

**Practice for Lesson 4:
Integrating Jenkins Pipelines
with Jobs**

Practices for Lesson 4

Overview

In these practices, you will trigger the Jenkins Jobs from one job to another automatically in the Jenkins Dashboard, create the pipeline using Groovy script in Jenkins instance and learn to integrate the GitHub source code to the Jenkins pipeline, and further delegate a pipeline Job using Agent in Jenkins Instance.

Practice 4-3: Integrate GitHub as Source Code on Jenkins

Overview

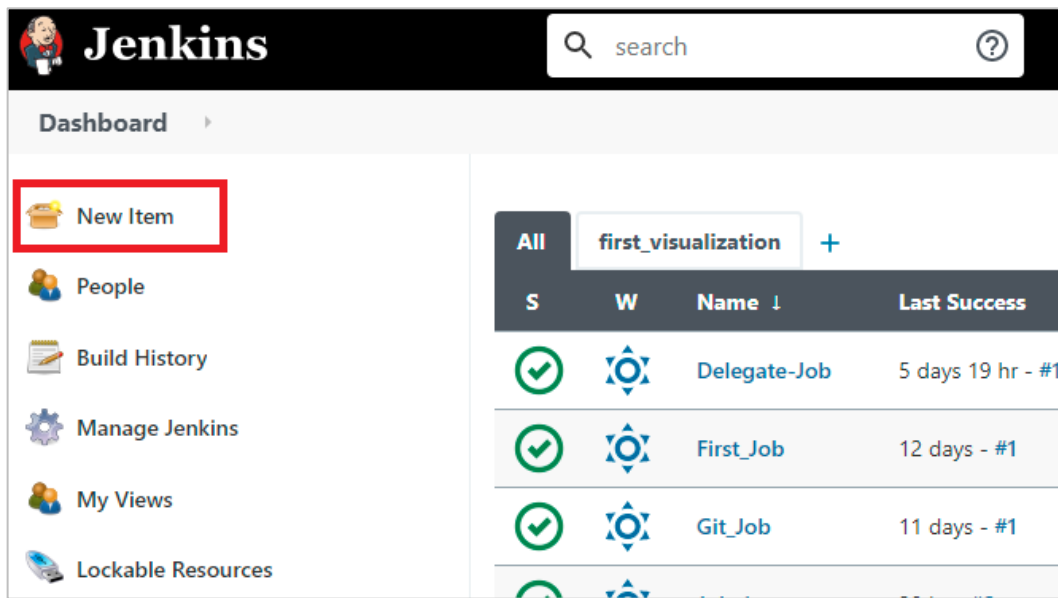
In this practice, you will learn how to integrate the GitHub source code to the Jenkins pipeline and execute the code.

Assumptions

You should have completed the Practice of Lesson 4-2.

Tasks

1. Create a Pipeline in Jenkins instance to integrate with GitHub.
 - a. In the Jenkins Dashboard, navigate to main menu and select **New Item** to create a **Groovy** Scripted Pipeline as shown below.





- b. Provide the **name** for the GitHub Pipeline, further select **Pipeline** and click **OK** as shown below.


Enter an item name


GitHub_Pipeline_Scripted

» Required field


Freestyle project
 This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.


Maven project
 Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.


Pipeline
 Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


External Job
 This job allows you to record the execution of a process run outside Jenkins, even on a remote system. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

OK

- c. Scroll down to **Pipeline** and select **Pipeline script** under **Definition** as shown below.

General Build Triggers **Advanced Project Options** Pipeline

Advanced Project Options

Advanced...

Pipeline

Definition

Pipeline script

Script

1

try sample Pipeline...

