

# Jenkins Pipeline

Ostan Dsouza

CodeCraft Technologies

10/06/2020

# Table of Contents

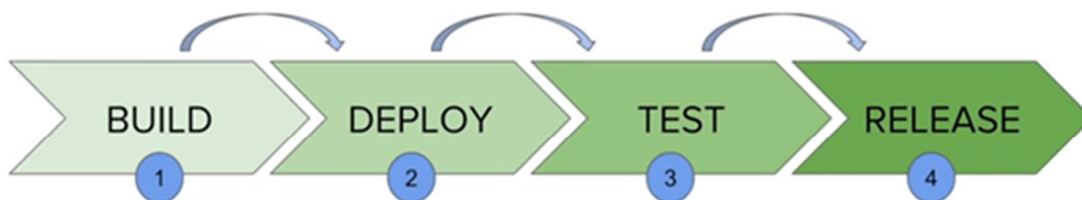
- Chapter 1: **Introduction** ..... 3
  - Section 1.1: **What is Jenkins Pipeline?** ..... 3
  - Section 1.2: **Why Use Jenkins Pipeline?** ..... 4
  
- Chapter 2: **Types of Pipelines** ..... 4
  - Section 2.1: **Scripted** ..... 4
    - Section 2.1.1: **Scripted Pipeline Concepts** ..... 4
  - Section 2.2: **Declarative** ..... 5
    - Section 2.2.1: Declarative Pipeline Concepts ..... 5
  - Section 2.3: **Declarative versus Scripted Pipeline syntax** ..... 6
  
- Chapter 3: ENHO Pipeline..... 7
  - Section 3.1 Pipeline Breakdown ..... 11
    - Section 3.1.1 **Snippet Generator** ..... 12

## Chapter 1: Introduction

In Jenkins, a pipeline is a group of events or jobs which are interlinked with one another in a sequence. In simple words, Jenkins Pipeline is a combination of plugins that support the integration and implementation of **continuous delivery pipelines** using Jenkins. A pipeline has an extensible automation server for creating simple or complex delivery pipelines "as code," via pipeline DSL (Domain-specific Language).

### Section 1.1: What is Jenkins Pipeline?

In a Jenkins Pipeline, every job has some sort of dependency on at least one or more jobs or events.



The above diagram represents a continuous delivery pipeline in Jenkins. It contains a collection of states such as build, deploy, test and release. These jobs or events are interlinked with each other. Every state has its jobs, which work in a sequence called a continuous delivery pipeline. By modelling a series of related tasks, users can take advantage of the many features of Pipeline:

- **Code:** Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable:** Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile:** Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible:** The Pipeline plugin supports custom extensions to its DSL [\[1\]](#) and multiple options for integration with other plugins.

## Section 1.2: Why Use Jenkins Pipeline?

Jenkins is a continuous integration server which has the ability to support the automation of software development processes. You can create several automation jobs with the help of use cases, and run them as a Jenkins pipeline.

Here are the reasons why you should use Jenkins pipeline:

- Jenkins pipeline is implemented as a code which allows several users to edit and execute the pipeline process.
- Pipelines are robust. So, if your server undergoes an unpredicted restart, the pipeline will be automatically resumed.
- You can pause the pipeline process and make it wait to continue until there is an input from the user.
- Jenkins Pipelines support big projects. You can run many jobs, and even use pipelines in a loop.

## Chapter 2: Types of Pipelines

Two types of syntax are used for defining your JenkinsFile.

- Declarative
- Scripted

### Section 2.1: Scripted

Scripted Jenkins pipeline runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to translate the pipeline into atomic commands. Both declarative and scripted syntax are different from each other and are defined totally differently.

#### Section 2.1.1: Scripted Pipeline Concepts

**Node:** The node is a machine on which Jenkins runs is called a node. A node block is used in scripted pipeline syntax.

Eg: node {  
  
}

## Section 2.2: Declarative

Declarative pipeline syntax offers an easy way to create pipelines. It contains a predefined hierarchy to create Jenkins pipelines. It gives you the ability to control all aspects of a pipeline execution in a simple, straight-forward manner. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.

