# Jenkins Shared Library

As Pipeline is adopted for more and more projects, common patterns are likely to emerge. Oftentimes it is useful to share parts of Pipelines between various projects to reduce redundancies and keep code "DRY".

Pipeline has support for creating "Shared Libraries" which can be defined in external source control repositories and loaded into existing Pipelines.

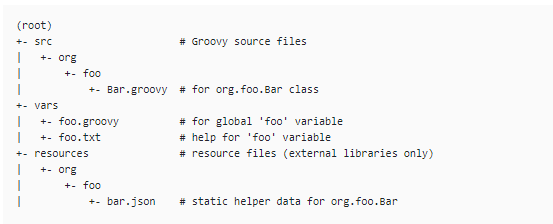
## Shared Library

A Shared Library is defined with a name, a source code retrieval method such as by SCM, and optionally a default version. The name should be a short identifier, as it will be used in scripts. The version could be anything understood by that SCM; for example, branches, tags, and commit hashes all work for Git. You may also declare whether scripts need to explicitly request that library, or if it is present by default. Furthermore, if you specify a version in Jenkins configuration, you can block scripts from selecting a different version.

The best way to specify the SCM is using an SCM plugin, which has been specifically updated to support a new API for checking out an arbitrary named version (Modern SCM option).

## Directory structure

The directory structure of a Shared Library repository is as follows:



The ‘src’ directory should look like standard Java source directory structure. This directory is added to the classpath when executing Pipelines.

The vars directory hosts script files that are exposed as a variable in Pipelines. The name of the file is the name of the variable in the Pipeline. So if you had a file called vars/log.groovy with a function like def info (message)…​in it, you can access this function like log.info "hello world" in the Pipeline. You can put as many functions as you like inside this file.

The basename of each .groovy file should be a Groovy (~ Java) identifier, conventionally camelCased. The matching .txt, if present, can contain documentation, processed through the system’s configured markup formatter (so may really be HTML, Markdown, etc., though the .txt extension is required.

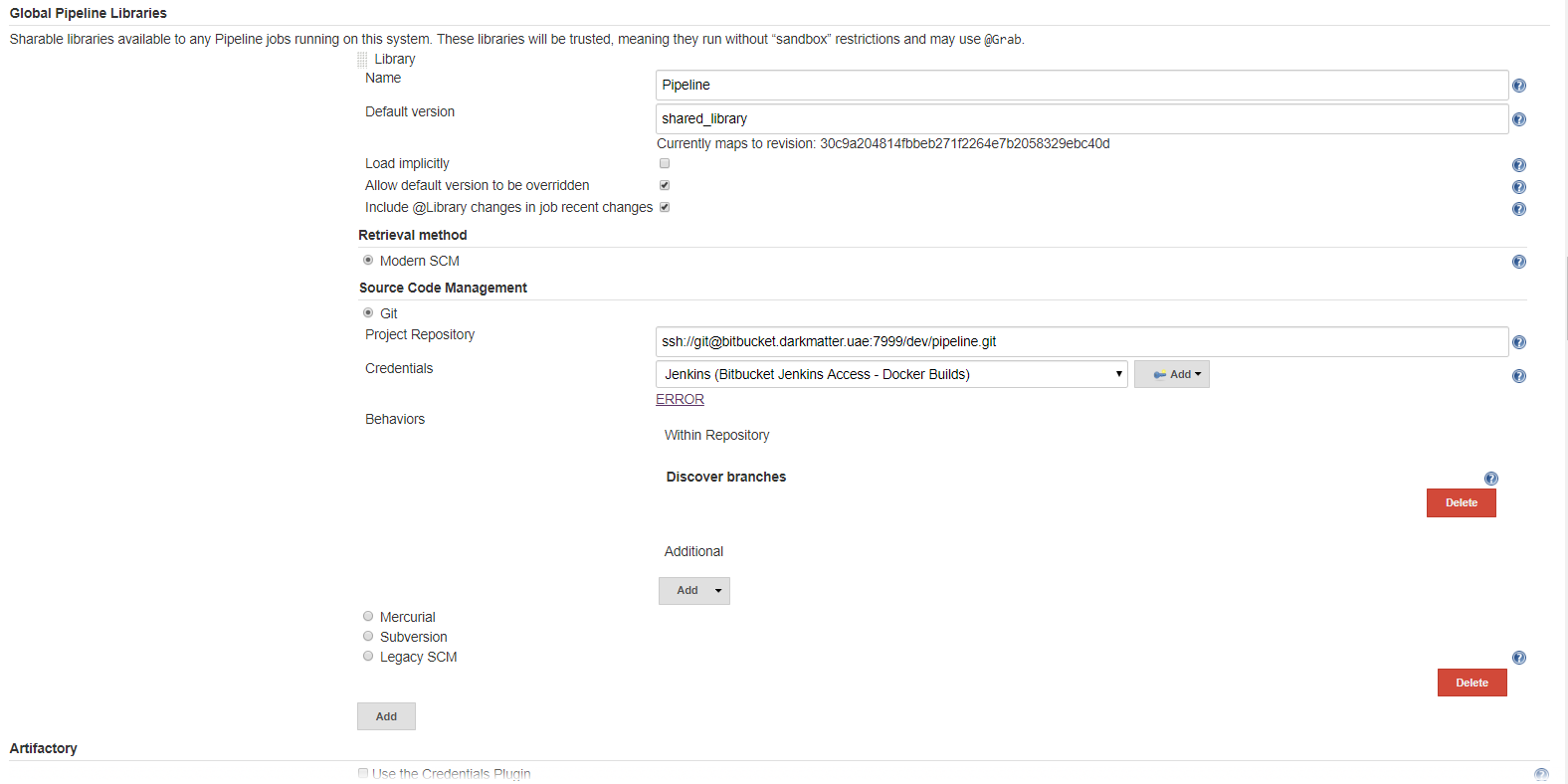
The Groovy source files in these directories get the same “CPS transformation” as in Scripted Pipeline.

