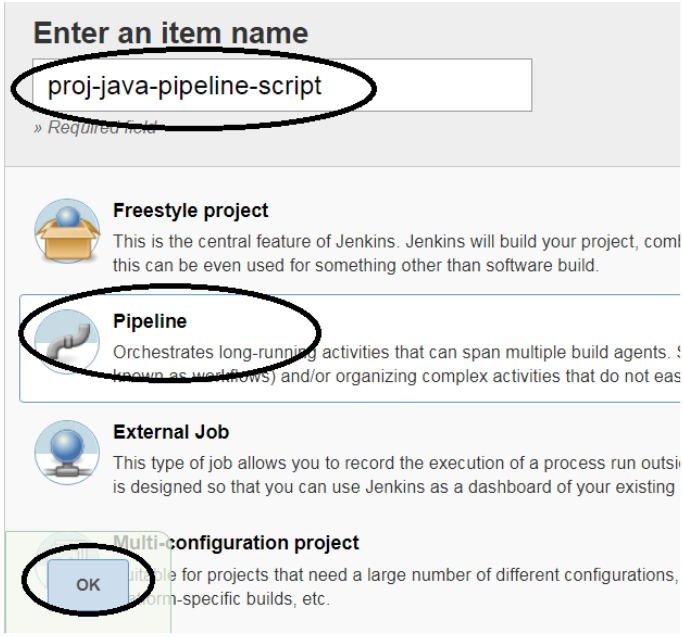
Lab Scenario for Jenkins with GITHUB integration as a pipeline script

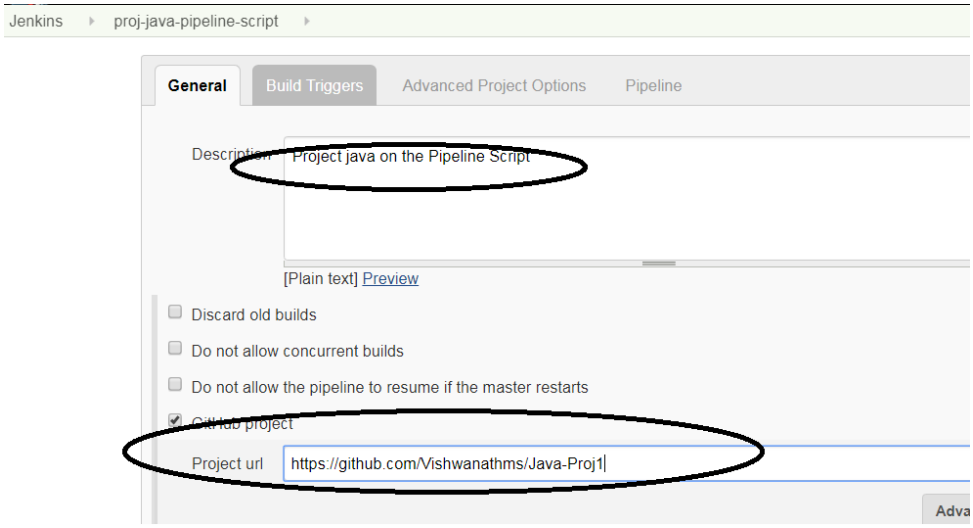
1. **Create the "Java code" repo as below on the github**

2. **Create a Jenkins pipeline script for this java code to run.**

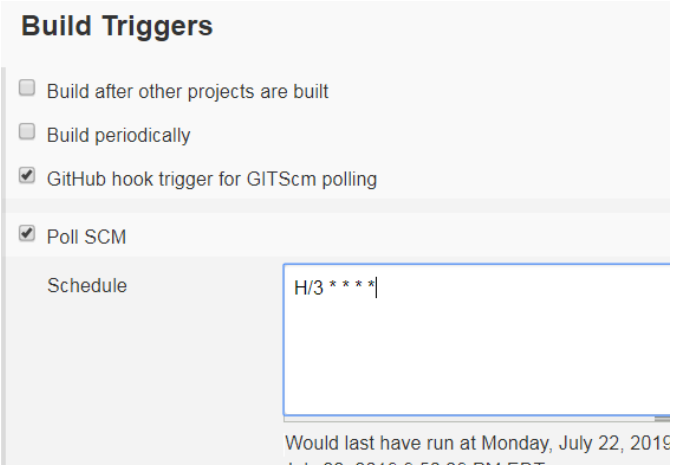


We would need to select "pipeline" instead of "Freestyle project"

