workspace add the location e.g. D:\jenkins demo\freestyle
* Don’t select any Source code management and no Build Triggers
* Save the configuration
* Build the job
* Observe workspace

C)

* Click on configure (Freestyle1)
* Click on Build periodically and add \*/1 \* \* \* \*

Save the code wait for a min new build will happen automcatically

Delete once done

Cronjob documentation https://crontab.guru/

D)

* Add build steps --🡪 Execute Windows batch command

cd com\one

javac Demo1.java

java Demo1

**Building remotely**

Step1 : get token

* Goto home -🡪click on username
* Select configure
* Add token name(Token1) and click generate token . copy the text value

e.g. token1=11ce435f3425cd8a15d61864314b301684

Step 2: go to jenkins -🡪Global security

* Enable proxy compatibility as,

Goto Jenkins-🡪configure global security --🡪 CSRF protection --🡪Enable proxy compatibility

Step 3: add authorize project plugin

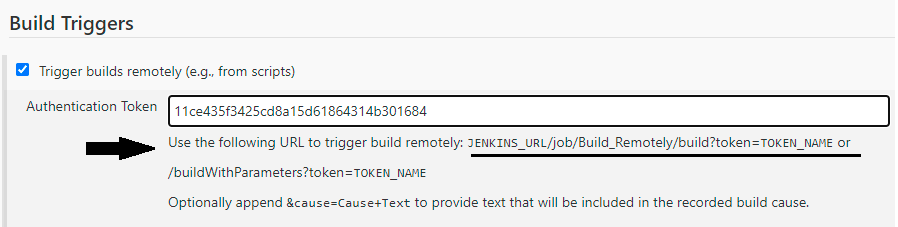
To create Project

* Add new Item🡪Freestyle project

Use SG\_FreeStyle\_workspace

Step 4: goto job select Build triggers remotely

* Add the name of token e.g. token name
* Below in text it will give the URL for that job to fire copy that



e.g. JENKINS\_URL/job/Build\_Remotely/build?token=TOKEN\_NAME which will be

localhost:8080/job/Build\_Remotely/build?token=token1

**in postman**

* Fire get request to get the jenkins crumb from =

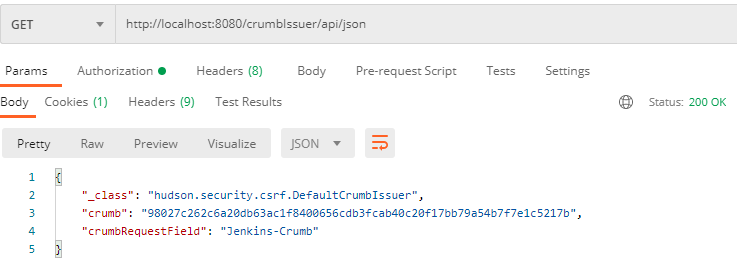
Use 🡪GET

URL ---🡪 <http://localhost:8080/crumbIssuer/api/json>

Authorization= Basic

Add Jenkins username and password

We will get the crumb value as,



Copy the crumb value

* now fire post request for

http://localhost:8080/job/SG\_FreeStyle\_workspace/build?token=token1( same URL copied from step 4)

* add authorization= Basic

admin as username and password= admin\_pass( value used at initial login)

* add header = Jenkins-crumb and value obtained from the get method

if get 403 error then dont add ?token=…in the URL

If we have job with parameters we can do

<http://localhost:8080/job/FreeStyle_Demo1/buildWithParameters?token=token1&GOAL=COMPILE>

OR u can also do,

curl -H "Jenkins-Crumb:7f2ea66e5aae6d4adac04ae05fae115d1bbce6318cc2736d03490090b089d9c5" -u admin:69c6462d79694ad981908597a2fd772c http://127.0.0.1:8080/job/FreeStyle1/build?token=token1

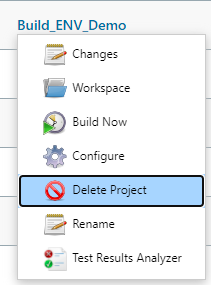
refer:--🡪 <https://medium.com/@narenchejara/trigger-jenkins-job-remotely-using-jenkins-api-20973618a493>

Locking resources

<https://medium.com/pareture/jenkins-job-locking-d34d1fbb6a0a>

After building projects in various ways we now can,

Delete the project from Jenkins home -🡪Project Name-🡪 Dropdown arrow,



Or Disable project as,

Jenkinshome -🡪select Project -🡪 on right hand top corner -🡪Disable Project

**Day2 Types of Plugins**

## Demo1 : Maven Plugin

* We will find the plugins at Jenkins Home-🡪 manage -🡪 Manage Plugins
* We can configure the Global tools at, Jenkins Home-🡪 manage -🡪 Global Tool Configuration
* Find JDK, GIT and MAVEN
* If not configured then Configure as per local path

Let’s Use Maven Plugin

**Demo1: Maven Project**

A)from command prompt generate the project as,

mvn archetype:generate -DgroupId=com.abc -DartifactId=DemoProject -DarchetypeArtifactId-maven-archetype-quickstart -Dinteractivemode=false

<https://www.soais.com/creating-a-maven-project-with-command-line/>

* In the build step set the value for pom.xml

C:\Users\admin\DemoProject\pom.xml

* Save and now build the job
* Observe the console output
* Goto drive and observe the target folder which proves compilation happens

B)

* In the post Steps
* Select Maven3
* Goal as install
* Click on advanced----🡪 add pom as C:\Users\admin\DemoProject\pom.xml
* Click on build to find the jar file created

**See how to set password for GIT**

https://www.thegeekstuff.com/2016/10/jenkins-git-setup/

**Demo1: Maven Project**

Either update the earlier project or create a new maven project

* create a mvn project in Jenkins demos folder by,

mvn archetype:generate -DgroupId=com.data -DartifactId=java-mvnproject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

* create a new job of type maven project
* in build section add,
  + Root POM value to point pom.xml as,

D:\jenkins demos\java-mvnproject\pom.xml

* GOAL and Options values to the goal such as --🡪 compile test
* execute the code
* observe the folder on drive

1. update build to add “install as goal”

* try to execute the jar file
* it will not as its not create by default executable jar

1. update pom. xml by,

<build>

<plugins>

<plugin>

<!-- Build an executable JAR -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>com.data.App</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

\*\*\*\*add build plugin always at the end to work

* execute the build once again
* go to drive in which you have jar file and execute,

java -jar java-mvnproject-1.0-SNAPSHOT.jar

* update the job now for post steps as,
* java -jar "D:\jenkins demos\java-mvnproject\target\java-mvnproject-1.0-SNAPSHOT.jar"

we are using “” bcoz the name of the directory contains spaces

* build the project and find output

**Adding the plugins to the Jenkins job, creating a Jenkins job to check for source code changes, creating maven build jobs**