A resources directory allows the library Resource step to be used from an external library to load associated non-Groovy files. Currently this feature is not supported for internal libraries.

Other directories under the root are reserved for future enhancements.

## Global Shared Libraries

There are several places where Shared Libraries can be defined, depending on the use-case. Manage Jenkins » Configure System » Global Pipeline Libraries as many libraries as necessary can be configured.



Since these libraries will be globally usable, any Pipeline in the system can utilize functionality implemented in these libraries.

These libraries are considered "trusted:" they can run any methods in Java, Groovy, Jenkins internal APIs, Jenkins plugins, or third-party libraries. This allows you to define libraries which encapsulate individually unsafe APIs in a higher-level wrapper safe for use from any Pipeline.

## Folder-level Shared Libraries

Any Folder created can have Shared Libraries associated with it. This mechanism allows scoping of specific libraries to all the Pipelines inside of the folder or subfolder.

Folder-based libraries are not considered "trusted:" they run in the Groovy sandbox just like typical Pipelines.

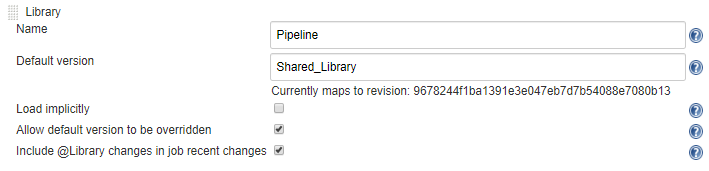
## Automatic Shared Libraries

Other plugins may add ways of defining libraries on the fly. For example, the GitHub Branch Source plugin provides a "GitHub Organization Folder" item which allows a script to use an untrusted library such as github.com/someorg/somerepo without any additional configuration. In this case, the specified GitHub repository would be loaded, from the master branch, using an anonymous checkout.

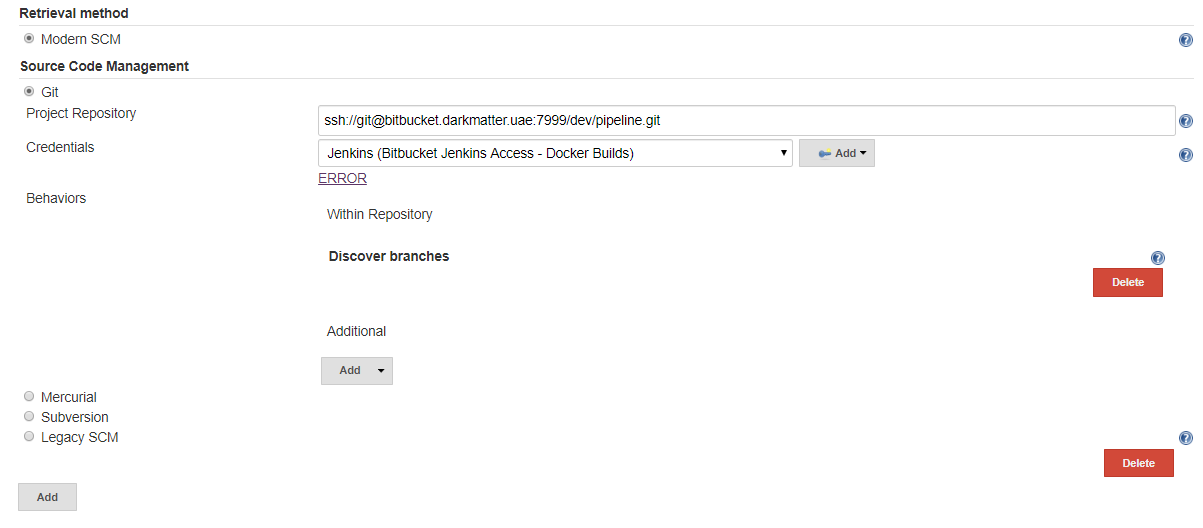
# Jenkins Pipeline Using Shared Library.

