

# DevOps in Action

# SCM Hands-on

# Github - Create an account

- Go to <https://github.com/join> in a web browser.

Create your personal account

**Username \***

✓

This will be your username. You can add the name of your organization later.

**Email address \***

✓

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

**Password \***

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Verify account

✓

wikiHow to Create an Account on GitHub

# Github - Create an account

- **Enter your personal details**
- **Click the "Create an account" button.**
- Complete the CAPTCHA puzzle.
- Click the "Verify email address" button in the message from GitHub.
- Select your preferences and click Submit.
- Open Inbox and search for email from - [noreply@github.com](mailto:noreply@github.com)
  - Click - "Verify Email Address"

# GitHub Repositories

- Contain all the repositories on which the user is working.

# Github - fork your application code

- Visit - <https://github.com/atingupta2005/hello-world-maven>
- A fork is a copy of a repository.
- Forking a repository allows you to freely experiment with changes without affecting the original project.

The screenshot shows the GitHub repository page for `atingupta2005 / hello-world-maven`. The repository has 1 watch, 0 stars, and 0 forks. The `Fork` button is highlighted with a red rectangle. Below the repository name, there are tabs for `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, and `Insights`. The `Code` tab is selected. Below the tabs, there are buttons for `Go to file`, `Add file`, and `Code`. The `Code` button is green. Below the buttons, there is a table showing the commit history. The first commit is by `atingupta2005` with the message `first commit`, hash `ae8a60e`, and timestamp `2 hours ago`. It has 1 commit. Below the commit history, there is a table showing the file structure. The first file is `src` with the message `first commit` and timestamp `2 hours ago`. The second file is `gitignore` with the message `first commit` and timestamp `2 hours ago`.

atingupta2005 / hello-world-maven

Watch 1 Star 0 Fork 0

<> Code ! Issues 🔗 Pull requests ▶ Actions 📁 Projects 📖 Wiki 🛡 Security 📈 Insights

🔗 master 1 branch 0 tags

Go to file Add file Code

atingupta2005 first commit ae8a60e 2 hours ago 1 commits

|           |              |             |
|-----------|--------------|-------------|
| src       | first commit | 2 hours ago |
| gitignore | first commit | 2 hours ago |

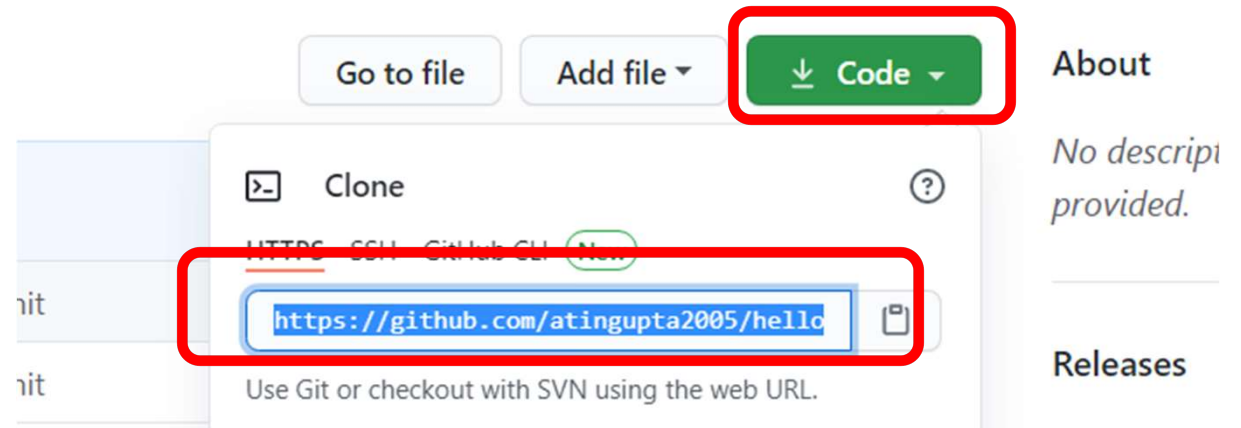
About

No description, website, or topics provided.

Releases

# Git clone the github code

- Cloning a repository pulls down a full copy of all the repository data that GitHub has at that point in time
- The git clone command is used to create a copy of a specific repository or branch within a repository.