- d. Copy the Groovy script provided below and paste it in the **Script** block as shown below. Select the checkbox of **Use Groovy Sandbox**.

```

pipeline {
    agent any
    stages {
        stage('Git-Checkout') {
            steps {

```

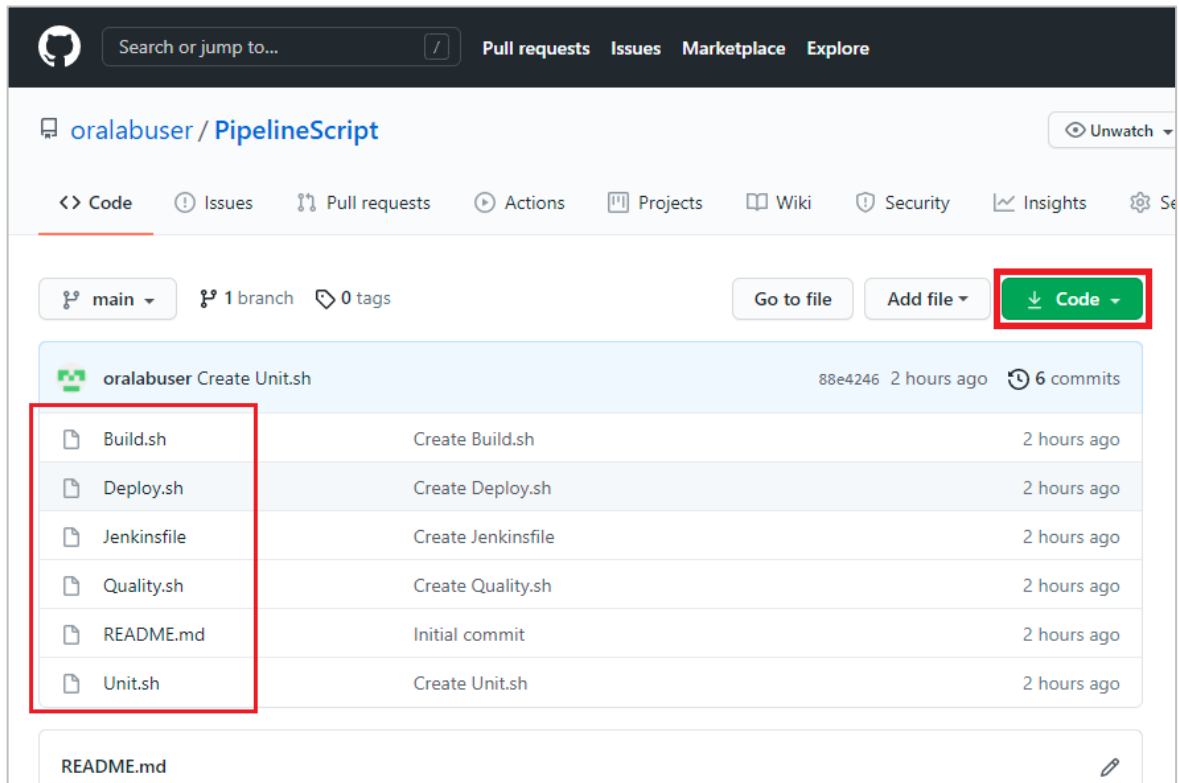
```

        echo "Checking out from Git Repo";
    }
}
stage('Build') {
    steps {
        echo "Building the checked-out project!";
    }
}
stage('Unit-Test') {
    steps {
        echo "Running JUnit Tests";
    }
}
stage('Quality-Gate') {
    steps {
        echo "Verifying Quality Gates";
    }
}
stage('Deploy') {
    steps {
        echo "Deploying to Stage Environment for more
tests!";
    }
}
}
post {
    always {
        echo 'This will always run'
    }
    success {
        echo 'This will run only if successful'
    }
    failure {
        echo 'This will run only if failed'
    }
    unstable {
        echo 'This will run only if the run was marked as
unstable'
    }
    changed {
        echo 'This will run only if the state of the pipeline
has changed'
    }
}

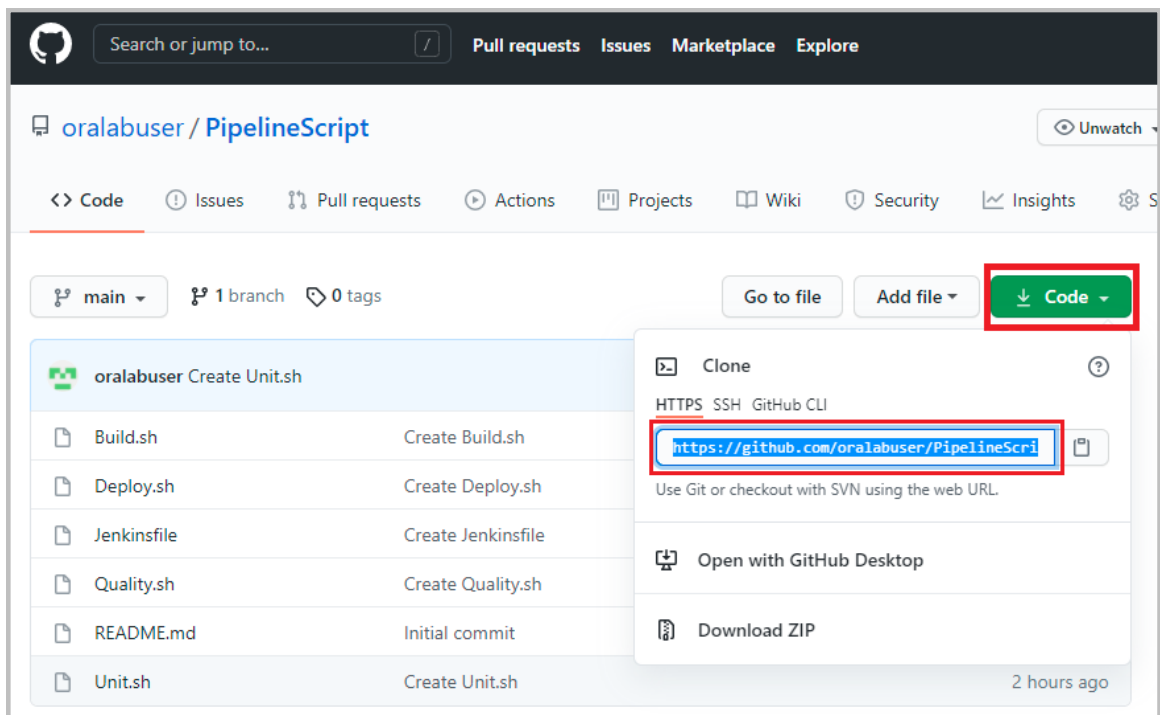
```