To create a pipeline need to configure Jenkins to define our shared library.

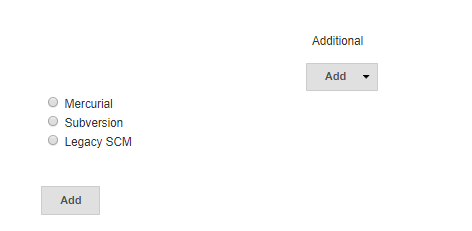
* Go to manage Jenkins -> Configure system -> Global Pipeline Libraries, define your library as necessary can be configured.



* Provide the SCM credentials to fetch the shared library from the defined repo in the followimg tab.



* To add More such Libraries click on Add tab below and provide the details as necessary.



* To invoke the Shared Library to our pipeline script use the syntax as below.

@Library('Pipeline@Shared\_Library')\_

Pipeline – is the repo name

Shared\_Library – is the Shared Library branch.

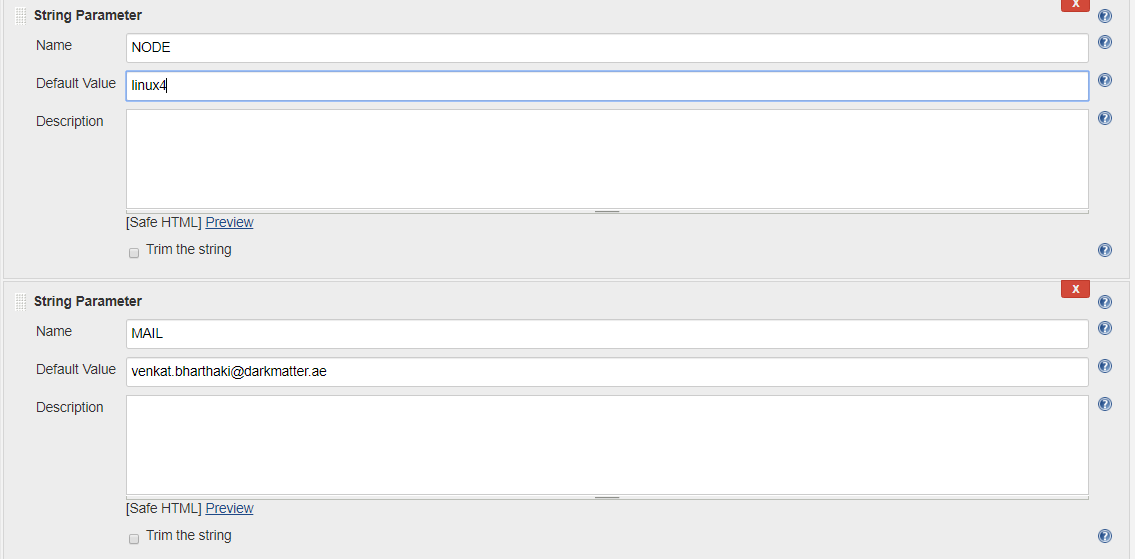
* In the Shared Library we have the directory structure as below

vars/ //where we have the .groovy files which is call from jenkinsfile

src/ //where all the class files are stored and are exported to jenkinsfile and Shared Library

Example – To send a sample email using Shared Library

* Shared Library configurations are done as mentioned above.
* Jenkins job is configured by giving parameters for mail and node



* Create a branch for Jenkins file and provide the Shared Library details.

@Library('Pipeline@Shared\_Library')\_

Email()

* Create a branch for Share Library where we have our template for Jenkinsfile.

import com.org.xebia.\* //to import from src directory

def call(Map stageParams){

s1 = new Notification() //creating an object for our class file

timeout(time: 240000, unit: 'SECONDS', activity: false) {

node("${env.NODE}") {

try{

properties() //to call the groovy step from shared library

stage('Notification') {

s1.email() //loading the class files

}

}

catch (any) {

currentBuild.result = 'FAILED'

}

}

}

}

* Creating a class file in the same Shared Library branch

Inside the src directory create the directory structure.

src/com/org/xebia/

in this present directory create a class file with .groovy extension

For ex. Notification.groovy

package com.org.xebia //package the class file

def email(){ //define the type

script{

emailext (

body: '''${SCRIPT, template="email-html-dm.template"}''',

mimeType: 'text/html',

to: "${env.MAIL}",

subject: "[JENKINS] ${env.JOB\_NAME}")

}

}

* After committing all the code build the job, and the console output is shown below.