\*\*\*Goto global tool setting -🡪 configure maven and JDK if not Done already

**Working With GIT**

**Adding project to Git and using Git Plugin Refer Git Integration Project**

* Generate the project of type maven in eclipse
* Configure the pom.xml as,
  + Java version

<properties>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

</properties>

* + Dependency addition

<dependencies>

<!-- https://mvnrepository.com/artifact/junit/junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<!-- Build an executable JAR -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>com.data.App</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

* and add it to git
  + go into workspace folder
  + git init
  + git add .
  + git commit -m "maven project"
  + git remote add origin <https://github.com/spunkymandar/javarepo>
  + git push -u origin master
* Create a new Job Demo1 of type maven project
* Select source code mgmt.
  + Add URL and credentials
* Click on add pre-build steps
* In Build add goal as -f projectName\pom.xml as Jenkins\_Maven\_demo\pom.xml and goal as test
* Click on save
* Save and apply
* Now build the job
* Find the console output

C)

**Now,**

* Add build step
* Add top level maven as install and
* Execuatble windows batch as,

Jenkins\_Maven\_demo\target\Jenkins\_Maven\_Demo-0.0.1-SNAPSHOT.jar

D)

* Click on configure----🡪 In **Build Triggers** -🡪 select Build periodically
* Add the cron job expression as \*/5 \* \* \* \*
* CRON JOB EXPRESSION ARE AS:::::

If you want to schedule your build every 5 minutes, this will do the job : \*/1 \* \* \* \*

If you want to schedule your build every day at 8h00, this will do the job : 0 8 \* \* \*

* Wait for 5 min to c the effect

E) Select GitHub hook trigger for GITScm polling

Steps to configure the GitHub Hook :

1. Go to repository
2. Select the settings options
3. Add web hook: The webhook should be <https://1b97-103-183-55-91.ngrok-free.app/github-webhook/> where the first part is ngrok and other is constant github-webhook.
4. To add the web hook u must add the url. Which is to be configured via ngrok
5. [Adding a GitHub Webhook in Your Jenkins Pipeline - DZone](https://dzone.com/articles/adding-a-github-webhook-in-your-jenkins-pipeline#:~:text=1%20Go%20to%20your%20project%20repository.%202%20Go,webhooks%22.%205%20Write%20the%20Payload%20URL%20as%3A%20https%3A%2F%2F228b9f82.ngrok.io%2Fgithub-webhook%2F)
6. [Windows | ngrok documentation](https://ngrok.com/docs/guides/device-gateway/windows/)
7. Very important step : add thing all the http requests routing to ngrok

ngrok http <http://localhost:8080>

this generates the static IP address for apps that are running on localhost:8080

Make sure that the antivirus is off.

1. In jenkin go to <http://localhost:8080/manage/configure> 🡪 edit Jenkin Location and change the property of Jenkins url to url of ngrok https://1b97-103-183-55-91.ngrok-free.app/

Change something in the workspace then commit that to git now it will build automatically need to add webhook in GIT

F) Now Select Poll SCM and add the same expression as above wait for 5 min again

Poll SCM periodically polls the SCM to check whether changes were made (i.e. new commits) and builds the project if new commits where pushed since the last build, whereas build periodically builds the project periodically even if nothing has changed.

\*\*\*\*

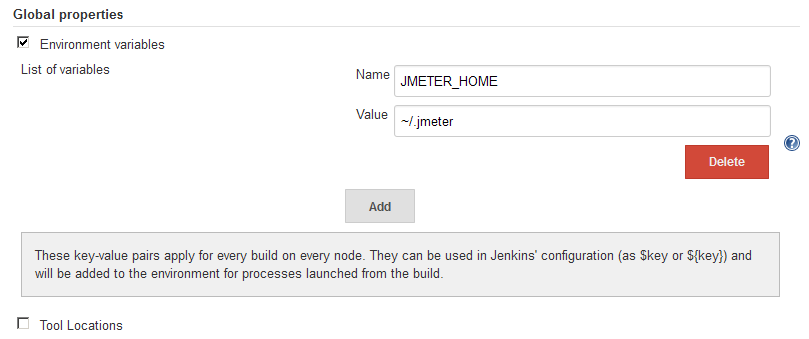
To use build number:---🡪

build-helper:parse-version versions:set '-DnewVersion=${parsedVersion.majorVersion}.${parsedVersion.minorVersion}-${BUILD\_NUMBER}'

# Build environment

**Go to ----🡪 Manage Jenkins - Configure System - Global properties**

**Add global properties and we can use them in job**



<https://www.theserverside.com/tutorial/Jenkins-environment-variables-list-for-shell-script-build-jobs>

<https://wiki.jenkins.io/display/JENKINS/Building+a+software+project>

Find all the Environmental variables at,

<http://localhost:8080/env-vars.html>

Demo 1 Using Enviornmental Variables

Refer Job--🡪 Build\_ENV\_Demo

* Go to **Go to ----🡪 Manage Jenkins - Configure System - Global properties**
* **Select the check box Enviornmental variable and click on add**
  + **Name = GLOBAL\_ENV**
  + **Value= THIS\_IS\_GLOBAL\_VARIABLE**

**Add Another,**

* + Name = GLOBAL\_ENV1
  + Value = I am from Global ENV

Now, we can use this variable in the Job

* Create a new freestyle project,
* Go to Build section and select Execute Windows batch command,

Add the content as,

echo "BUILD\_NUMBER" :: %BUILD\_NUMBER%

echo "BUILD\_ID" :: %BUILD\_ID%

echo "BUILD\_DISPLAY\_NAME" :: %BUILD\_DISPLAY\_NAME%

echo "JOB\_NAME" :: %JOB\_NAME%

echo "JOB\_BASE\_NAME" :: %JOB\_BASE\_NAME%

echo "BUILD\_TAG" :: %BUILD\_TAG%

echo "EXECUTOR\_NUMBER" :: %EXECUTOR\_NUMBER%

echo "NODE\_NAME" :: %NODE\_NAME%

echo "NODE\_LABELS" :: %NODE\_LABELS%

echo "WORKSPACE" :: %WORKSPACE%

echo "JENKINS\_HOME" :: %JENKINS\_HOME%

echo "JENKINS\_URL" :: %JENKINS\_URL%

echo "BUILD\_URL" ::%BUILD\_URL%

echo "JOB\_URL" :: %JOB\_URL%

\*\*in non windows -----🡪 $BUILD\_NUMBER  
e.g.

echo "BUILD\_NUMBER" :: $BUILD\_NUMBER  
echo "BUILD\_ID" :: $BUILD\_ID  
echo "BUILD\_DISPLAY\_NAME" :: $BUILD\_DISPLAY\_NAME  
echo "JOB\_NAME" :: $JOB\_NAME  
echo "JOB\_BASE\_NAME" :: $JOB\_BASE\_NAME  
echo "BUILD\_TAG" :: $BUILD\_TAG  
echo "EXECUTOR\_NUMBER" :: $EXECUTOR\_NUMBER  
echo "NODE\_NAME" :: $NODE\_NAME  
echo "NODE\_LABELS" :: $NODE\_LABELS  
echo "WORKSPACE" :: $WORKSPACE  
echo "JENKINS\_HOME" :: $JENKINS\_HOME  
echo "JENKINS\_URL" :: $JENKINS\_URL  
echo "BUILD\_URL" ::$BUILD\_URL  
echo "JOB\_URL" :: $JOB\_URL

* Save and build the job
* Go to console and find the console output.

