Introduction

[Jenkins](https://phoenixnap.com/kb/what-is-jenkins) is an open-source automation server for software developers. It lets users create and manage repeatable elements that make software development and deployment more straightforward and efficient. One of these elements are Jenkins shared libraries.

**In this tutorial, we will define what Jenkins shared libraries are and explain how you can create, manage, and use them in software development.**

Prerequisites

* A copy of Jenkins installed and ready to use (learn how to [Install Jenkins on Windows 10](https://phoenixnap.com/kb/install-jenkins-on-windows), [Ubuntu 18.04](https://phoenixnap.com/kb/install-jenkins-ubuntu), [Debian 10](https://phoenixnap.com/kb/install-jenkins-on-debian-10), [CentOS 8](https://phoenixnap.com/kb/how-to-install-jenkins-on-centos-8), or [install Jenkins on a Kubernetes cluster](https://phoenixnap.com/kb/how-to-install-jenkins-kubernetes))
* Access to a web browser

**What Is a Shared Library in Jenkins?**

A shared library in Jenkins is a collection of Groovy scripts shared between different Jenkins jobs. To run the scripts, they are pulled into a Jenkinsfile.

Each shared library requires users to define a name and a method of retrieving source code. These methods include local files, [Git repositories](https://phoenixnap.com/kb/what-is-a-git-repository), and Jenkins SCM plugins. Optionally, users can also set a default library version.

**Note**: Shared libraries are stored in git repositories. Versioning, tagging and other Git functionalities are at the user's disposal.

**Why Use a Jenkins Shared Library?**

Developers use shared libraries to avoid writing the same code from scratch for multiple projects. Shared libraries share code across development projects, thus optimizing the [software development life cycle](https://phoenixnap.com/blog/software-development-life-cycle). This drastically cuts down time spent on coding and helps avoid duplicate code.

Creating a shared library also simplifies the process of pushing source code updates for a project. Updating the library source code also updates the code of every project that uses the library.

**How to Create a Shared Library in Jenkins?**

Creating a shared library in Jenkins involves the following steps.

**Step 1: Create a Groovy Script**

First, create a Groovy script using the code you want to save to the library.

For example, the following Groovy script creates a greeting function that prints **Hello** and a user-defined **name** parameter as the output:

#!/usr/bin/env groovy

def call(String name = 'human') {

echo "Hello, ${name}."

}

**Step 2: Add the Script to a Git Repository**

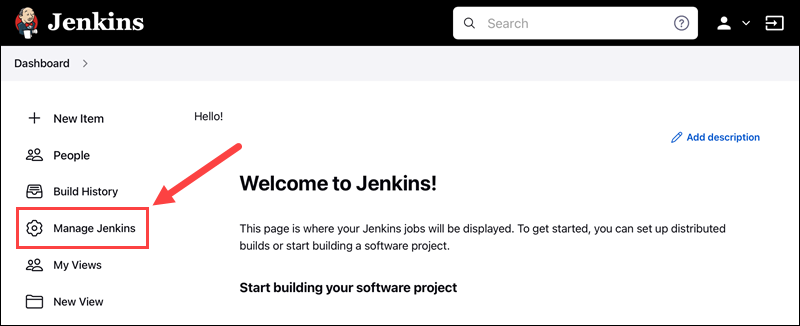
Create a new Git repository containing a directory named *vars*. Save the script in the *vars* directory as *sayHello.groovy*.



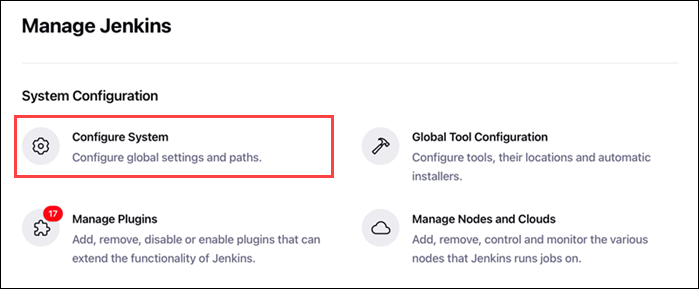
**Step 3: Add a Shared Library in Jenkins**

1. Open the Jenkins dashboard in your web browser.

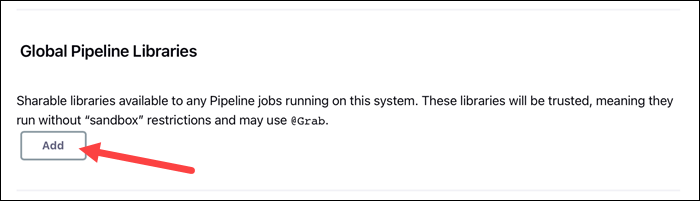
2. Click the **Manage Jenkins** link on the left-hand side of the Jenkins dashboard.