### Section 2.2.1: Declarative Pipeline Concepts

**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. It is used in declarative pipeline syntax.

Eg: pipeline {

}

**Stage:** This block contains a series of steps in a pipeline. i.e., build, test, and deploy processes all come together in a stage. Generally, a stage block visualizes the Jenkins pipeline process.

Let's see an example for multiple stages, where each stage performs a specific task:

```
pipeline {
  agent any
  stages {
    stage ('Build') {
      ...
    }
    stage ('Test') {
      ...
    }
    stage ('QA') {
      ...
    }
    stage ('Deploy') {
      ...
    }
    stage ('Monitor') {
      ...
    }
  }
}
```

**Step:** A step is a single task that executes a specific process at a defined time. A pipeline involves a series of steps defined within a stage block.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Running build phase...'
      }
    }
  }
}
```

### Section 2.3: Declarative versus Scripted Pipeline syntax

Though both these pipelines are based on the groovy DSL, the scripted pipeline uses stricter groovy based syntaxes because it was the first pipeline to be built on the groovy foundation. Since this Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax. The declarative pipeline is defined within a block labelled '**pipeline**' whereas the scripted pipeline is defined within a '**node**'.

#### *Jenkinsfile (Scripted Pipeline)*

```
node {
  stage('Build') {
    //
  }
  stage('Test') {
    //
  }
  stage('Deploy') {
    //
  }
}
```

#### *Jenkinsfile (Declarative Pipeline)*

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        //
      }
    }
    stage('Test') {
      steps {
        //
      }
    }
    stage('Deploy') {
      steps {
        //
      }
    }
  }
}
```

## Chapter 3: ENHO Pipeline

Here is a sample of a Jenkinsfile using Declarative Pipeline syntax

```
pipeline {
    agent {
        label 'Mobile'
    }
    stages {
        stage('build_generate') {
            steps {
                build job: 'HomeOwner_Android_ProdBuild'
            }
        }
        stage('clone_repo') {
            steps {
                script{
                    env.FAILURE_STAGE = 'clone_repo'
                }
                checkout([$class: 'GitSCM', branches: [[name: '*/Codecraft_Android']],
doGenerateSubmoduleConfigurations: false, extensions: [[$class:
'CleanBeforeCheckout']], submoduleCfg: [], userRemoteConfigs:
[[credentialsId: 'rspecs-keys', url:
'git@bitbucket.org:enphaseembedded/mobile_automation.git']]])
            }
        }
        stage('upload_apk') {
            steps {
                script{
                    env.FAILURE_STAGE = 'upload_apk'
                }
                browserstack('9fcaabab-26a6-46c4-86e1-b6bbde22bbef') {}
                browserstackAppUploader(appPath: '/Users/admin/Downloads/latest-
debug.apk'){
```

```

    script {
        env.BROWSERSTACK_APP_ID="${BROWSERSTACK_APP_ID}"
    }
}

stage('test') {
    steps {
        sh 'export PATH="$PATH:/Users/jenkins/apache-maven-3.6.1/bin"'

        script{
            env.FAILURE_STAGE = 'test'
        }

        sh 'printenv'

        sh 'mvn clean test'
        Dsurefire.suiteXmlFiles=src/test/java/com/enphase/${type}.xml'

        sh 'ruby "$WORKSPACE/HTMLReporter/main.rb" "$WORKSPACE/allure-
results" $JOB_BASE_NAME "https://global-
allure.herokuapp.com/$BUILD_NUMBER/allure/'

    }
}

stage('reports') {
    steps {
        script {
            env.FAILURE_STAGE = 'reports'

            allure([
                includeProperties: false,
                jdk: "",
                properties: [],
                reportBuildPolicy: 'ALWAYS',
                results: [[path: 'target/allure-results']]
            ])
        }
    }
}