And Click "OK"

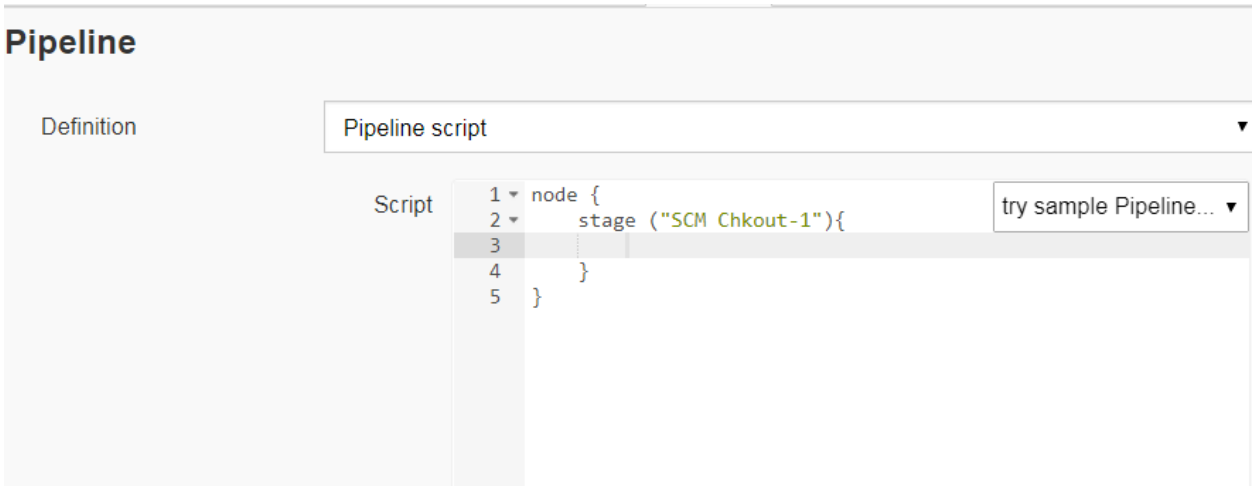


Paste the link of the java project github link for a automatic trigger.



This would trigger the pipeline script, if any changes occur in the Java GITHUB repo.

Now going to the Pipeline script



The below script is the basic skeleton of the pipeline script

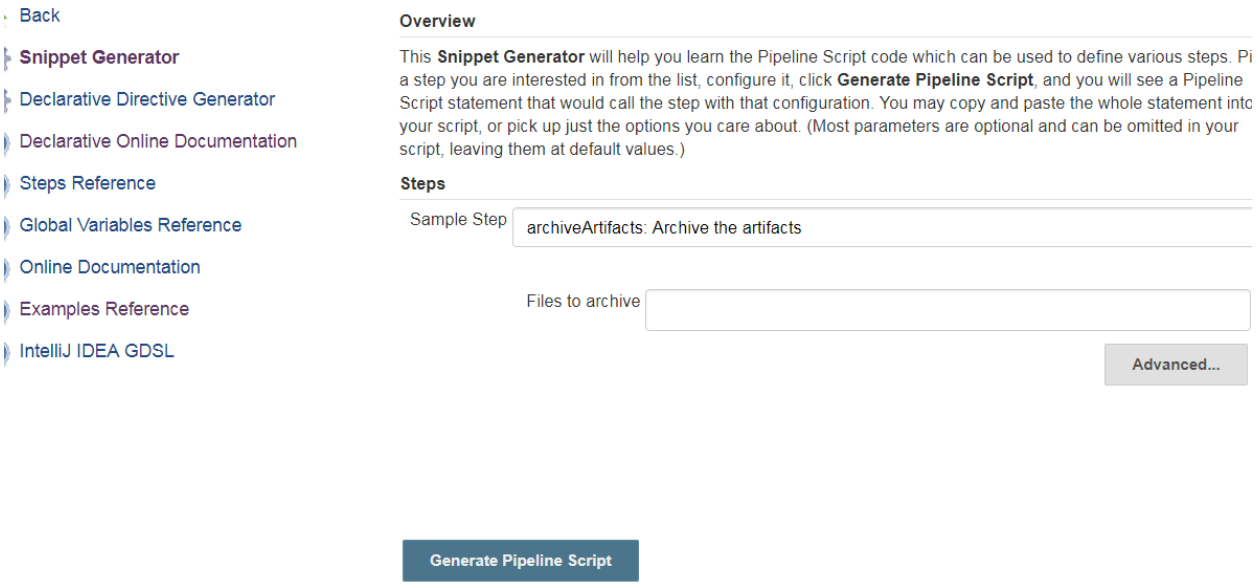
```
node {
    stage ("SCM Chkout-1"){

    }
}
```

