



Jenkins

Jenkins File

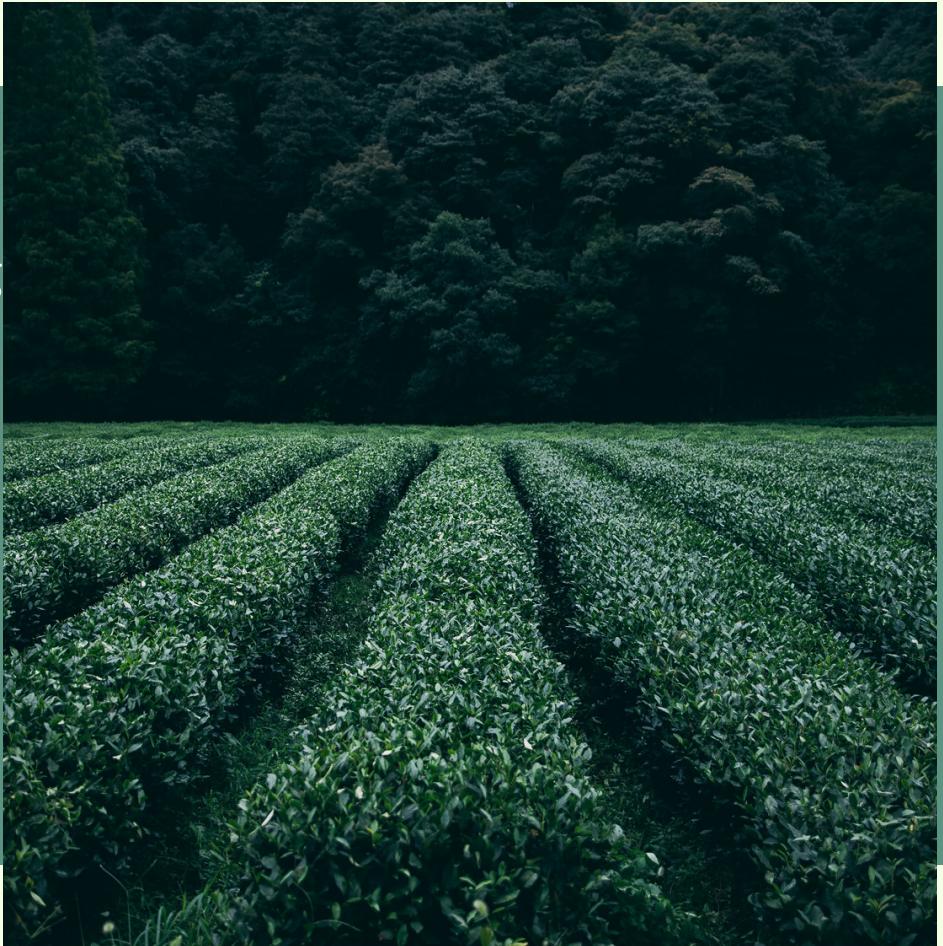
JENKINS

BY ALOK SRIVASTAVA - NETWORK NUTS



JENKINSFILE

A pipeline in Jenkins is defined using a script called the Jenkinsfile. This provides more automation, allows your pipeline to be treated like all other application code, and can be stored in version control.



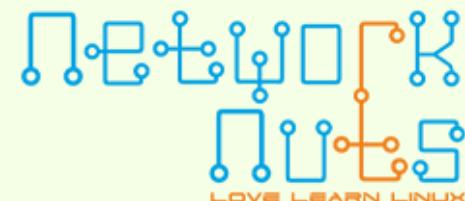
The pipeline syntax can be presented in two forms: **declarative** and **scripted**.

The declarative syntax is a simple and opinionated way of writing your pipeline.

The scripted pipeline is built with Groovy and is generally a more flexible and expressive way of creating your pipelines. When choosing which model to use, it all depends on your requirements.

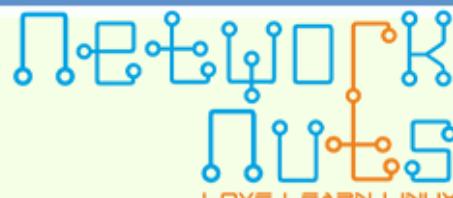
The declarative model works with simple pipelines and lacks most of the flexibility offered by the scripted model.