```



```
}
```

```
stage ('global-allure') {  
    when {  
        expression {  
            return currentBuild.result == 'FAILURE' || currentBuild.result ==  
                'SUCCESS' || currentBuild.result == 'UNSTABLE';  
        }  
    }  
    steps {  
        script{  
            env.FAILURE_STAGE = 'global-allure'  
        }  
        build job: 'AllureReport', parameters: [  
            string(name: 'Temp', value: "${BUILD_NUMBER}")  
        ]  
    }  
}
```

```
post{  
  
    always{  
        junit testDataPublishers: [[${class: 'AutomateTestDataPublisher'}]], testResults:  
            'target/surefire-reports/TEST-*.xml'  
    }  
}
```

```

success{
    echo "Success Pipeline: ${currentBuild.fullDisplayName}"
    script {
        def mailRecipients = 'ostan@codecraft.co.in'
        def jobName = currentBuild.fullDisplayName
        email body: "${FILE,path='HTMLReporter/index.html'}",
        mimeType: 'text/html',
        subject: "[Jenkins][${currentBuild.result}] Test Automation Report",
        to: "${mailRecipients}",
        replyTo: "${mailRecipients}",
        recipientProviders: [[class: 'CulpritsRecipientProvider']]
    }
}

failure {
    echo "Failure Pipeline: ${currentBuild.result}"
    echo "Attention @here ${env.JOB_NAME} #${env.BUILD_NUMBER} has failed."
    script {
        def mailRecipients = 'ostan@codecraft.co.in'
        def jobName = currentBuild.fullDisplayName
        email body: "${FILE,path='HTMLReporter/index.html'}",
        mimeType: 'text/html',
        subject: "[Jenkins][${currentBuild.result}] Build is failing in ${env.FAILURE_STAGE} stage",
        to: "${mailRecipients}",
        replyTo: "${mailRecipients}",
        recipientProviders: [[class: 'CulpritsRecipientProvider']]
    }
}
}

```

## Section 3.1 Pipeline Breakdown

**pipeline** is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.

```
pipeline {  
  
}
```

**agent** is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline. In below example execution will continue on slave 'Mobile'.

```
agent {  
  
    label: 'Mobile'  
}
```

**stage** is a syntax block that describes a stage of this Pipeline. Read more about stage blocks in Declarative Pipeline syntax on the Pipeline\_syntax page. As mentioned above, stage blocks are optional in Scripted Pipeline syntax.

```
stages {  
    stage {  
    }  
}
```

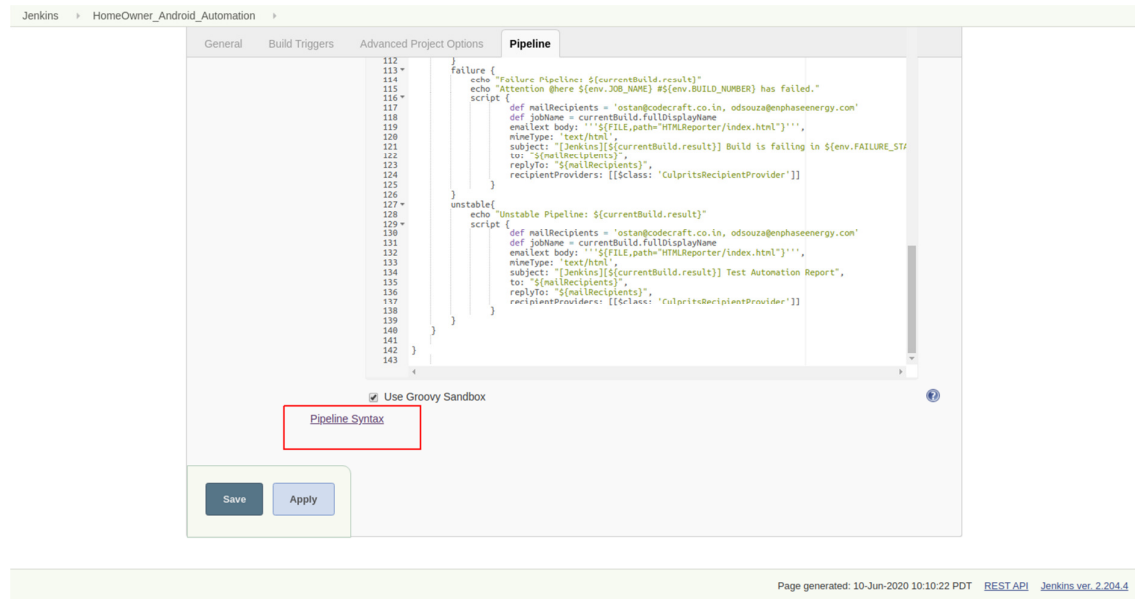
**steps** is Declarative Pipeline-specific syntax that describes the steps to be run in this stage.

```
steps {  
  
}
```