3. Under *System Configuration*, click the **Configure System** button.



4. Scroll down to the *Global Pipeline Libraries* section and click the **Add** button.

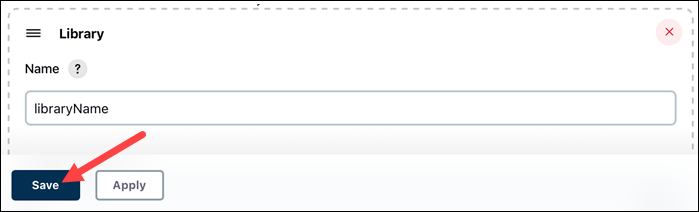


5. A new dialog opens prompting you to set up library details, such as name and SCM method. Fill in the details to configure the library:

* **Name:** The name of the new Shared Library.
* **Default version:** Set a default version for the new Shared Library. Depending on the SCM, this can be a branch name, tag, commit hash, etc.
* **Load implicitly:** Checking this option allows scripts to automatically access the library without needing to request it.
* **Allow default version to be overridden:** Checking this option allows scripts to select a custom version of the library.
* **Include @Library changes in job recent changes:** Checking this option causes changes to the library to affect all the builds that use the library.
* **Cache fetched versions on master for quick retrieval:** Checking this option makes Jenkins cache fetched versions of this library on the master.
* **Retrieval method:** Choose Modern SCM (when using Jenkins SCM plugins) or Legacy SCM (when using SCM methods not integrated with Jenkins) from the drop-down menu. Selecting Modern SCM lets you choose an SCM plugin and configure relevant options.
* **Library Path (optional):** Lets you set a relative path from the SCM root to the root of the library directory.

**Note:** Checking the **Load implicitly** option when creating a shared library automatically loads the library in every pipeline you make.

6. Once you are done configuring the new library, click the **Save** button to save the changes to Jenkins.



**Note:** Need a cheap sandboxing environment with automated OS-deployment options? See how easy it is to [deploy a development sandbox](https://phoenixnap.com/kb/setting-up-sandbox-on-s0-bmc) for as little as $0.10/hour.

**Step 4: Use a Shared Library in a Jenkins Pipeline**

After you add the shared library from the Git repository to Jenkins, you can refer to it in your pipeline scripts.

1. Create a new pipeline item in Jenkins. Type the item's name and select **Pipeline** as the project type. Click OK once done.

2. Add a simple script that loads the shared library. We used *Alex* as the **name** parameter:

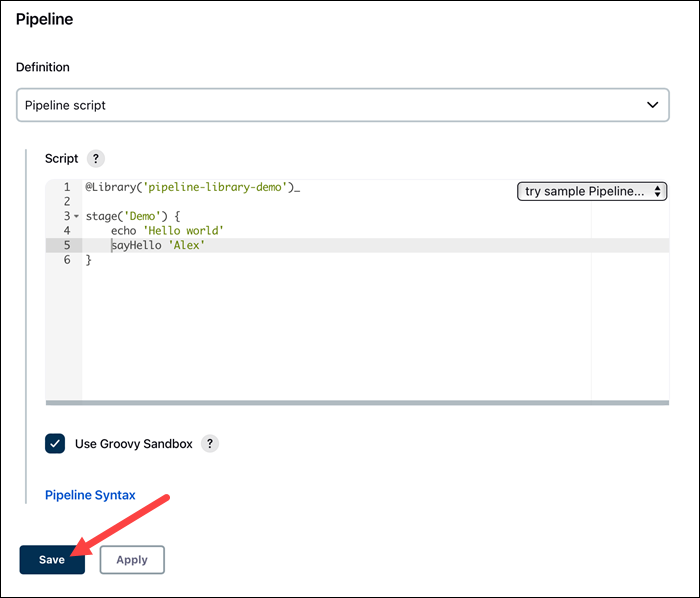
@Library('pipeline-library-demo')\_

stage('Demo') {

echo 'Hello world'