# Demo2 : Adding environmental variables in Build Environment section,

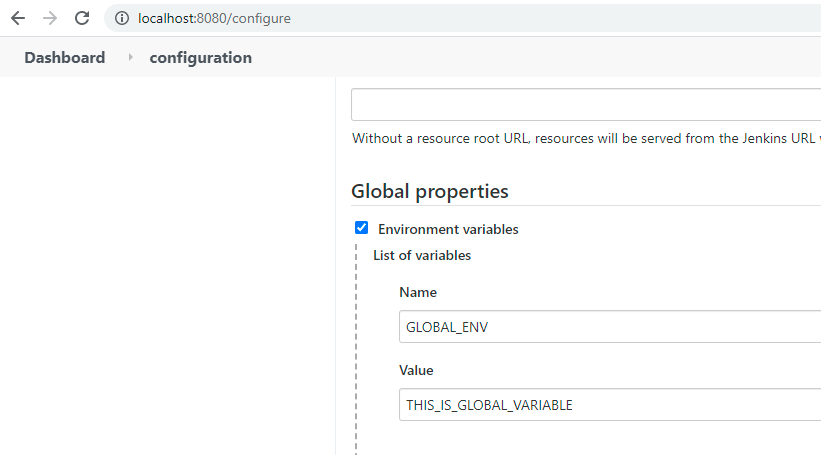
* In the above Job
* In the Build environment select Use secret texts or files
* Goto bindings select username and password (seperated)
  + Username Variable= GIT\_USERNAME
  + Password variable = GIT\_PASS
  + Add credentials
* In build section
  + Select Execute windows batch command and add,
  + echo %GIT\_USERNAME% And %GIT\_PASS%
* Build the section and find console output

# Demo 3 : Adding Global environmental variables in Build Environment section,

Go to Jenkins—manage Jenkins🡪configure system

Add Global Properties,

Name= GLOBAL\_ENV and value= THIS\_IS\_GLOBAL\_VARIABLE as,



Let’s use it

* In the above Job
* In build section
  + Select Execute windows batch command and add,

echo %GLOBAL\_ENV%

echo %GLOBAL\_ENV1%

* Build the section and find console output

create a freestyle project and then add following to execute shell

other variable available =

we can use build environment

## we can use [Environment Injector Plugin](https://plugins.jenkins.io/envinject)

Use case:

* To remove inherited environment variables (PATH, ANT\_HOME, ...) at node level (master/agent), available by default for a job run.
* To inject variables in the first step of the job (before the SCM checkout)
* To inject variables based on user parameter values
* To execute an initialization script before a SCM checkout.
* To execute an initialization script after a SCM checkout
* To inject variables as a build step obtained from a file filled in by a previous build step
* To know environment variables used for a build
* To inject build cause as environment variable for a build
* To inject environment variables from the evaluation of a Groovy script (powered by Script Security Plugin)
* To export environment variables as a metadata in your binary repository

# Demo 3 Using Enviornment Injection Plugin but for jenkin 2.4.0 and above (Not work with me)

* add the plugin to Jenkins

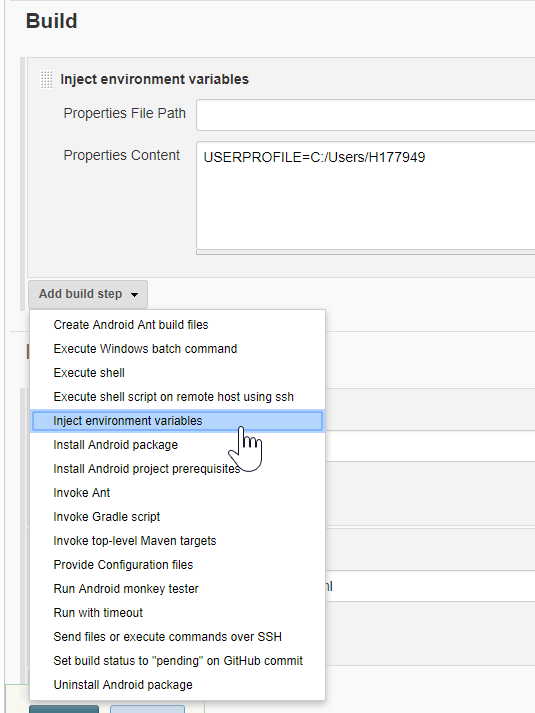
Go to Plugin Manager --🡪 Available

Type **Environment Injector**

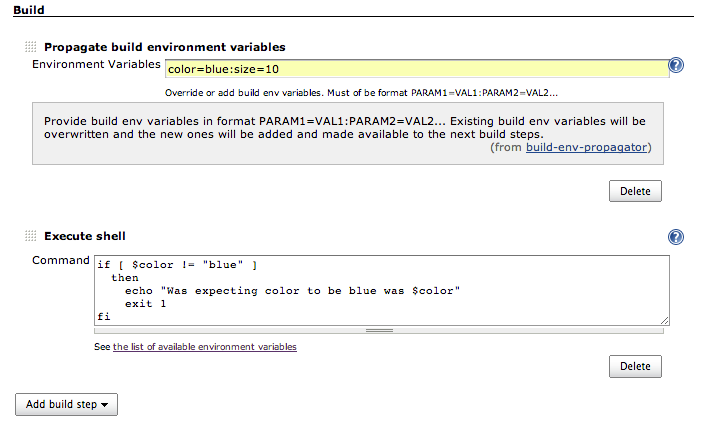
Select the checkbox and click on install without restart

* Go to your job **Configure** screen where we want to this plugin
  + Find **Add build step** in **Build** section and
  + select **Inject environment variables**

Set the desired environment variable as VARIABLE\_NAME=VALUE pattern. In my case, I changed value of USERPROFILE variable