- f. The GitHub link is provided below which is in the public domain, consisting of the **.sh** program files to be executed.

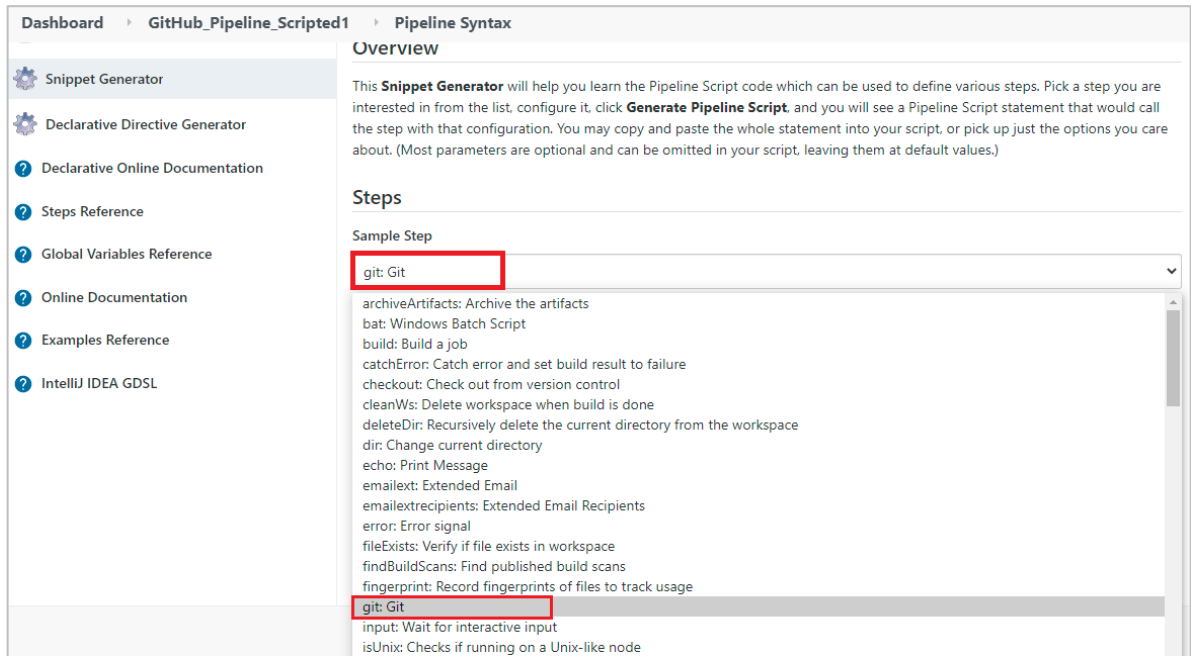
Link: [oralabuser/PipelineScript \(github.com\)](https://github.com/oralabuser/PipelineScript)