Now we would need to add the stages we want to execute.

We can start by clicking on **"pipeline syntax"**

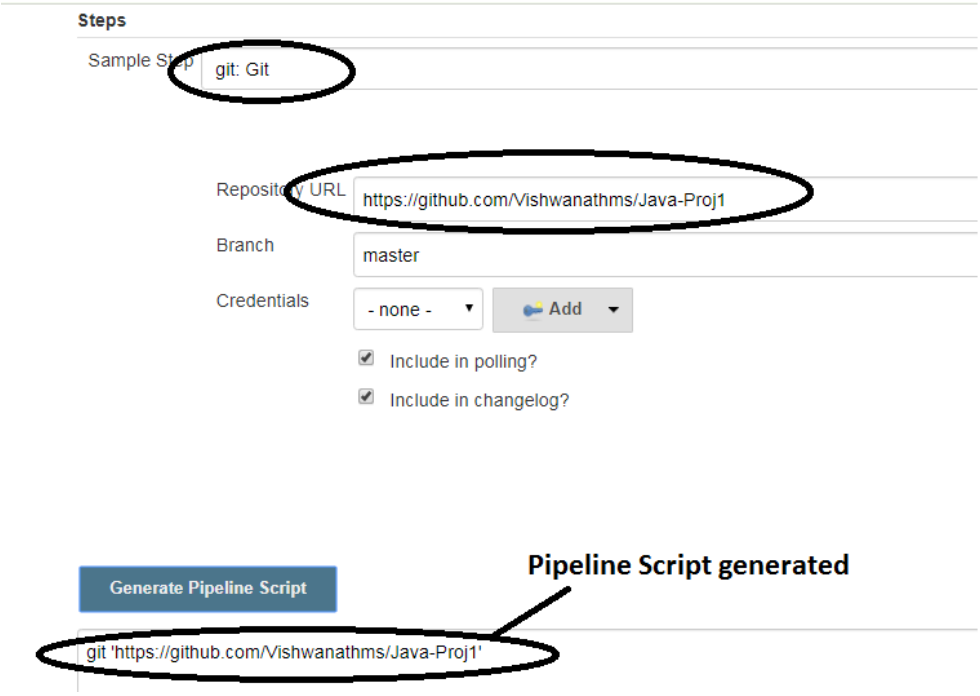
We get the below new page.