OR



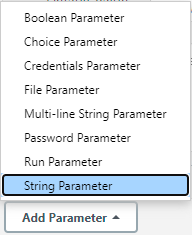
**4. Parameterized jobs Etc**

<https://wiki.jenkins.io/display/jenkins/parameterized+build>

## Demo1

Refer --🡪 Job3\_Maven\_Git / **SG\_Maven\_Git\_Parameterised**

* Create a maven project from Git URL
* Go to General -🡪 This project is parameterized
  + Click on drop down on Add parameters
  + Select String Parameter,



* + Now Add,

Name = username

Default Value = novalue

Description = your description

* + Now Select Choice Parameter and Add,

Name = GOAL

Choice =

test

install

compile

Description = your description

* In build section Root pom= Jenkins\_Maven\_demo\pom.xml

Goals and Options=$GOAL -Dusername=$username

* Under Post steps add windows batch commond as

echo %username% executing %GOAL% uses the technology Java

* Save run the build observe the output
* Now Let’s add file parameter,

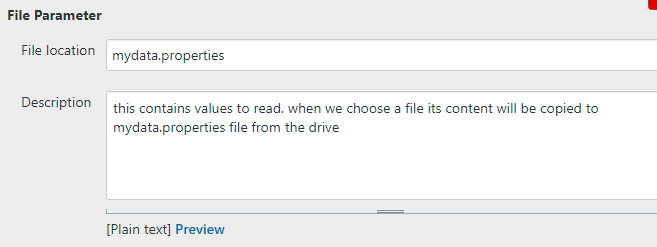
On drive create a properties file with some key values e.g.

name.technology=JAVA

version.id=2

* Goto this is a Parameterised Project
* In Add parameter select File Parameter

Add File location (It’s the location with file name in which the data from your drive will be copied in Jenkins environment) e.g. mydata.properties



* Now, update the Post steps ---🡪 windows batch commond as

echo %username% executing %GOAL% uses the technology

type myfile.properties

Build the Job and Goto console to find output

## Demo 2 Building Conditionally

## Use Display Free\_Style\_Demo1 /**SG\_Maven\_Git\_Parameterised\_Conditional**

* create a freestyle project
* use custom workspace
* General -🡪 build parameterized project -🡪 add choice

Add nam= GOAL

Choice Parameters as = COMPILE

EXECUTE

* In Pre steps select conditional step(single)
  + Select Run ? e.g String match

String 1 = $GOAL

String 2 = COMPILE

* + In Builder select the option e.g. Execute Windows batch command
  + Save and build

If we want to choode from the choices then,

* In Build conditional step(multiple)

Run----🡪 Strings match

String1 = $select

String2 = COMPILE

Steps to run --🡪 Execute windows batch command

Command = echo compiling project

cd Project\_freestyle

javac MyClass.java

* Build conditional step(multiple)

Run----🡪 Strings match

String1 = $select

String2 = EXECUTE

Steps to run --🡪 Execute windows batch command

Command = echo running project

cd Project\_freestyle

java MyClass

refer :--🡪 <https://www.serverkaka.com/2018/08/build-jenkins-job-with-condition-and.html>

## Add Post Build actions

To add the Java Doc in code file use,

**creating Javadoc use job Freestyle\_Demo1**

* open project D:\jenkins demos\Project\_freestyle
* add the following in the MyClass.java

/\*

\* @author ABC XYZ

\* @version 1.0

\* @since 2020-01-01

\*

\*

\*/

* goto job-🡪 build -🡪execute windows batch command

javadoc -d doc MyClass.java

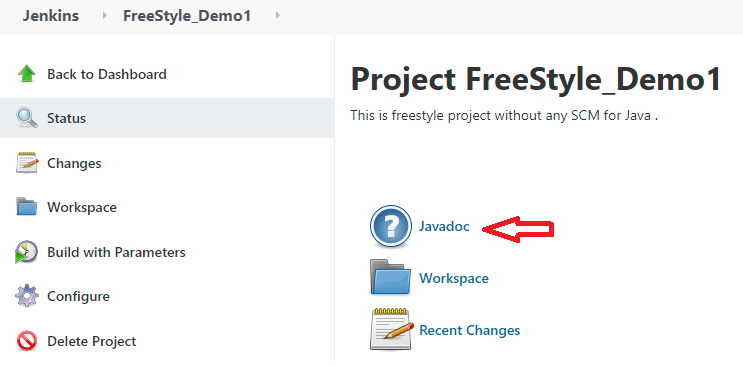
* add post build action as,

Publish java doc

Java doc directory = doc

* when we build the project on home screen of that job we will get Javadoc link

clicking on which we will link java docs



## Demo 3 Adding File parameters

* In Job3\_Maven\_Git go to parameterized project
* Select Add Parameter -🡪 click File parameter
* File location = mydata.properties
* Add description
* Click on save.
* Go to build with parameters
* Now in File parameter when you select a file it will be saved on Jenkins with name= mydata.properties

## Demo 4 Adding and Using Credentials

**Adding credentials of type “secrete text”**

## Creating Global credentials

* Go to Jenkins home-🡪 on right top -🡪 select your user name
* On left Panel you will find Credentials --🡪 click on that
* Go to Stores scoped --🡪 click on Domain -🡪 in our case (global)
* On left click on credentials
* Scope = Global
  + Kind= Secret Text
  + Secrete = “add a secrete text here of your choice”
  + ID= “add secrete” e.g. secret\_text
  + Description =” add a description here”
  + Click on add
    - Note: be sure keep the default “Global” scope for credentials that need to be accessible to build jobs. \*
    - \*Note: the complete list of secret types can vary from one instance to another as other plugins can contribute to secret types.
    - \*Note: be sure to specify the ID for the credential because that is how it is referenced in the job when using the  
      parameter expression option. \*
* Now go to job,
  + Select General--🡪 This project is parameterized
  + Click on add parameter-🡪 credentials parameters
    - Name= sec\_text
    - Credential type = Secret text
    - Default value= select from drop down -🡪 secrete text which we configure in earlier step
    - Add description
* Its time to use the secrete now,
  + **Way 1**
  + Go to Post steps
  + Select Execute windows batch command
    - Add the command as ---🡪 echo My secret is %sec\_text%
  + **Way2**
    - Goto build environment
    - Select Use secrete text
    - Click on add
    - Add variable = data\_sec\_text
    - Select specific credentials
  + **Way 3:**

withCredentials([usernamePassword(credentialsId: 'hello-kb', passwordVariable: 'pass', usernameVariable: 'user')]) {

// the code in here can access $pass and $user

}

<https://support.cloudbees.com/hc/en-us/articles/203802500-Injecting-Secrets-into-Jenkins-Build-Jobs>

**Parameters from File**

Define a .properties file

Add values in terms of key= value

We can use in pipeline

stage**(**'reading from a file'**)** **{**

steps **{**

*// Here's an example of downloading a properties file*

sh 'curl -OL https://raw.githubusercontent.com/monodot/jenkins-demos/master/environment-variables/extravars.properties'