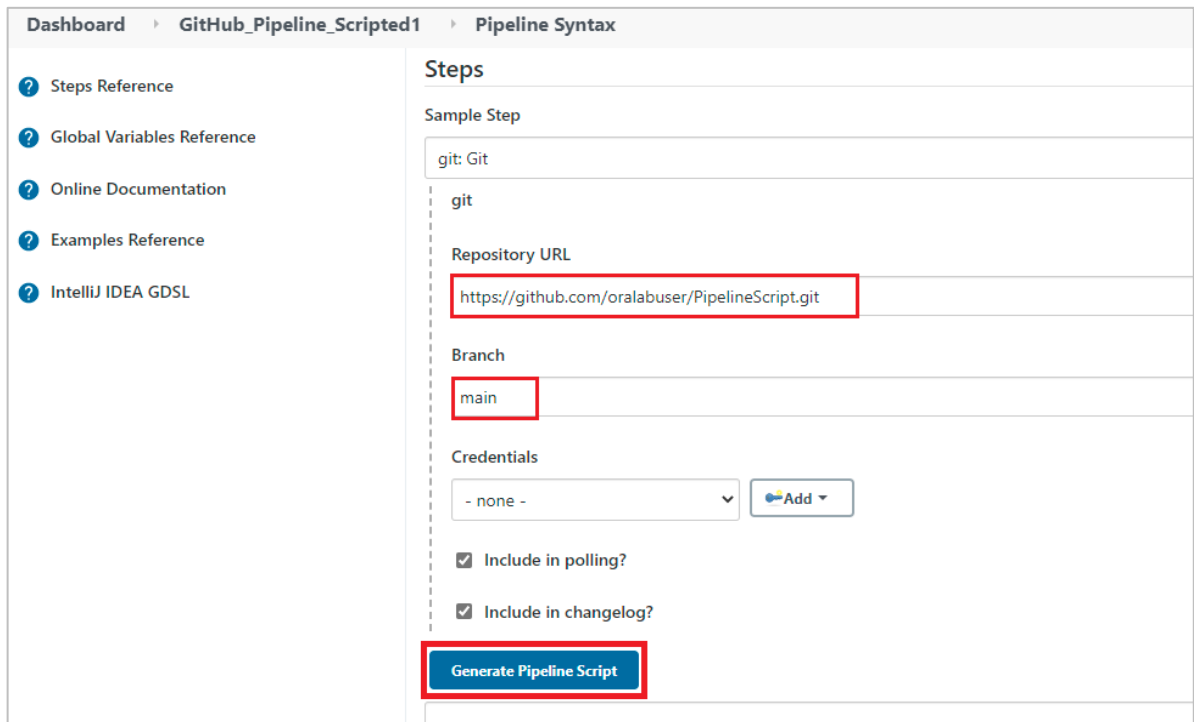
- g. Click on **Code** and copy the **HTTPS** link as shown below to access the shell script files.