We will be working with the scripted pipeline



The scripted pipeline has some special directives that perform different functions.

Directive	Explanation node
<code>node</code>	This defines where the job is going to be run. We will explore more about this in the next chapter as we cover setting up master-slave relationships on Jenkins.
<code>dir</code>	This directive defines what directory/folder to run the following directives on.
<code>stage</code>	This defines the stage of your pipeline, for example, what task it's running.
<code>git</code>	This points to the remote repository where you pull the changes from.
<code>sh</code>	This defines the shell script to run on a UNIX-based environment. On a Windows environment, we would use the <code>bat</code> directive instead.
<code>def</code>	As mentioned previously, the pipeline is written in Groovy; thus, we can define functions to perform different actions. In this case, we defined a <code>printMessage</code> function, which prints out different messages at the start and end of our pipeline.



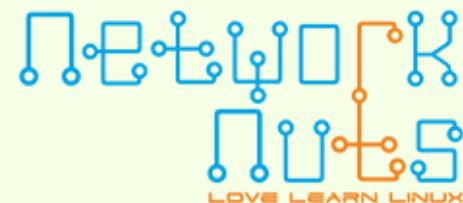
Creating the Pipeline.

Go to the Jenkins dashboard and select New Item.

Enter an appropriate name for the project and select Pipeline for the project type.

In the project configuration, under the General tab,

select GitHub project and enter the appropriate URL.



Enter an item name

pipelinetest

» 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.



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.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



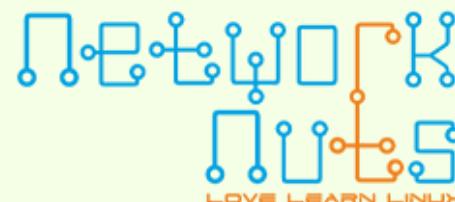
Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Organization

Organizations are a way to group Jenkins instances together.



General Build Triggers Advanced Project Options Pipeline

Description

[Plain text] [Preview](#)

Discard old builds [?](#)

Do not allow concurrent builds [?](#)

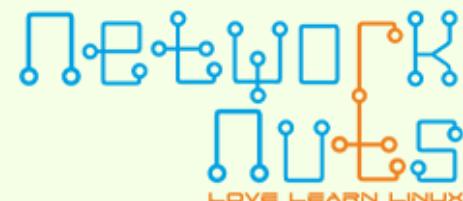
Do not allow the pipeline to resume if the master restarts [?](#)

GitHub project [?](#)

Project url [?](#)

[Advanced...](#)

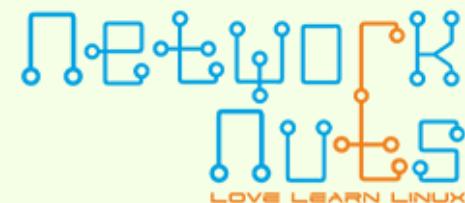
Under the **Build Triggers** section, select the **GitHub hook trigger for GITScm polling**, which will help us automatically trigger builds on our pipeline whenever a commit is pushed to GitHub. This configuration will only work with a **hosted Jenkins server** and configured GitHub webhook on the repository.



Build Triggers

- Build after other projects are built ?
- Build periodically ?
- GitHub hook trigger for GITScm polling ?

The final configuration of the project is creating our pipeline.

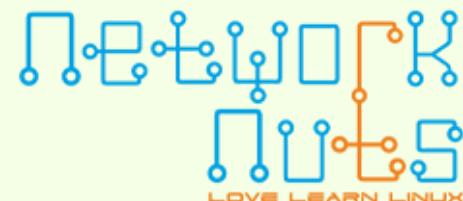


Your final pipeline script configuration should look like this

Pipeline script

```
Script
1 -> node('master') {
2 ->     stage("Fetch Source Code") {
3 ->         git "https://github.com/alokaryan/jenkinslab"
4 ->     }
5 ->     dir('jenkinslab') {
6 ->         printMessage('Running Pipeline')
7 ->         stage("Testing") {
8 ->             sh 'python echo.py'
9 ->         }
10 ->         printMessage('Job Completed')
11 ->     }
12 ->
13
14 -> def printMessage(message){
15 ->     echo "${message}"
16 -> }
17 |
```

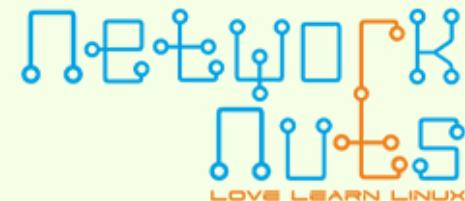
Use Groovy Sandbox



Press **Apply** to save the configuration so far before Jenkins logs you out, which may result in some data loss.