*// Use a script block to do custom scripting*

script **{**

**def** props **=** readProperties **file:** 'extravars.properties'

env**.**WEATHER **=** props**.**WEATHER

**}**

sh "echo The weather is $WEATHER"

**}**

**}**

pipeline {

agent any

parameters {

base64File 'small'

stashedFile 'large'

}

stages {

stage('Example') {

steps {

withFileParameter('small') {

sh 'cat $small'

}

unstash 'large'

sh 'cat large'

}

}

}

}

**Email Configuration**

**Set up two step verification**

1. Visit link -🡪 <https://www.google.com/landing/2step/>
2. Click on get started and then login to your account
3. Again click on get started

<https://support.outreach.io/hc/en-us/articles/206126307-Set-Up-Gmail-Two-Step-Authentication>

Use project Git\_Integration

Manage Jenkins--🡪 Email notification

Smtp server as = smtp.gmail.com

Select Use SMTP Authentication = [tejaswini.jog@gmail.com](mailto:tejaswini.jog@gmail.com)

Add password

Select Use TLS

SMTP Port= 587

Select Test email

Add receipt : email id to whom mail to send

Now, in the job -🡪configuration -🡪 Build settings -🡪send email

**If you get address not configured yet in the email address then**

Jenkins-🡪manage Jenkins-🡪system configuration---🡪 Jenkins Location --🡪 add the email address

\*\*\*email will be sent if job fail-🡪goto test file and add assert which will fail

**Email with content**

Git integration project

* Add email Extension and Email extension Template
* Goto job -🡪 Post Build Action-🡪Edutable Email Notification

Project From : [tejaswini.jog@gmail.com](mailto:tejaswini.jog@gmail.com)

Project Receipt List : [mandarmjog@gmail.com](mailto:mandarmjog@gmail.com)

Project Reply-To List : [tejaswini.jog@gmail.com](mailto:tejaswini.jog@gmail.com)

Subject : Build failure

Default Content : The build has failed

Attach Build Log

* Click on advance settings and add Trigger

Select Always

Select Send to List = Developers and Receipt List

* Save build and check email

**Security**

<https://www.tutorialspoint.com/jenkins/jenkins_security.htm>

Adding new users

* Jenkins-🡪manage Jenkins-🡪manage users
* Add new users
* Logout and login with new credentials
* If U logged in with new credentials now logged out to use role based authorization

**Role Based Authorization**

https://www.guru99.com/create-users-manage-permissions.html

**Role based Authorization**

1. **Adding Roles :**Add a new plugin Role-based Authorization Strategy

# Demo1

* Goto Manage Jenkins--🡪 Global Security--🡪 Authorization --🡪

Select **Role-Based Strategy**

* Goto Manage Jenkins--🡪 Security --🡪 Manage and assign Roles ---🡪 Manage Roles

## Step 1 :Add Roles

* Type the name of the role in the Text fields. E.g developer, tester, devops etc
* Here we will add “developer” and click on “add button”
* Now we can add the new role applicable to Items
* e.g. role name= java-role with Pattern =job.\* and
* role name= java-role1 with Pattern =job.\*
* now you can add node roles as
* node\_role with patter node\*
* \*\*\*\* don’t add (\*node\* ) as first \* creates issues

## Step 2 Create Roles

* Now its time to assign role
* Goto manage roles-🡪 assign roles--🡪 add a global role or group in the text field. This is the user which we created under security--🡪 Manage Users. Now, click on add to add a new Role

## Step 3 Assign Role

* Once the user is added select the checkboxes in front of it to decide the role so as to indirectly decide the responsibilities
* Same ways add permissions to further roles

**Matrics based security**

* Goto Configure Global Security ---🡪 Authorization --🡪 Select radio button for Matrix based security
* Clilck on add user or Group
* Now add the name of user
* A new row will be added
* In this row select what that user can be able to do in overall , Job , Job run etc operations.

https://www.thegeekstuff.com/2016/06/jenkins-security/

**Unit and integration testing, ignoring the tests, display test results**

**Add Jmeter plugin Add a JUnit Plugin, TestComplete xUnit Plugin, test analyzer plugin**

Refer MyProject1 or Job4 Testreport

<https://github.com/java-tejaswini/Jenkins_Maven.git>

Demo 1 / **Jenkins\_Maven\_Demo**

* Add the plugins JUnit Plugin
* Job -🡪 Build --🡪 Invoke top level Maven targets
* Set Goal as -🡪 -f MyProject1/pom.xml test
* In Post Build action -🡪 Publish Junit Test result report
* Add Test Report XML --🡪 MyProject1/target/surefire-reports/\*.xml

This is ProjectName\_in\_workpace/target/folder/\*.xml

* Build the job
* Go to workspace and find the [Jenkins\_Maven\_Demo](http://localhost:8080/job/Job4_TestReport/ws/Jenkins_Maven_Demo/)**/**[target](http://localhost:8080/job/Job4_TestReport/ws/Jenkins_Maven_Demo/target/)**/ surefire-reports** and find the xmls

B) PreSteps: Process xUnit test result Reports

* Report type: Junit
* Pattern -----🡪 Jenkins\_Maven\_Demo\target\surefire-reports\\*.xml
* Select last three check boxes
* Add Threshold 1 , 1, 10 , 10 in each text field

Build --🡪

* Jenkins\_Maven\_demo\pom.xml
* Test as goal
* Save and build

## Demo 2:

* Goto Build step and add,

Goals and options as --🡪 test surefire-report:report

OR

**Testing**

**It needs the installation of TestComplete we will not go for this**

* Add a JUnit Plugin
* Add TestComplete xUnit
* Create a new job from Git as maven project
* Add URL and credentials
* Build triggers --🡪 Build whenever a SNAPSHOT dependency is built
* Build step ----🡪

Root POM = Jenkins\_Maven\_demo\pom.xml

GOAL and options = test

* Add Post Build actions -🡪 Process xUnit test result report

Add pattern as --🡪 Jenkins\_Maven\_Demo\target\surefire-reports\\*.xml

Add threshold

Select Fail the build if test results were not updated this run, Delete temporary JUnit files, Stop and set the build status to failed if there are errors when processing a result filecheckboxes

Automation Testing with Selenium

## Demo 1

Refer Selenium\_Test\_Maven Project

* Download chromedriver.exe from https://chromedriver.chromium.org/downloads
* **In Jenkins add selenium plugin and selenium HTML report plugin**
* Add the dependency in the maven project as,

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.10.0</version>

</dependency>

* Now, add a selenium test in src/test/java as

**import** org.junit.After;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Ignore;

**import** org.junit.Test;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** JenkinsDemo

{

**private** **static** String *Base\_Url* = "https://www.facebook.com";

**private** WebDriver driver;

@Before

**public** **void** setUp()

{

driver = **new** ChromeDriver();

driver.get(*Base\_Url*);