# Use maven to compile & package java source code



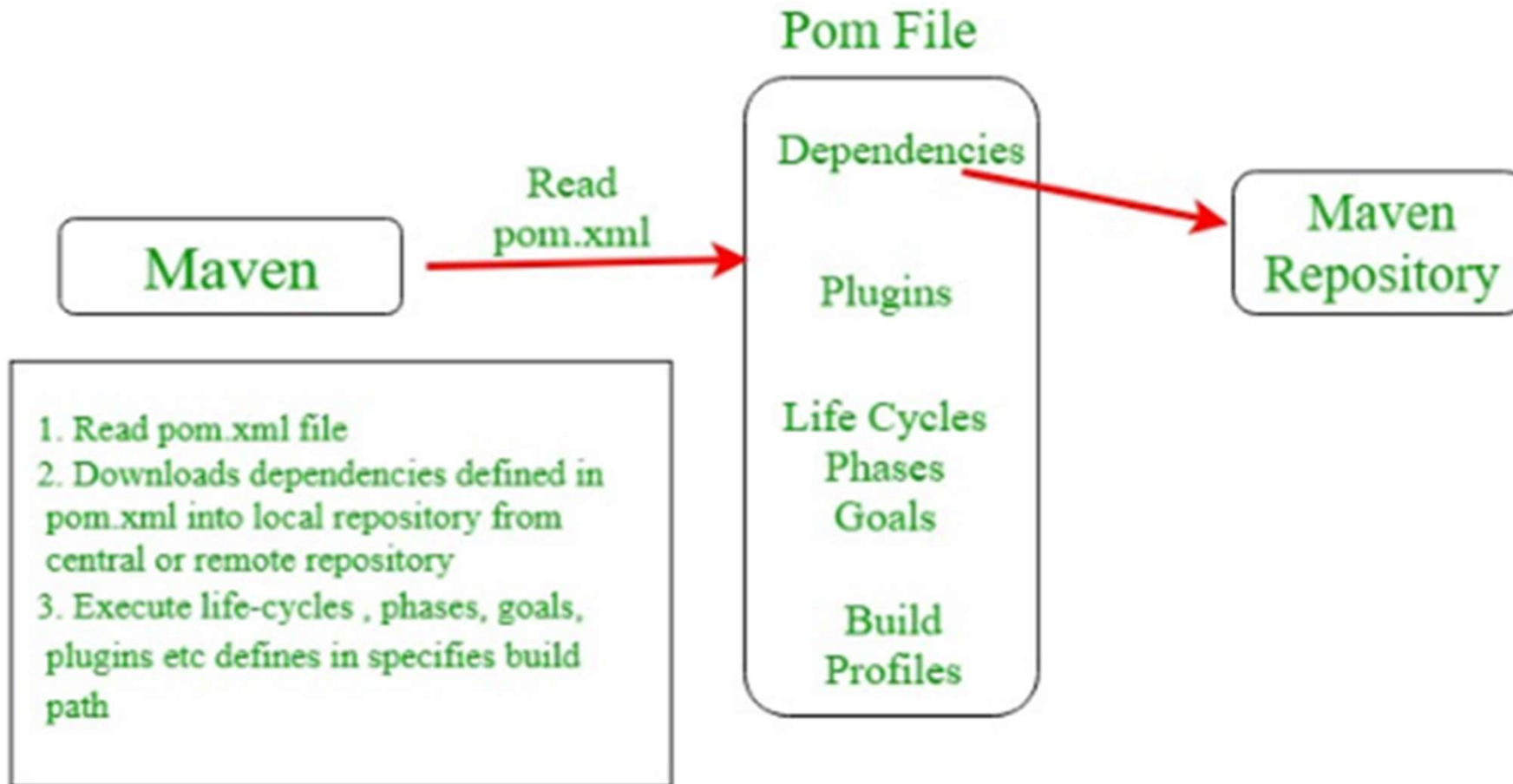
# What is Maven?

- A powerful project management tool that is based on POM (Project Object Model)
- It is used for project build, dependency and documentation
- It can be used for building and managing any Java-based project