- h. In the **Pipeline Syntax Generator** page, navigate to **Sample Step** and select **git: Git** as shown below.



- i. Paste the HTTPS GitHub link copied in the **Repository URL**, provide the **Branch** as **main** and click **Generate Pipeline Script** to generate the GitHub code snippet.



- j. As shown below the GitHub Pipeline Script in Groovy is generated. Copy the script.

☒ Include in polling?

☒ Include in changelog?

[Generate Pipeline Script](#)

git branch: 'main', url: 'https://github.com/oralabuser/PipelineScript.git'

- k. Paste the GitHub script in the code as shown below to integrate with the GitHub account and get all the files in the GitHub repository to the Jenkins workspace.

General Build Triggers Advanced Project Options **Pipeline**

Definition

Pipeline script

Script

```

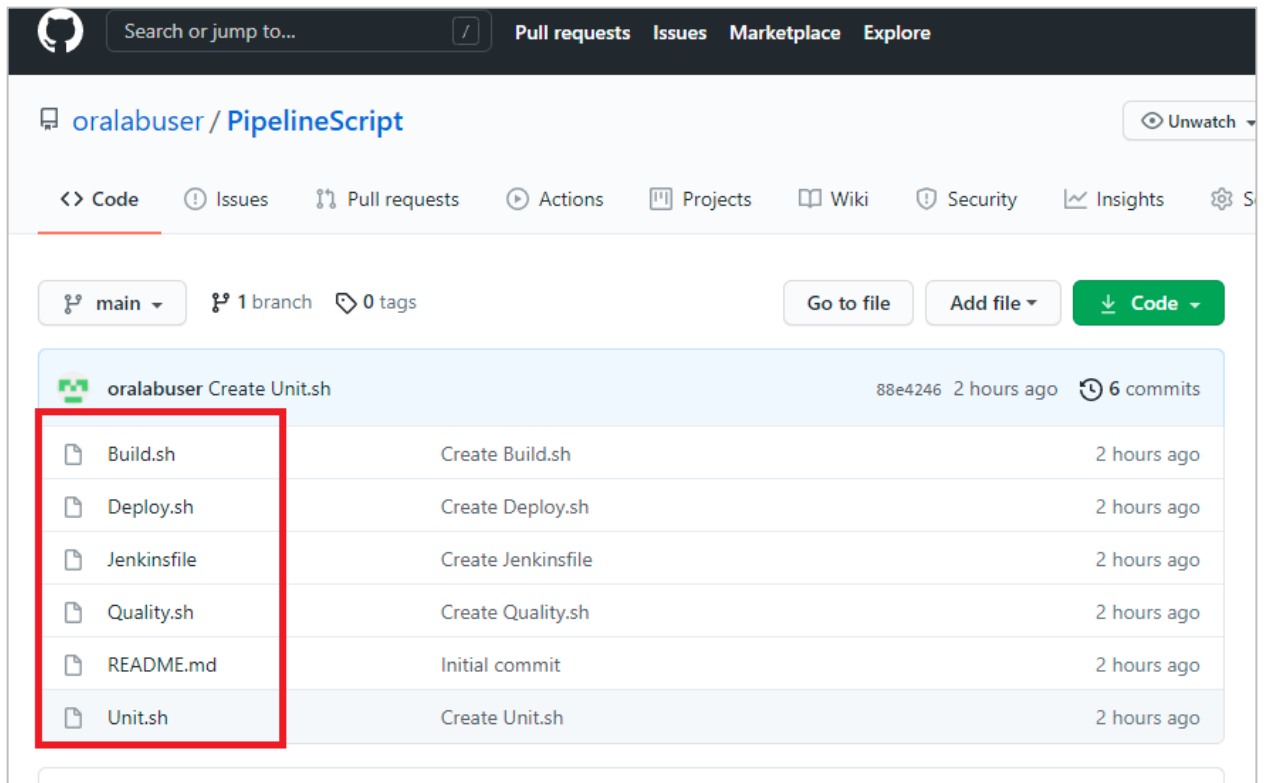
1 pipeline {
2   agent any
3   stages {
4     stage('Git-Checkout') {
5       steps {
6         echo "Checking out from Git Repo";
7         git branch: 'main', url: 'https://github.com/oralabuser/PipelineScript.git'
8       }
9     }
10    stage('Build') {
11      steps {
12        echo "Building the checked-out project!";
13      }
14    }
15    stage('Unit-Test') {
16      steps {
17        echo "Running JUnit Tests";
18      }
19    }
20  }
21 }