}

@After

**public** **void** after()

{

driver.quit();

}

@Test

**public** **void** testCasePassed()

{

Assert.*assertTrue*(driver.findElement(By.*xpath*("//form[@id='login\_form']")).isDisplayed());

}

@Test

**public** **void** testCaseFailed()

{

Assert.*assertTrue*(driver.findElement(By.*xpath*("//form[@id='failed case']")).isDisplayed());

}

@Ignore

@Test

**public** **void** testCaseIgnored()

{

Assert.*assertTrue*(driver.findElement(By.*xpath*("//form[@id='ignored case']")).isDisplayed());

}

}

* Save the code and commit to GIT

**In JENKINS**

* Create a new maven job
* Add url as =https://github.com/java-tejaswini/Jenkins\_Maven.git with credentials
* In the build step add pom =Jenkins\_Maven\_demo\pom.xml

Goals and options as install -Dtest=JenkinsDemo -Dwebdriver.chrome.driver=D:/setups/chromedriver\_win32/chromedriver.exe

* Click on save and build the job

## Demo2

For Publishing reports----🡪

Refer Project ---- > JUnit Test report

Create maven porjct using <https://github.com/java-tejaswini/Jenkins_Maven.git>

Project will contain following pom.xml

|  |
| --- |
| <properties> |
|  | <maven.compiler.target>1.8</maven.compiler.target> |
|  | <maven.compiler.source>1.8</maven.compiler.source> |
|  | </properties> |
|  |  |
|  | <dependencies> |
|  | <!-- https://mvnrepository.com/artifact/junit/junit --> |
|  | <dependency> |
|  | <groupId>junit</groupId> |
|  | <artifactId>junit</artifactId> |
|  | <version>4.12</version> |
|  | <scope>test</scope> |
|  | </dependency> |
|  |  |
|  | <dependency> |
|  | <groupId>org.seleniumhq.selenium</groupId> |
|  | <artifactId>selenium-java</artifactId> |
|  | <version>3.10.0</version> |
|  | </dependency> |
|  |  |
|  | </dependencies> |
|  |  |
|  |  |
|  | <build> |
|  | <plugins> |
|  | <plugin> |
|  | <groupId>com.lazerycode.jmeter</groupId> |
|  | <artifactId>jmeter-maven-plugin</artifactId> |
|  | <version>1.10.1</version> |
|  | <executions> |
|  | <execution> |
|  | <id>jmeter-tests</id> |
|  | <phase>verify</phase> |
|  | <goals> |
|  | <goal>jmeter</goal> |
|  | </goals> |
|  | <configuration> |
|  | <testFilesIncluded> |
|  | <jMeterTestFile>Test.jmx</jMeterTestFile> |
|  | </testFilesIncluded> |
|  | <testResultsTimestamp>false</testResultsTimestamp> |
|  | <propertiesUser> |
|  | <Threadcount>2</Threadcount> |
|  | <Loopcount>1</Loopcount> |
|  | <Rampup>1}</Rampup> |
|  | </propertiesUser> |
|  | </configuration> |
|  | </execution> |
|  | </executions> |
|  | </plugin> |
|  |  |
|  | <plugin> |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-site-plugin</artifactId> |
|  | <version>3.7.1</version> |
|  | </plugin> |
|  | <plugin> |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-surefire-plugin</artifactId> |
|  | <version>3.0.0-M4</version> |
|  | </plugin> |
|  |  |
|  |  |
|  | </plugins> |
|  | </build> |
|  | <!-- --> |
|  | <reporting> |
|  | <plugins> |
|  | <plugin> |
|  | <groupId>org.apache.maven.plugins</groupId> |
|  | <artifactId>maven-surefire-report-plugin</artifactId> |
|  | <version>3.0.0-M4</version> |
|  | </plugin> |
|  | </plugins> |
|  | </reporting> |

Add Junit test case to generate report

Now in Job

Build step

Pom as 🡪Jenkins\_Maven\_Demo\pom.xml

Goal -🡪 clean test site

In Post build actions

aggregate downstream test result

Select both check boxes

Publish html report

Html directory= Jenkins\_Maven\_Demo/target/surefire-reports

Index page= surefire-report.html

Report title= Myreport

\*\* on excecution of junit it will generate /site folder. which will contain project information as well as report ni html format “surefire-report.html”

**JMETER Integration Performance Trends**

https://www.blazemeter.com/blog/how-to-use-the-jenkins-performance-plugin

# Demo

Refer to Jenkins\_Maven\_JMeter

* Goto --🡪 <http://jmeter.apache.org/download_jmeter.cgi>
* Download .zip from binaries

Demos on JMETER at-🡪 https://www.guru99.com/hands-on-with-jmeter-gui.html

Refer job JMETER APP

* Create Freestyle Job
* Build parameterized project as,
* String parameter
* SCRIPT\_PATH =D:\setups\apache-jmeter-5.3\apache-jmeter-5.3\extras\Test.jmx
* THREADS = 1
* RAMPUP = 1
* RESULT\_PATH = D:\setups\apache-jmeter-5.3\apache-jmeter-5.3\bin\myresult2.jtl
* Source code mgmt. = None
* Build ---🡪 Execute Windows batch command

D:\setups\apache-jmeter-5.3\apache-jmeter-5.3\bin\jmeter -n -t %SCRIPT\_PATH% -l %RESULT\_PATH%

* Post Build Actions -🡪 Publish Performance test result report
  + Source file =$RESULT\_PATH
  + Evaluation mode = standard
* Save