**sh** is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.

## Section 3.1.1 Snippet Generator

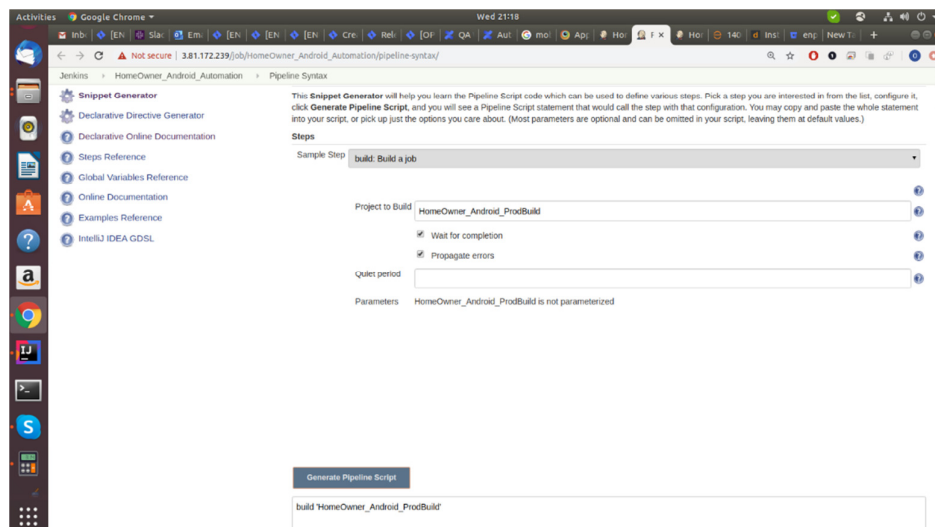
Jenkins provides its own snippet generator for declarative type pipelines



### Build Job:

For build job stage we have to start dependant build job, we can use snippet generator to achieve this.

```
stage('build_generate') {
    steps {
        build job: 'HomeOwner_Android_ProdBuild'
    }
}
```



## Clone repo:

For clone repo stage we have to use git plugin, we can use snippet generator to achieve this.

```
stage('clone_repo') {  
    steps {  
        checkout([$class: 'GitSCM', branches: [[name: '*/Codecraft_Android']],  
doGenerateSubmoduleConfigurations: false, extensions: [[$class:  
'CleanBeforeCheckout']], submoduleCfg: [], userRemoteConfigs:  
[[credentialsId: 'rspecs-keys', url:  
'git@bitbucket.org:enphaseembedded/mobile_automation.git']]])  
    }  
}
```

The screenshot displays the Jenkins Pipeline Snippet Generator interface. On the left, a sidebar contains links to various documentation and reference pages. The main area is titled 'Pipeline Syntax' and features a 'Sample Step' section with a 'checkout: Check out from version control' step. Below this, the configuration for the 'Git' SCM is shown. The 'Repository URL' is set to 'git@bitbucket.org:enphaseembedded/mobile\_automation.git', and the 'Credentials' are set to 'jenkins/\*'. The 'Branches to build' section shows a 'Branch Specifier (blank for \'any\')' of '\*/Codecraft\_Android'. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section includes checkboxes for 'Include in polling?' and 'Include in changelog?'. At the bottom, a 'Generate Pipeline Script' button is visible, followed by the generated Pipeline Script code.