```

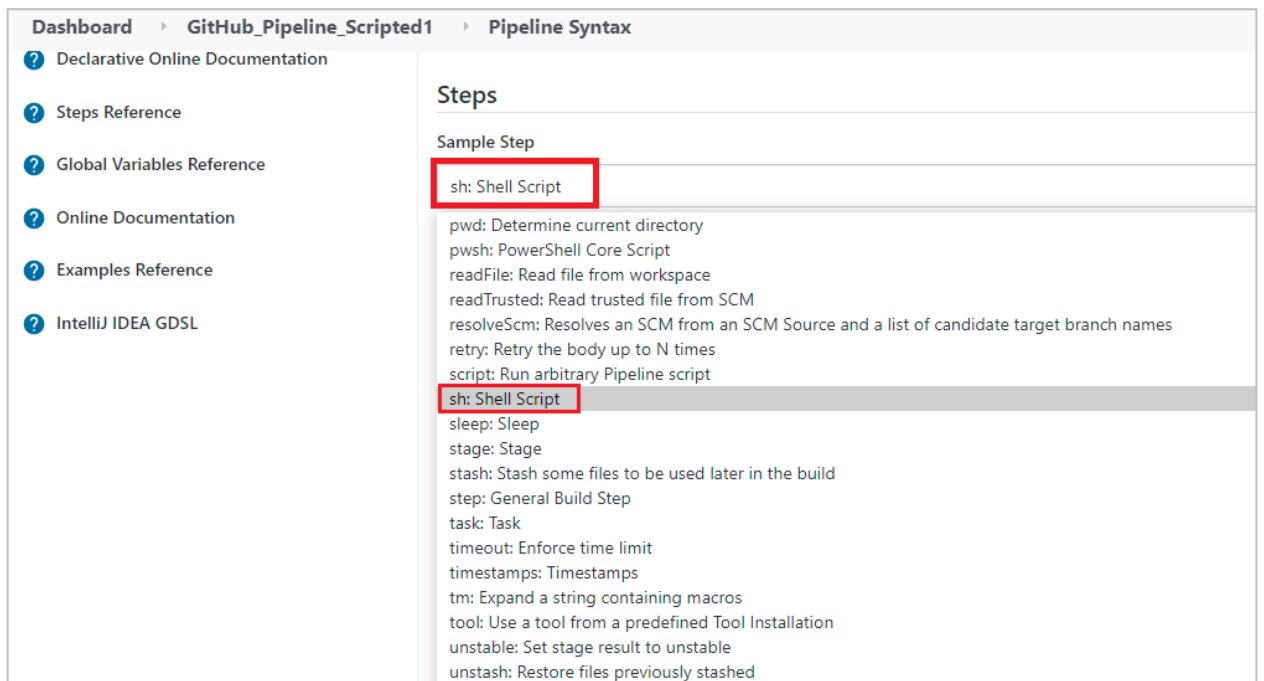
☒ Use Groovy Sandbox

[Pipeline Syntax](#)

- l. As shown below these are the list of files that will be copied to the Jenkins workspace.



- m. In the **Pipeline Syntax Generator** page, navigate to **Sample Step** and select **sh: Shell Script** as shown below.



- n. In the **Shell Script** block, type the command as shown below to execute the **Build.sh** file in the Jenkins Pipeline. Click **Generate Pipeline Script** and copy the **sh** code.

Steps

Sample Step

sh: Shell Script

sh

Shell Script

bash Build.sh

Advanced...

