**Shallow clone:**

Perform a shallow clone by fetching only the latest commit, instead of the entire repository history.

**Jenkins - Configuration System - Source Code Management - Behaviours (ADD) - check "Advanced clone behaviors" and set "Shallow clone" to true**

—--------------------------------------------------------------------------------------------------------------------------------------------------------------

**Sparse checkout:**

If you only need specific files or directories from the repository, you can use sparse checkout to fetch only those parts.

**Jenkins - Configuration System - Source Code Management - Behaviours (ADD) - check "Sparse Checkout paths" and add the paths you need**

—--------------------------------------------------------------------------------------------------------------------------------------------------------------

**Use a reference repository:**

A reference repository is a local, read-only copy of the repository that Jenkins can use to speed up cloning

**git clone --mirror <repository\_url> <local\_reference\_path>**

**Jenkins - Configuration System - Source Code Management - Behaviours (ADD) - check "Advanced clone behaviors" and set "Path of the reference repo to use during clone" to the local reference path.**

—--------------------------------------------------------------------------------------------------------------------------------------------------------------

**Use a local cache:**

If you have multiple Jenkins jobs that use the same repository, you can set up a local cache to avoid cloning the repository multiple times. Create a Jenkins pipeline shared library and configure your jobs to use the shared library.

=============>>>

1. **Create a shared library repository:**

Create a new Git repository to store your shared library code. This repository will act as a local cache for your Jenkins jobs. You can create this repository on your Git server (e.g., Bitbucket, GitHub, GitLab).

1. **Add a vars directory:**

In the shared library repository, create a **vars** directory. This directory will contain global variables that can be accessed by your Jenkins jobs.

1. **Create a global variable for the local cache:**

Inside the **vars** directory, create a Groovy file named **gitCache.groovy**. This file will define a global variable called **gitCache** that will handle the local cache logic.

Add the following code to **gitCache.groovy**:

**def call(String repositoryUrl, String branch = 'master', String credentialsId = null) {**

**def cacheDir = "${env.JENKINS\_HOME}/gitcache/${repositoryUrl.hashCode()}"**

**def gitExists = fileExists cacheDir**

**if (!gitExists) {**

**dir(cacheDir) {**

**git branch: branch, credentialsId: credentialsId, url: repositoryUrl**

**}**

**}**

**return cacheDir**

**}**

This code defines a **gitCache** function that takes a repository URL, branch, and credentials ID as input. It checks if a local cache exists for the given repository URL. If not, it clones the repository into a cache directory. The function returns the path to the cache directory.

1. **Configure the shared library in Jenkins:**

In your Jenkins instance, go to **"Manage Jenkins" > "Configure System" > "Global Pipeline Libraries". Click "Add"** to create a new shared library.

* **Set "Name" to a unique identifier** for your shared library (e.g., "mySharedLibrary").
* **Set "Default version" to the branch** you want to use (e.g., "master").
* **Under "Retrieval method", select "Modern SCM"** and configure the repository URL, credentials, and branch for your shared library repository.

1. **Use the shared library in your Jenkins jobs:**

In your Jenkins job's pipeline script, load the shared library and use the **gitCache** function to clone the repository using the local cache. Here's an example pipeline script:

**@Library('mySharedLibrary') \_**

**pipeline {**

**agent any**

**stages {**

**stage('Checkout') {**

**steps {**

**script {**

**def repositoryUrl = 'https://bitbucket.org/user/repo.git'**

**def branch = 'master'**

**def credentialsId = 'your-credentials-id'**

**def cacheDir = gitCache(repositoryUrl, branch, credentialsId)**

**dir("${WORKSPACE}") {**

**checkout([$class: 'GitSCM',**

**branches: [[name: branch]],**

**doGenerateSubmoduleConfigurations: false,**

**extensions: [[$class: 'RelativeTargetDirectory', relativeTargetDir: '']],**

**submoduleCfg: [],**

**userRemoteConfigs: [[url: cacheDir, credentialsId: credentialsId]]])**

**}**

**}**

**}**

**}**

**stage('Build') {**

**steps {**

**// Your build steps here**

**}**

**}**

**}**

**}**

This pipeline script loads the shared library, calls the **gitCache** function to get the local cache directory, and then checks out the repository using the local cache.