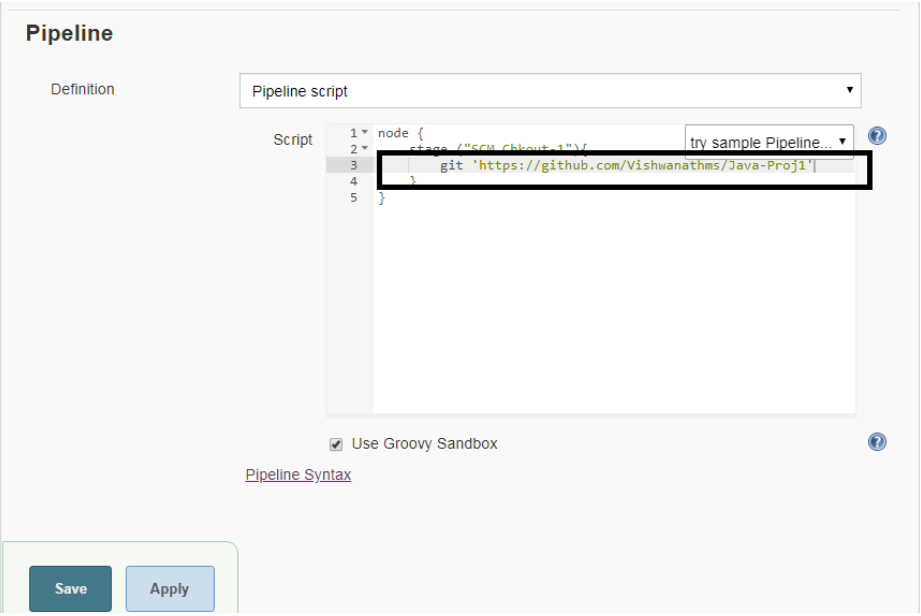
Here we can generate any kind of pipeline scripts and paste it back in the main pipeline script JOB.

For example.

Lets create an GIT pipeline script.



Now paste this script back in the main pipeline job page.



Now lets click "**Save**" and lets build this project to chk if the repo is downloaded to the Jenkins server into the project workspace or not.

Build is triggered manually.

And it show "Success".

```
 > /usr/bin/git checkout -b master 6c6bacac58ad89594d4bd250cae62f1e2ea4db41
Commit message: "Update Samplecode.java"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

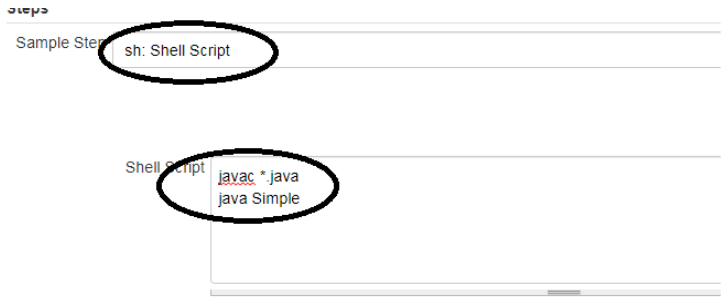
Lets check on the Jenkins server now.

```
[root@localhost workspace]# cd proj-java-pipeline-script/
[root@localhost proj-java-pipeline-script]# pwd
/var/lib/jenkins/workspace/proj-java-pipeline-script
[root@localhost proj-java-pipeline-script]# ls -l
total 4
-rw-r--r--. 1 root root 166 Jul 22 21:58 Samplecode.java
[root@localhost proj-java-pipeline-script]#
```

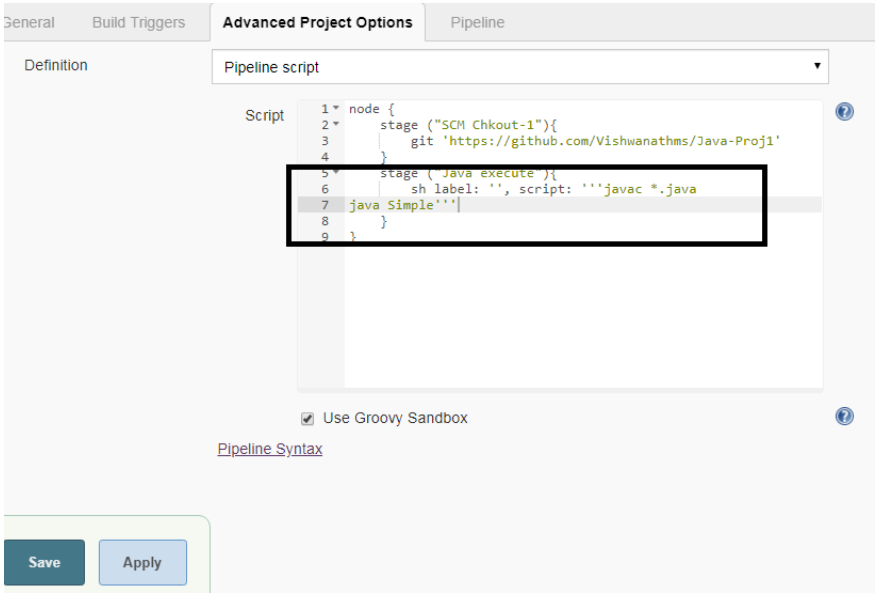
This cleared indicates, the ".java" file was download from the github repo.

Next lets run the java code for the output.

Going back to the pipeline syntax.



Lets paste this generated pipeline script back to the main Script page.



I hope this is understood.

And then click “Save”

Now its time to BUILD the project again.

The output of the build is as below, the java code was run successfully.