Generate Pipeline Script

sh 'bash Build.sh'

- o. Paste the code in the **Build** stage as shown below. Similarly, provide the code for **Unit.sh**, **Quality.sh** and **Deploy.sh**. Click **Save**.

General Build Triggers Advanced Project Options **Pipeline**

Pipeline script

Script

```

4 stage('Git-Checkout') {
5   steps {
6     echo "Checking out from Git Repo";
7     git branch: 'main', url: 'https://github.com/oralabuser/PipelineScript.git'
8   }
9 }
10 stage('Build') {
11   steps {
12     echo "Building the checked-out project!";
13     sh 'bash Build.sh'
14   }
15 }
16 stage('Unit-Test') {
17   steps {
18     echo "Running JUnit Tests";
19     sh 'bash Unit.sh'
20   }
21 }
22 stage('Quality-Gate') {
23   steps {
24     echo "Verifying Quality Gates";
25     sh 'bash Quality.sh'
26   }
27 }
28 stage('Deploy') {
29   steps {
30     echo "Deploying to Stage Environment for more tests!";
31     sh 'bash Deploy.sh'
32   }
33 }
34 }

```

try sample Pipeline...

Save Apply

- p. On updating the Groovy script, final code looks similar as shown below.

```

pipeline {
    agent any
    stages {
        stage('Git-Checkout') {
            steps {
                echo "Checking out from Git Repo";
                git branch: 'main', url:
'https://github.com/oralabuser/PipelineScript.git'
            }
        }
        stage('Build') {
            steps {
                echo "Building the checked-out project!";
                sh 'bash Build.sh'
            }
        }
        stage('Unit-Test') {
            steps {
                echo "Running JUnit Tests";
                sh 'bash Unit.sh'
            }
        }
        stage('Quality-Gate') {
            steps {
                echo "Verifying Quality Gates";
                sh 'bash Quality.sh'
            }
        }
        stage('Deploy') {
            steps {
                echo "Deploying to Stage Environment for more
tests!";
                sh 'bash Deploy.sh'
            }
        }
    }
    post {
        always {
            echo 'This will always run'
        }
        success {

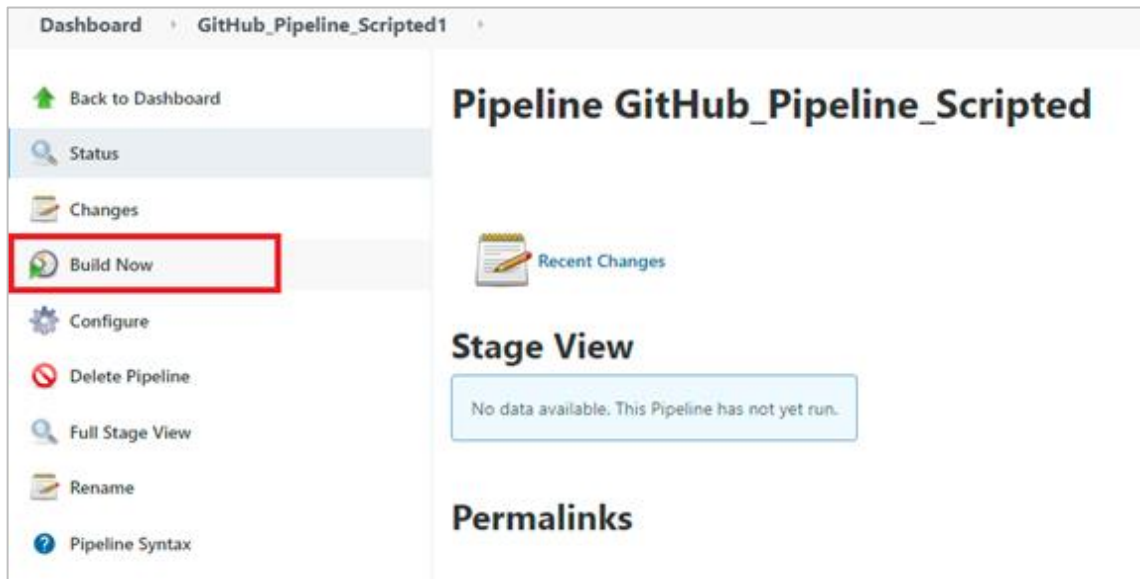
```

```

        echo 'This will run only if successful'
    }
    failure {
        echo 'This will run only if failed'
    }
    unstable {
        echo 'This will run only if the run was marked as
unstable'
    }
    changed {
        echo 'This will run only if the state of the pipeline
has changed'
        echo 'For example, if the Pipeline was previously
failing but is now successful'
    }
}
}