Jenkins > HomeOwner\_Android\_Automation > Pipeline Syntax

**Snippet Generator**

Declarative Directive Generator

Declarative Online Documentation

Steps Reference

Global Variables Reference

Online Documentation

Examples Reference

IntelliJ IDEA GDSDL

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or for the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

**Steps**

Sample Step: checkout: Check out from version control

SCM: Git

Repositories

Repository URL: git@bitbucket.org:enphaseembedded/mobile\_automation.git

Credentials: jenkins/\*

Add Repository

Branches to build

Branch Specifier (blank for 'any'): \*/Codecraft\_Android

Add Branch

Delete Branch

Repository browser: (Auto)

Additional Behaviours

Add

☒ Include in polling?

☒ Include in changelog?

Advanced...

Generate Pipeline Script

```
checkout([$class: 'GitSCM', branches: [[name: '*/Codecraft_Android']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '675e3ece-8a3f-4ec6-b4ad-6a42240e1f6f', url: 'git@bitbucket.org:enphaseembedded/mobile_automation.git']]])
```

## Browserstack upload:

For upload apk stage we have to use browserstack plugin, we can use snippet generator to achieve this.

```
steps {  
    browserstack('9fcaabab-26a6-46c4-86e1-b6bbde22bbef') {}  
    browserstackAppUploader(appPath: '/Users/admin/Downloads/latest-debug.apk'){  
  
    script {  
        env.BROWSERSTACK_APP_ID="${BROWSERSTACK_APP_ID}"  
    }  
}  
}
```

Jenkins > HomeOwner\_Android\_Automation > Pipeline Syntax

[Back](#)  
[Snippet Generator](#)  
[Declarative Directive Generator](#)  
[Declarative Online Documentation](#)  
[Steps Reference](#)  
[Global Variables Reference](#)  
[Online Documentation](#)  
[Examples Reference](#)  
[IntelliJ IDEA GDSL](#)

### Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

### Steps

Sample Step: browserstack: BrowserStack

BrowserStack Credentials: shashikiran7\*\*\*\*\* [Add](#)

☐ BrowserStack Local

Note: Skip this if you are using BrowserStack Local bindings in your code.

[Generate Pipeline Script](#)

```
browserstack('9fcaabab-26a6-46c4-86e1-b6bbde22bbef') {  
    // some block  
}
```

Jenkins > HomeOwner\_Android\_Automation > Pipeline Syntax

[Back](#)  
[Snippet Generator](#)  
[Declarative Directive Generator](#)  
[Declarative Online Documentation](#)  
[Steps Reference](#)  
[Global Variables Reference](#)  
[Online Documentation](#)  
[Examples Reference](#)  
[IntelliJ IDEA GDSL](#)

### Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

### Steps

Sample Step: browserstackAppUploader: BrowserStack App Uploader

App path: /Users/admin/Downloads/latest-debug.apk

File not found : /Users/admin/Downloads/latest-debug.apk

[Generate Pipeline Script](#)

```
browserstackAppUploader('/Users/admin/Downloads/latest-debug.apk') {  
    // some block  
}
```

## Publish Reports:

For reports stage we have to use allure plugin, we can use snippet generator to achieve this.

```
allure([
    includeProperties: false,
    jdk: "",
    properties: [],
    reportBuildPolicy: 'ALWAYS',
    results: [[path: 'target/allure-results']]
])
```

Back

Snippet Generator

Declarative Directive Generator

Declarative Online Documentation

Steps Reference

Global Variables Reference

Online Documentation

Examples Reference

IntelliJ IDEA GDSL

Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step

allure: Allure Report

Disabled

Results:

Path

target/allure-results

Delete

Add

Paths to Allure results directories relative from workspace.  
E.g. target/allure-results.

Properties

Add

JDK

InheritFromJob

JDK to be used for the report building. **jdk** or above.

Generate:

For all builds

For all unstable builds

For unsuccessful builds

Include build environment

Report path:

allure-report

Paths to Allure report directory relative from workspace.  
E.g. allure-report.

Allure Configuration (config.yml):

Path to config.yml which should be used to generate report

JDK

InheritFromJob

JDK to be used for the report building. **jdk** or above.

Generate:

For all builds

For all unstable builds

For unsuccessful builds

Include build environment

Report path:

allure-report

Paths to Allure report directory relative from workspace.  
E.g. allure-report.