```
 > /usr/bin/git fetch --tags --progress https://github.com/Vis
+refs/heads/*:refs/remotes/origin/*
 > /usr/bin/git rev-parse refs/remotes/origin/master^{commit}
 > /usr/bin/git rev-parse refs/remotes/origin/origin/master^{c
Checking out Revision 6c6bacac58ad89594d4bd250cae62f1e2ea4db41
 > /usr/bin/git config core.sparsecheckout # timeout=10
 > /usr/bin/git checkout -f 6c6bacac58ad89594d4bd250cae62f1e2e
 > /usr/bin/git branch -a -v --no-abbrev # timeout=10
 > /usr/bin/git branch -D master # timeout=10
 > /usr/bin/git checkout -b master 6c6bacac58ad89594d4bd250cae
Commit message: "Update Samplecode.java"
 > /usr/bin/git rev-list --no-walk 6c6bacac58ad89594d4bd250cae
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Java execute)
[Pipeline] sh
+ javac Samplecode.java
+ java Simple
Hello Java
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Let's cross verify on the Jenkins server as well.

```
[root@localhost proj-java-pipeline-script]# ls -l
total 8
-rw-r--r--. 1 root root 166 Jul 22 21:58 Samplecode.java
-rw-r--r--. 1 root root 545 Jul 22 22:06 Simple.class
[root@localhost proj-java-pipeline-script]# pwd
/var/lib/jenkins/workspace/proj-java-pipeline-script
[root@localhost proj-java-pipeline-script]#
```

The "Simple" class file is created as the result of the java compilation.

Troubleshooting:

Incase the build fails → saying no agent available and if the cursor is hung and waiting, means there are no Available agent to process the request.

As per the previous lab, we have set the below configuration under , Manage Jenkins → Configuration.

Labels	Master
Usage	Only build jobs with label expressions matching this node

For now, to execute the project, lets change the Usage to "Use this node as much as possible"