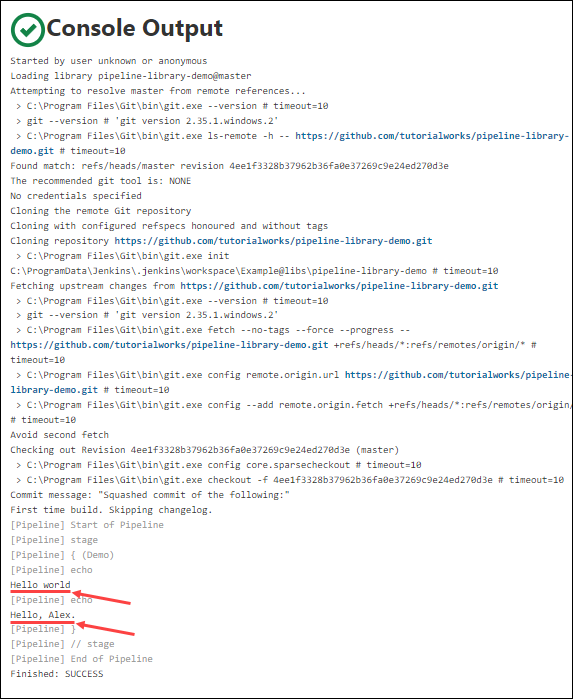
sayHello 'Alex'

}



3. Click **Save** to save the changes to the pipeline.

4. Building the pipeline and checking the console output reveals that the shared library correctly loads and interacts with the user input:



**How to Load a Shared Library in a Pipeline**

If a shared library doesn’t load implicitly, Jenkins offers several ways to refer to it using the Groovy script.

Add the following annotation to the top of the Jenkins code to manually refer to a script:

@Library('[library name]')\_

Append the at sign (**@**) and version name to load a specific version of the library:

@Library('[library name]@[library version]')\_

**Note:** Loading a specific version of a shared library requires you to check the **Allow the default version to be overridden** option when creating the library.

If you want to load multiple libraries at the same time, separate the names with a comma symbol (**,**) and a blank space:

@Library(['[library name 1]', '[library name 2]'])\_

**Note:** It is necessary to add an underscore symbol (**\_**) to the end of the **@Library** annotation to avoid errors.

Another option is to load shared libraries dynamically. You can place this type of annotation anywhere in the script:

library '[library name]'

It is also possible to dynamically load a third-party library that isn’t included in the Jenkins *Global Pipeline Libraries* settings. For example, loading a library from Git requires you to provide the library name, a Jenkins class tag, the library's Git URL and valid credentials to access the library.

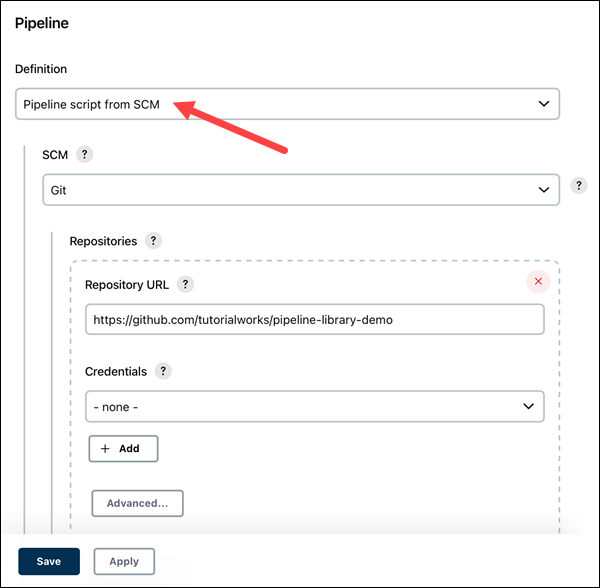
library identifier: '[library name]@[library version]', retriever: modernSCM(

[$class: 'GitSCMSource',

remote: '[library Git URL]',

credentialsId: '[Git credentials]'])

Another method is to link directly to a Git repository containing a library. When configuring a pipeline, scroll down to the *Pipeline* section and select the **Pipeline script from SCM** option from the *Definition* drop-down menu.



Once you select the **Git** option from the *SCM* drop-down menu, add the repository URL, credentials, and branch ID and click **Save** to confirm.

Conclusion

After reading this tutorial, you should be able to create a new shared library using a Groovy script, add it to Jenkins, and refer to it in your pipeline.