```

2. Build the Groovy Scripted Pipeline created in the Jenkins.
 - a. Select **Build Now** from the menu to execute the Pipeline as shown below.



- b. As shown below, **view** the **stages** of **pipeline** getting executed and click on the link under **Build History**.

Dashboard > GitHub_Pipeline_Scripted >

[Back to Dashboard](#)

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

Build History [trend](#)

find

[Apr 21, 2021 12:52 PM](#)

Atom feed for all Atom feed for failures

Pipeline GitHub_Pipeline_Scripted

[add description](#)

[Disable Project](#)

Recent Changes

Stage View

Average stage times:
(Average full run time: ~2s)

Git-Checkout	Build	Unit-Test	Quality-Gate	Deploy	Declarative: Post Actions
528ms	321ms	302ms	298ms	304ms	63ms

Apr 21 18:23 No Changes

Permalinks

- Last build (#1), 9.5 sec ago
- Last stable build (#1), 9.5 sec ago
- Last successful build (#1), 9.5 sec ago

- c. Click on **Console Output** to view the execution of the scripted pipeline integrating with the GitHub.

Dashboard > GitHub_Pipeline_Scripted > #1

[Back to Project](#)

Status

Changes

Console Output

[View as plain text](#)

[Edit Build Information](#)

Delete build '#1'

Git Build Data

Restart from Stage

Replay

Pipeline Steps

Console Output

Started by user [oralabuser](#)

Running in Durability level: MAX_SURVIVABILITY

[Pipeline] Start of Pipeline

[Pipeline] node

Running on [Jenkins](#) in [/var/lib/jenkins/workspace/GitHub_Pipeline_Scripted](#)

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Git-Checkout)

[Pipeline] echo

Checking out from Git Repo

[Pipeline] git

The recommended git tool is: NONE

No credentials specified

Cloning the remote Git repository

Cloning repository <https://github.com/oralabuser/PipelineScript.git>

> git init /var/lib/jenkins/workspace/GitHub_Pipeline_Scripted1 # timeout=10

Fetching upstream changes from <https://github.com/oralabuser/PipelineScript.git>

> git --version # timeout=10

> git --version # 'git version 2.23.4'

> git fetch --tags --force --progress -- <https://github.com/oralabuser/PipelineScript.git> +refs/heads/*:refs/remotes/origin/* # timeout=10

> git config remote.origin.url <https://github.com/oralabuser/PipelineScript.git> # timeout=10

> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10

- d. As shown below, the scripted pipeline integrating with GitHub is executed successfully.

```

[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying to Stage Environment for more tests!
[Pipeline] sh
+ bash Deploy.sh
Deploying Build : 04/21/21 : 12:53:37
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
This will always run
[Pipeline] echo
This will run only if the state of the pipeline has changed
[Pipeline] echo
For example, if the Pipeline was previously failing but is now successful
[Pipeline] echo
This will run only if successful
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

- e. Connect to Jenkins AWS instance from Putty to verify the path of the workspace consisting of the GitHub repository files as shown below.

```

[ec2-user@ip-172-31-33-131 ~]$ cd /var/lib/jenkins/workspace/GitHub_Pipeline_Scripted
[ec2-user@ip-172-31-33-131 GitHub Pipeline Scripted]$ ls
Build.sh  Deploy.sh  Jenkinsfile  Quality.sh  README.md  Unit.sh
[ec2-user@ip-172-31-33-131 GitHub_Pipeline_Scripted]$

```

3. Keep the Jenkins Dashboard and the AWS Management Console open for the next practice.