3. Create a Jenkins advance pipeline script for the same java code to run.

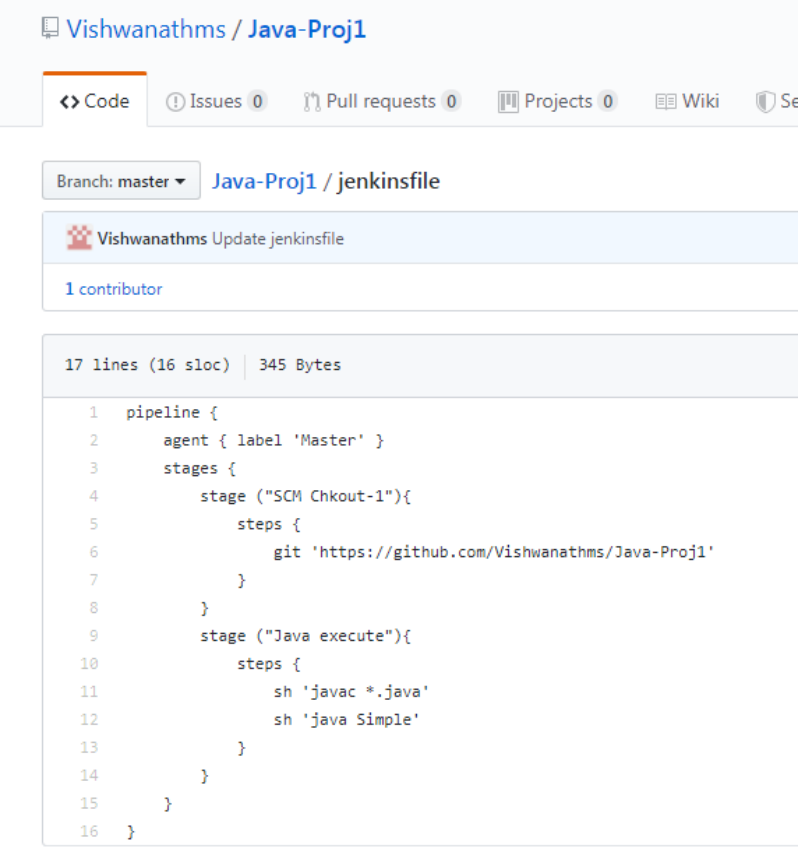
```
pipeline {
    agent { label 'Master' }
    stages {
        stage ("SCM Chkout-1"){
            steps {
                git 'https://github.com/Vishwanathms/Java-Proj1'
            }
        }
        stage ("Java execute"){
            steps {
                sh label: '', script: '''javac *.java java Simple'''
            }
        }
    }
}
```

And then do a build , even this would work fine.

4. Put the Jenkins pipeline in the SCM and manage the changes of the pipeline code.

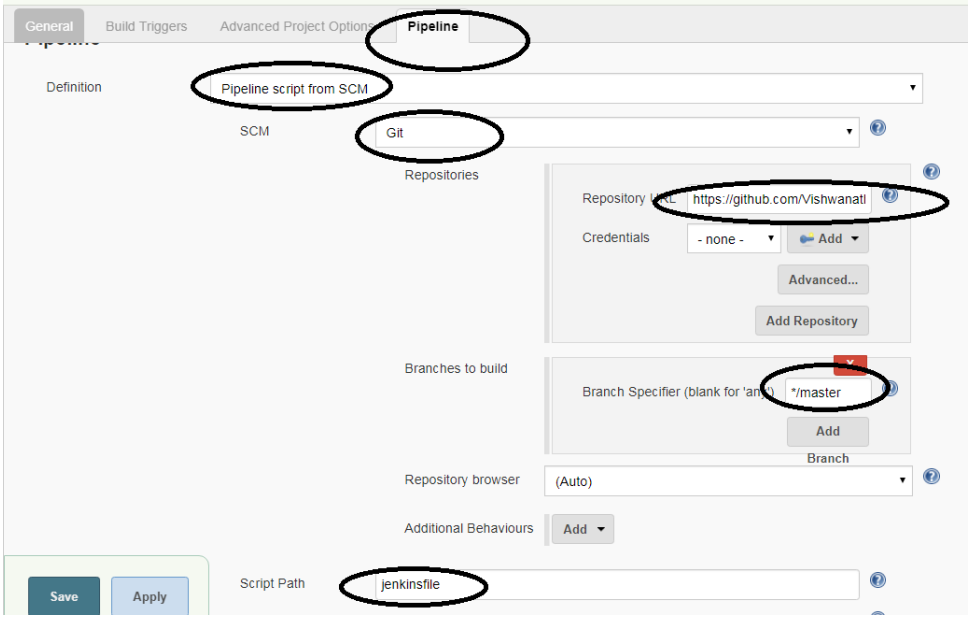
Either you can create a new repository in github for this, or we can use the same repository that has the java code in it.

In our case, we would be using the same repo of java code.



we have create a file in the repo called **"jenkinsfile"**.
With the same pipeline script.