Allure Configuration (config.yml):

Path to config.yml which should be used to generate report

Generate Pipeline Script

allure includeProperties: false, jdk: "", properties: [[key: "", value: ""]], results: [[path: 'target/allure-results']]

## Global allure:

For global-allure stage we have to start dependant parameterized allure job, we can use snippet generator to achieve this.

```
build job: 'AllureReport', parameters: [  
    string(name: 'Temp', value: "${BUILD_NUMBER}")
```

The screenshot shows the Jenkins Pipeline Syntax generator interface. The left sidebar contains links: Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSDL. The main area is titled 'Steps' and shows a 'Sample Step' dropdown set to 'build: Build a job'. Below this, the 'Project to Build' is set to 'AllureReport'. There are two checked checkboxes: 'Wait for completion' and 'Propagate errors'. The 'Quiet period' is set to an empty field. The 'Parameters' section shows 'Temp' with the value '{BUILD\_NUMBER}'. At the bottom, a 'Generate Pipeline Script' button is present. Below the button, the generated script snippet is displayed: `build job: 'AllureReport', parameters: [string(name: 'Temp', value: '{BUILD_NUMBER}')]`.

## HTML Reports:

For post build always stage we have to use Publish JUnit HTML report plugin, we can use snippet generator to achieve this.

```
junit testDataPublishers: [[${class: 'AutomateTestDataPublisher'}]], testResults: 'target/surefire-reports/TEST-*.xml'
```

The screenshot shows the Jenkins Pipeline Syntax generator interface for the 'junit testDataPublishers' step. The left sidebar contains links: Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSDL. The main area shows the 'Test report XMLs' field set to 'target/surefire-reports/TEST-\*.xml'. Below this, there is a link to 'Fileset Inclusion' and a checkbox for 'Retain long standard output/error'. The 'Health report amplification factor' is set to '1.0', with a note: '1% failing tests scores as 99% health. 5% failing tests scores as 95% health'. Under 'Additional test report features', the 'Embed BrowserStack Report' checkbox is selected. There is an 'Add' button and a 'Delete' button. At the bottom, a 'Generate Pipeline Script' button is present. Below the button, the generated script snippet is displayed: `junit testDataPublishers: [[${class: 'AutomateTestDataPublisher'}]], testResults: 'target/surefire-reports/TEST-*.xml'`.



## Mail Notification:

For post build success/failure stage we have to use extended email plugin, we can use snippet generator to achieve this.

```
emailext body: "${FILE,path='HTMLReporter/index.html'}",
mimeType: 'text/html',
subject: "[Jenkins][${currentBuild.result}] Build is failing in
${env.FAILURE_STAGE} stage",
to: "${mailRecipients}",
replyTo: "${mailRecipients}",
recipientProviders: [[${class: 'CulpritsRecipientProvider'}]]
```

The screenshot displays the Jenkins Pipeline Syntax editor for a pipeline named 'HomeOwner\_Android\_Automation'. The 'Steps' section is expanded, showing a 'Sample Step' of type 'emailext: Extended Email'. The configuration fields are as follows:

- To:** ostan@codecraft.co.in
- Recipient Providers:** A list containing 'Culprits' with a 'Delete' button.
- Subject:** [Jenkins][\${currentBuild.result}] Test Automation Report
- Body:** \${FILE,path='HTMLReporter/index.html'}
- Reply-To:** ostan@codecraft.co.in
- Body MIME Type:** text/html
- Attachments Pattern:** (empty)
- Attach Build Log:** ☐
- Compress Build Log:** ☐
- Pre-send Script:** (empty text area)
- Post-send Script:** (empty text area)

At the bottom, there is a 'Generate Pipeline Script' button and a preview of the generated pipeline script:

```
emailext body: "${FILE,path='HTMLReporter/index.html'}", mimeType: 'text/html', recipientProviders: [culprits()], replyTo: 'ostan@codecraft.co.in', subject: '[Jenkins][${currentBuild.result}] Test Automation Report', to: 'ostan@codecraft.co.in'
```