By using this shared library and the **gitCache** function, you can avoid cloning the repository multiple times and speed up your Jenkins jobs.

—--------------------------------------------------------------------------------------------------------------------------------------------------------------

In this scenario, we will set up a reference repository to speed up cloning in a Jenkins job. We will use a Git repository hosted on GitHub as an example.

1. **Clone the repository with the -- option on the Jenkins server:**

First, SSH into your Jenkins server and navigate to a directory where you want to store the reference repository. Then, run the following command to create a mirror of the repository:

git *clone* --mirror https://github.com/user/repo.git /path/to/*local*/reference/repo

Replace **https://github.com/user/repo.git** with the URL of your Git repository and **/path/to/local/reference/repo** with the desired local path for the reference repository.

1. **Configure the Jenkins job to use the reference repository:**

In your Jenkins job configuration, follow these steps:

* Under "Source Code Management", select "Git".
* Enter the repository URL and credentials (if required).
* Click the "Add" button next to "Additional Behaviours" and select "Advanced clone behaviours".
* In the "Path of the reference repo to use during clone" field, enter the local path of the reference repository you created in step 1 (e.g., **/path/to/local/reference/repo**).

1. **Save the job configuration and run the job:**

Save the changes to your Jenkins job configuration and run the job. Jenkins will now use the reference repository to speed up the cloning process.

Here's an example of how the Jenkins job configuration might look:

* Repository URL: **https://github.com/user/repo.git**
* Credentials: **your-credentials-id**
* Additional Behaviours:
  + Advanced clone behaviours:
    - Path of the reference repo to use during clone: **/path/to/local/reference/repo**

By using a reference repository, Jenkins can significantly reduce the time it takes to clone a repository, especially for large repositories with a long history. The reference repository acts as a local cache, allowing Jenkins to fetch only the changes since the last update, rather than downloading the entire repository history.

**OR**

If you want to set up reference repositories for multiple Git repositories at once, you can create a script that clones all the repositories with the **--mirror** option and stores them in a designated directory on the Jenkins server. Then, configure each Jenkins job to use the corresponding reference repository.

Here's an example of how you can achieve this:

1. **Create a text file with a list of repository URLs:**

Create a file named **repo\_list.txt** and list the repository URLs, one per line:

https://github.com/user/repo1.git

https://github.com/user/repo2.git

https://github.com/user/repo3.git

1. **Create a script to clone all repositories:**

Create a script named clone\_all\_repos.sh with the following content:

#!/bin/bash

REFERENCE\_REPO\_DIR="/opt/jenkins/reference-repos"

REPO\_LIST\_FILE="repo\_list.txt"

mkdir -p "${REFERENCE\_REPO\_DIR}"

*while* *read* -r repo\_url; *do*

repo\_name=$(basename "${repo\_url}" .git)

git *clone* --mirror "${repo\_url}" "${REFERENCE\_REPO\_DIR}/${repo\_name}"

done < "${REPO\_LIST\_FILE}"

This script reads the repository URLs from repo\_list.txt, clones each repository with the --mirror option, and stores them in the /opt/jenkins/reference-repos directory.

1. **Run the script on the Jenkins server:**

Upload the repo\_list.txt and clone\_all\_repos.sh files to your Jenkins server and run the script:

chmod +x clone\_all\_repos.sh

./clone\_all\_repos.sh

This will create reference repositories for all the listed repositories in the /opt/jenkins/reference-repos directory.

1. **Configure Jenkins jobs to use the reference repositories:**

For each Jenkins job, configure the "Path of the reference repo to use during clone" setting under "Advanced clone behaviours" to point to the corresponding reference repository:

/opt/jenkins/reference-repos/repo1

/opt/jenkins/reference-repos/repo2

/opt/jenkins/reference-repos/repo3

By following these steps, you can set up reference repositories for multiple Git repositories at once and configure your Jenkins jobs to use them, speeding up the cloning process for all jobs.