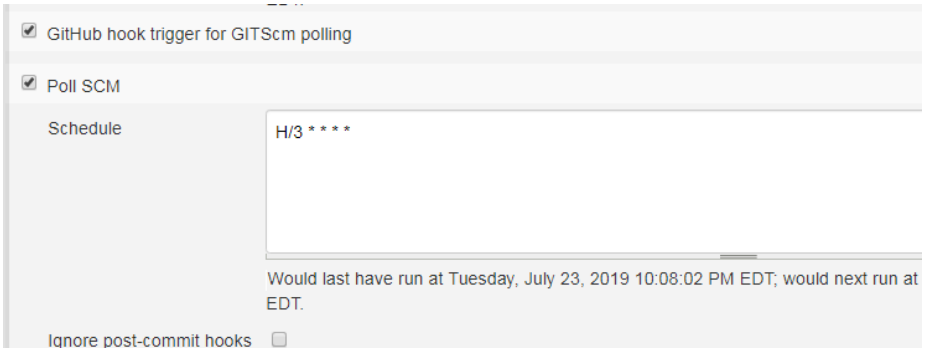
Now lets, change the option the Jenkins job to point it to this jenkinsfile



Note→ the file name under – **script path** should match the name of the file in the repo that we just created.

Then execute the build and check if successful.

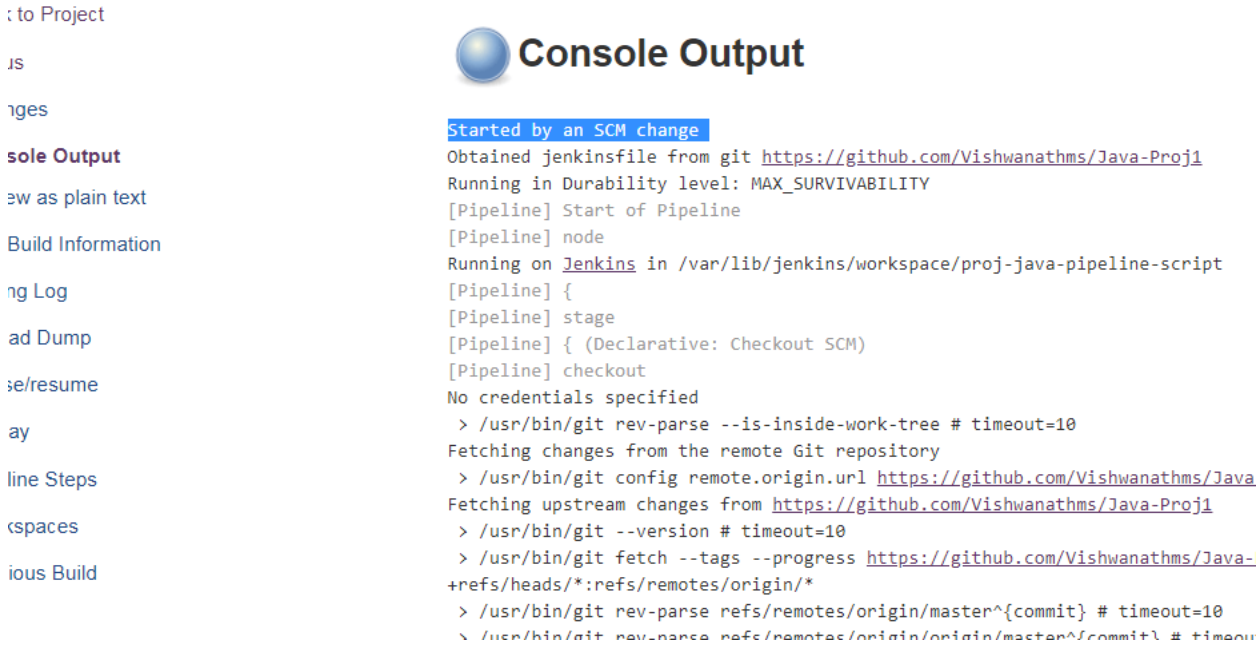
Also, since we have configured, the SCM poll



Lets do some changes to the **"jenkinsfile"** in the repo, and commit it, so that the changes triggers the build automatically.

Note→ which is the exact way that is done in the production/Development env.

The output is below.



Thanks, -- END of this LAB..