Select Save to persist your configuration and redirect you back to the project dashboard.

Select Build Now on the left-hand menu to build your project. Since we have no commits to push, we will manually trigger a build.





[Recent Changes](#)

Stage View

trend =

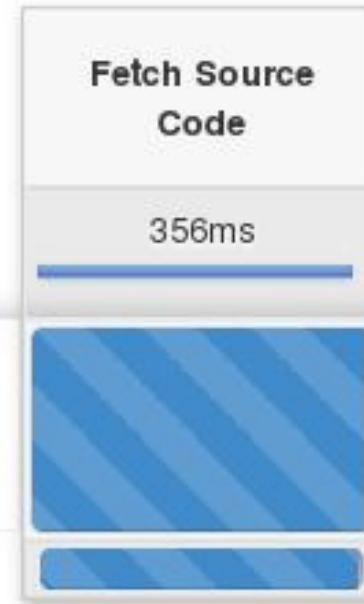
X

M

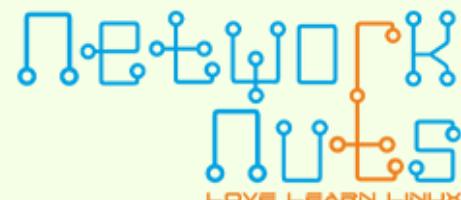
↑
↑
↓

Average stage times:

#1	Apr 02 07:04	No Changes
----	-----------------	------------



```
Running on Jenkins in /var/lib/jenkins/workspace/pipelinetest
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Fetch Source Code)
[Pipeline] git
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/alokaryan/jenkinslab
> git init /var/lib/jenkins/workspace/pipelinetest # timeout=10
Fetching upstream changes from https://github.com/alokaryan/jenkinslab
> git --version # timeout=10
> git fetch --tags --progress https://github.com/alokaryan/jenkinslab +refs/heads/*:re...
> git config remote.origin.url https://github.com/alokaryan/jenkinslab # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/alokaryan/jenkinslab # timeout=10
Fetching upstream changes from https://github.com/alokaryan/jenkinslab
> git fetch --tags --progress https://github.com/alokaryan/jenkinslab +refs/heads/*:re...
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 68488740778c03f2829c5f24304d0c2df5c72617 (refs/remotes/origin/mas...
> git config core.sparsecheckout # timeout=10
> git checkout -f 68488740778c03f2829c5f24304d0c2df5c72617
> git branch -a -v --no-abbrev # timeout=10
> git checkout -b master 68488740778c03f2829c5f24304d0c2df5c72617
Commit message: "Merge pull request #2 from alokaryan/test"
First time build. Skipping changelog.
[Pipeline]
```



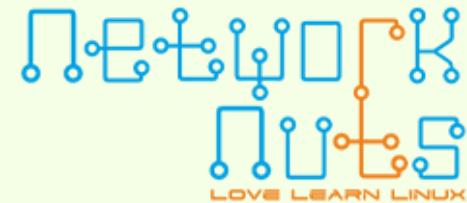
A scenic landscape featuring rolling green hills and a valley below under a clear blue sky with scattered white clouds. A dirt path leads through the foreground towards the horizon.

installing

BLUE OCEAN

Jenkins has a new way of visualizing pipelines, called Blue Ocean.

On the Jenkins home dashboard, go to Manage Jenkins -> Manage Plugins. Under the Available tab, search for Blue Ocean.



 New Item People Build History Manage Jenkins My Views Lockable Resources Credentials Open Blue Ocean New View

Network Nuts

[All](#) [+](#)

S	W	Name	Last Success
		<u>ONE</u>	2 days 3 hr - #1
		<u>pipelinetest</u>	N/A
		<u>projectparameter</u>	2 days 4 hr - #2
		<u>testgit</u>	2 days 5 hr - #2
		<u>TWO</u>	2 days 3 hr - #1

Icon: [S](#) [M](#) [L](#)**Build Queue** 
No builds in the queue.