**To create Maven Jmeter Integration**

## Refer Jenkins\_Jemeter\_Maven project to refer = Jenkins\_Maven\_Demo

* In the pom.xml add

<build>

<plugins>

<plugin>

<groupId>com.lazerycode.jmeter</groupId>

<artifactId>jmeter-maven-plugin</artifactId>

<version>1.10.1</version>

<executions>

<execution>

<id>jmeter-tests</id>

<phase>verify</phase>

<goals>

<goal>jmeter</goal>

</goals>

<configuration>

<testFilesIncluded>

<jMeterTestFile>Test.jmx</jMeterTestFile>

</testFilesIncluded>

<testResultsTimestamp>false</testResultsTimestamp>

<propertiesUser>

<Threadcount>2</Threadcount>

<Loopcount>1</Loopcount>

<Rampup>1}</Rampup>

</propertiesUser>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

* Create a maven job from Git URL
* In prestep

-f Jenkins\_Maven\_demo\pom.xml test

* Build --🡪 Root POM= Jenkins\_Maven\_demo\pom.xml

GOAL ==== -Dperformancetest.threadCount=3 install test

* POST Build actions -🡪 publish Performance Test result report

Source data files=Jenkins\_Maven\_Demo/target/jmeter/results/Test.jtl

Standard mode

* Save build

**Distributed execution** building multimode system

**Configuring slave node in your Jenkins, adding labels to nodes, managing the nodes**

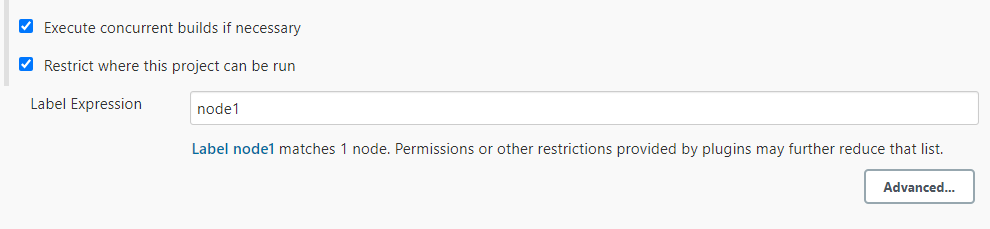
**Adding a new node**

* Open command prompt and type

java -DJENKINS\_HOME=C:\Users\admin\jenkins1 -jar jenkins\_new.war --httpPort=8081

* Now,lets add an agent

Goto manage Jenkins--🡪mange node ---🡪 click on new node

* Provide name as node1
* Description
* Remote directlry C:\Users\admin\jenkins1
* This is we used as Jenkins home in command as well
* Label =node1
* Launch method= Launch agent by connecting it to the master
* Click on save
* In status it will show launch agent button
* On clicking a .jnlp file will be downloaded . double click the file so that agent will be connected to the master
* Now select any job and find restrict run n a node….
* Set label as node1 which is label we have given.
* 
* Goto build and run it. In log we will observe….

Building remotely on [node1](http://localhost:8080/computer/node1) in workspace C:\Users\admin\jenkins1\workspace\Job3\_Maven\_Git

We can also add tools for that node in configuration of node1-🡪tools list for jdk n maven

\*\*\* <https://wiki.jenkins.io/display/JENKINS/Step+by+step+guide+to+set+up+master+and+agent+machines+on+Windows>

Maintaining and deploying Jenkins

**Backups**

Install backup plugin

* Goto Manage Jenkins--🡪 Backup Manager

1. Click on setup

* Add backup directory = d:\jenkins demos\backup
* Filename template = [bakup\_@date@.@extension@](mailto:bakup_@date@.@extension@)
* Select backupjob workspace as backup content
* Click on save

1. Now select Hudson backup configuration

* Click on it
* It will execute some commands
* And a backup file is created

1. When we want to recover from backup we can select restore from backup , select the zip file and click on restore

**Migrating Jenkins from one server to another.**

1. Copy all the files in your **JENKINS\_HOME** directory over to the new server.

2. Point **JENKINS\_HOME** on the new server at the new directory.

3. Copy the Jenkins war file (or your servlet container setup if you have one) over to the new machine and start it up.

All Jenkins settings, jobs, plugins, config, etc. live in JENKINS\_HOME.

You just need a copy of it to start it elsewhere.

Automatic deployment

Refer Demo1

Ngrok download = https://ngrok.com/download

<https://www.jdev.it/deploying-your-war-file-from-jenkins-to-tomcat/>

https://medium.com/appfleet/deploy-application-using-jenkins-tomcat-server-and-pollscm-4d358fa0d6ee

* Add Deploy to container Plugin
* In tomcat directory
  + /conf/server.xml change the port by

<Connector connectionTimeout="20000" port="8083" protocol="HTTP/1.1" redirectPort="8443"/>

* + /conf/tomcat-users.xml add

<role rolename="manager-gui"/>

<role rolename="admin-gui"/>

<role rolename="manager-script"/>

<user password="tomcat" roles="manager-gui,admin-gui,manager-script,admin" username="tomcat"/>

* Start the tomcat (the configuration is used for demo1\_works. See servers/Tomcat 7.0)
* Create the war file and add that to the location
* Start ngrok so that Jenkins will use the tomcat url as,
  + Open command prompt.
  + Goto ngrok.exe location and type

ngrok.exe http 80

* In the Jenkins Job
* In Build environment add

xcopy "D:\jenkins demos\war" "D:\jenkins demos\freestyle" /Y

xcopy "folder with location from whom to copy file" "destination" /Y

* POST Build Action
  + Name of war = Deploy.war
  + Data
  + Add credentials
  + <https://76e29d791f88.ngrok.io/manager>

Scaling Jenkins = <https://www.jenkins.io/doc/xbook/scaling/architecting-for-scale/>

HTML Publisher

Add HTML Publisher plugin

Declarative pipeline

USE --🡪 <https://tomd.xyz/jenkins-env-vars/>

List environment variables

pipeline {

agent any

stages {

stage("Env Variables") {

steps {

sh "printenv"

}

}

}

}

Reading environmental variable

Working 1

pipeline **{**

agent any

environment **{**

FAVOURITE\_FRUIT **=** 'tomato'

**}**

stages **{**

stage**(**'echo env vars'**)** **{**

steps **{**

*// This prints out all environment variables, sorted alphabetically*

*// You should see the variables declared further below*

sh 'printenv | sort'

sh "echo I like to eat ${FAVOURITE\_EGG\_TYPE} eggs"

sh "echo And I like to drink ${FAVOURITE\_DRINK}"

sh "echo My favourite city is ${FAVOURITE\_CITY}"

*// Build Parameters are also set as environment variables in the shell.*

sh "echo The worst GoT character is: ${WORST\_THRONES\_CHARACTER}"

*// We can also access some of the built-in environment variables*

sh "echo My hostname is: ${HOSTNAME}"

*// Environment variables can be overridden within a certain block*

withEnv**([**'FAVOURITE\_CITY=Portland'**])** **{**

sh "echo My favourite city is temporarily ${FAVOURITE\_CITY}"

**}**

**}**

*// This block is evaluated before executing the steps block*

environment **{**

FAVOURITE\_EGG\_TYPE **=** "poached"

FAVOURITE\_DRINK **=** "sauvignon blanc"

FAVOURITE\_CITY **=** "London"

FAVOURITE\_FRUIT **=** "satsuma"

**}**

**}**

stage**(**'second stage'**)** **{**

steps **{**

*// This will echo tomato, because the env var was set at the global scope*

sh 'echo My favourite fruit is ${FAVOURITE\_FRUIT}'

**}**

**}**

**}**

**}**

**Use it for environmental variables**

<https://e.printstacktrace.blog/jenkins-pipeline-environment-variables-the-definitive-guide/>

PIPELINE

Demo 1

pipeline {

agent any

stages {

stage('One') {

steps {

echo 'Hi,welcome to pipeline'

}

}

stage('Two') {

steps {

input('Do you want to proceed?')

}

}

stage('Three') {

when {

not {

branch "master"

}

}

steps {

echo "Hello"

}

}

stage('Four') {

parallel {

stage('Unit Test') {

steps {

echo "Running the unit test..."

}

}

}

}

}

}

pipeline {

agent any

stages {

stage("Env Variables") {

steps {

echo "The build number is ${env.BUILD\_NUMBER}"

echo "You can also use \${BUILD\_NUMBER} -> ${BUILD\_NUMBER}"

echo "I can access $BUILD\_NUMBER in shell command as well."

}

}

}

}

pipeline {

agent any

stages {

stage("Env Variables") {

steps {

bat('set')

}

}

}

}

OR IN MAC

pipeline {

agent any

stages {

stage("Env Variables") {

steps {

sh "printenv"

}

}

}

}

Create a new pipeline project

Select this is a parameterised project checkbox

Add creadential parameter

Name= GIT\_CRED

Credential type = username with password

Add default value

Add another parameter of type Password parameter

Name= GIT\_PASS

Add password value

Now goto pipeline section

Select definition as, pipeline script nd type the code

pipeline {

environment{

username = 'java-tejaswini'

}

agent any

stages {

stage('Build') {

steps {

script{

withCredentials([usernamePassword(credentialsId: 'git\_id', usernameVariable: '', passwordVariable: '')])

{

print 'username=' + username

// Get some code from a GitHub repository

git credentialsId: 'java-tejaswini', url: 'https://github.com/java-tejaswini/Jenkins\_Maven.git'

// Run Maven on a Unix agent.

// sh "mvn -Dmaven.test.failure.ignore=true clean package"

// To run Maven on a Windows agent, use

bat "mvn -f Jenkins\_Maven\_demo/pom.xml -Dmaven.test.failure.ignore=true clean package"

}

}

}

post {

// If Maven was able to run the tests, even if some of the test

// failed, record the test results and archive the jar file.

success {

junit '\*\*/target/surefire-reports/TEST-\*.xml'

// archiveArtifacts 'target/\*.jar'

}

}

}

}

}

Demo2:

pipeline with Git and credentials

[add the plugin](https://www.jenkins.io/doc/pipeline/steps/credentials-binding/) **[Credentials Binding Plugin](https://www.jenkins.io/doc/pipeline/steps/credentials-binding/)**

* Select This project is parameterised

Add Credential parameter = GIT\_CRED

Credential type = user name password

Select the default value

* Add password parameter

Name = GIT\_PASS

Add value for password

pipeline {

environment{

username = 'java-tejaswini'

}

agent any

stages {

stage('Build') {

steps {

script{

withCredentials([usernamePassword(credentialsId: 'git\_id', usernameVariable:'java-tejaswini', passwordVariable: '$GIT\_PASS')])

{

print 'username=' + username

// Get some code from a GitHub repository

git credentialsId: 'java-tejaswini', url: 'https://github.com/java-tejaswini/Jenkins\_Maven.git'

// Run Maven on a Unix agent.

// sh "mvn -Dmaven.test.failure.ignore=true clean package"

// To run Maven on a Windows agent, use

bat "mvn -f Jenkins\_Maven\_demo/pom.xml -Dmaven.test.failure.ignore=true clean package"

}

}

}

post {

// If Maven was able to run the tests, even if some of the test

// failed, record the test results and archive the jar file.

success {

junit '\*\*/target/surefire-reports/TEST-\*.xml'

// archiveArtifacts 'target/\*.jar'

}

}

}

}

}

Pipeline

<https://www.guru99.com/jenkins-pipeline-tutorial.html>

create a new view

select the initial project

select anyother project

goto --🡪Build Triggers --🡪 Build after other projects are built

provide the name of other project

observe the pipline

Use demo 1 for war file upload

Webhook addition

<https://www.blazemeter.com/blog/how-to-integrate-your-github-repository-to-your-jenkins-project>

run ngrok to get public ip

goto git respective repository ---🡪 setting

select webhook

add payload url generated by the ngrok as [gnrokURL/github-webhook/](http://d26d58b4204f.ngrok.io/github-webhook/)

http://d6c47cd6a4d0.ngrok.io/github-webhook/

select the radio button let me select individual events

click on Add webhook

.NET

<https://www.c-sharpcorner.com/article/continuous-integration-for-net-projects-with-jenkins/>

<https://www.c-sharpcorner.com/article/setup-build-of-net-project-using-jenkins/>

<https://www.swtestacademy.com/jenkins-dotnet-integration/>

<https://devops4solutions.medium.com/ci-cd-using-jenkins-and-dotnet-c141cd32d0d3>

<https://www.jenkins.io/doc/>

<https://www.jenkins.io/doc/book/>

Pipeline for Java

<https://levelup.gitconnected.com/jenkins-pipeline-with-gitlab-for-java-projects-d2e10c08e255>

pipeline {  
agent any  
  
stages {  
stage('Build') {  
steps {  
sh 'mvn clean install -DskipTests'  
}  
}  
stage('Test') {  
steps {  
sh 'mvn test'  
}  
}  
}   
}

## Tomcat Deployement

## Refer to ---- https://www.theserverside.com/video/Step-by-step-Jenkins-Tomcat-deploy-of-a-WAR-file

For war Deployment

Crete a new Spring Intializatializer

Select war instead of Jar

Select web as dependency

Run the project as maven install

Find .war in target

Code overage

https://www.bogotobogo.com/DevOps/Jenkins/Jenkins\_Adding\_Code\_Coverage\_and\_Metrics.php

* Goto plugin management and add [Cobertura Reporter](https://istanbul.js.org/docs/advanced/alternative-reporters/#cobertura) plugin
* Now goto project pom.xml and add a new goal as,

##### **Config coverage tool to generate reports.**

Config maven to generate Cobertura coverage reports:

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

<version>2.7</version>

<configuration>

<formats>

<format>xml</format>

</formats>

<check/>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>cobertura</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

* Now configure a new job as Maven Job
* Use the repo url as -🡪 <https://github.com/java-tejaswini/Jenkins_repo_delete.git> (refering to Jenkins\_Maven Project)
* In build section add

Root pom --🡪 pom.xml

Goals and options -🡪 test cobertura:cobertura -Dcobertura.report.format=xml

* In Post build actions

Slecte publish Cobertura Coverage report

Add xml report pattern as -🡪 \*\*/target/site/cobertura/coverage.xml

* Save and build the job you will find the report created

Git connection on local :

Use git clone and open the repo in eclipse

Git fetch

Git branch

Create a branch in github

Git fetch

Git checkout branch1

Change the content

Git add .

Git commit -m “working with”

Git push

Use the pull request in github

Git fetch

Git checkout master