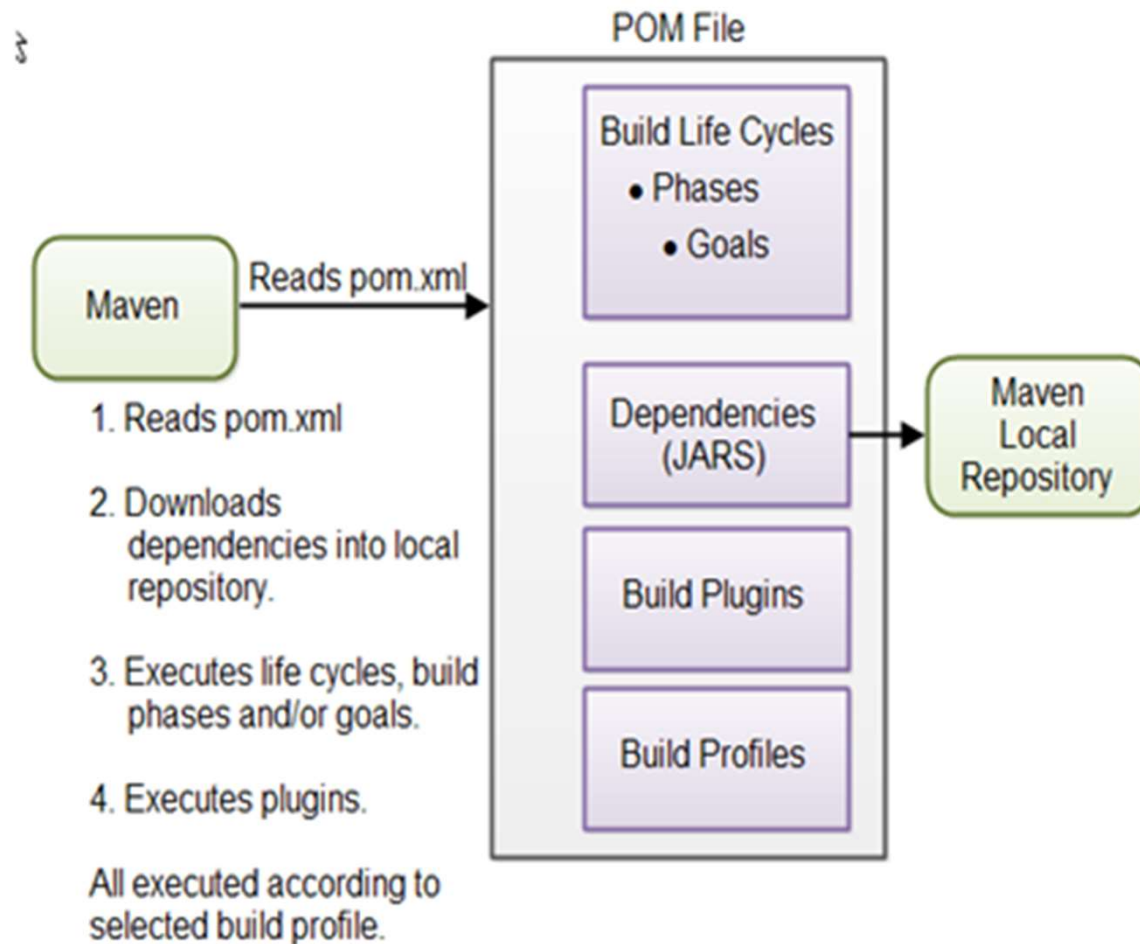
# What can Maven do?

- Can easily build a project
- Can add jars, plugins and other dependencies based on needs
- Helpful in updating central repository of JAR's and other dependencies
- Can build any number of projects into packaging as JAR, WAR etc

# How maven works?



# Overview of Maven core concepts



# Steps

- Install JDK 8
- Install Maven
- Set up the project
- Write a Test
- Go through the source code of maven project
- Define a simple Maven build - pom.xml
- Declare Dependencies
- Build Java code

# Commands - Install maven

- `sudo apt -y update`
- `# Check if java already installed?`
- `java --version`
  
- `#Run below steps if Java not installed`
- `sudo apt install -y openjdk-8-jdk`
  
- `# Install maven`
- `sudo apt install -y maven`

# Commands - Install jdk

- `git clone https://github.com/atingupta2005/hello-world-maven.git`
- `cd hello-world-maven/`
- `mvn compile`

# Understanding Builds

- The process of translating source code into an executable application is called a build.



# Build Tools

- Apache Maven: allows building application written in Java
- Gradle
- Ant
- NAnt
- MsBuild

# Deploy .jar file manually

- # Make sure to cd into the project directory:
- pwd
  - /home/atingupta2005/hello-world-maven
- mvn package
- java -jar target/gs-maven-0.1.0.jar

# Maven build lifecycle

- 1. Compile
  - Source code is compiled.
- 2. Test
  - Launches the unit test placed at src/test/java folder
- 3. Validate
  - To validate that the POM is correctly formed according to model version definition.
- 4. Package
  - To group the compiled code in the specified distributable format (jar, war, etc.).
- 5. Install
  - Installs packaged project into local repository. Then, it can be used by other projects.
- 6. Deploy
  - Similar to install
  - Puts the final package on the shared repository

# Continuous Integration Tool

# Jenkins - Deploy Jenkins on Ubuntu server

- Step 1: Install Java
- Step 2: Add the Jenkins Repository
- Step 3: Install Jenkins
- Step 4: Modify Firewall to Allow Jenkins
- Step 5: Set up Jenkins

# Jenkins - Step 1: Install Java

- `java --version`
- #Run below steps if Java not installed
- `sudo apt update`
- `sudo apt install -y openjdk-8-jdk`

# Jenkins - Step 2: Add the Jenkins Repository

- `wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -`
- `sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'`

# Jenkins - Step 3: Install Jenkins

- `sudo apt -y update`
- `sudo apt install -y jenkins`
- `systemctl status jenkins`



# Jenkins - Step 4: Modify Firewall to Allow Jenkins

- `sudo ufw allow 8080`
- `sudo ufw status`
- # Also make sure to add the port # 8080 in Inbound Rules in Azure/AWS Portal

# Jenkins - Step 5: Set up Jenkins

- Visit:
  - [http://your\\_ip\\_or\\_domain:8080](http://your_ip_or_domain:8080)
- Take password:
  - `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`
- Copy the password from your terminal, paste it into the Administrator password field and click Continue.
- Click on the Install suggested plugins box
- Once the plugins are installed, you will be prompted to set up the first admin user
- Fill out all required information and click Save and Continue

# Continuous Integration setup - Jenkins and Github

# Introduction

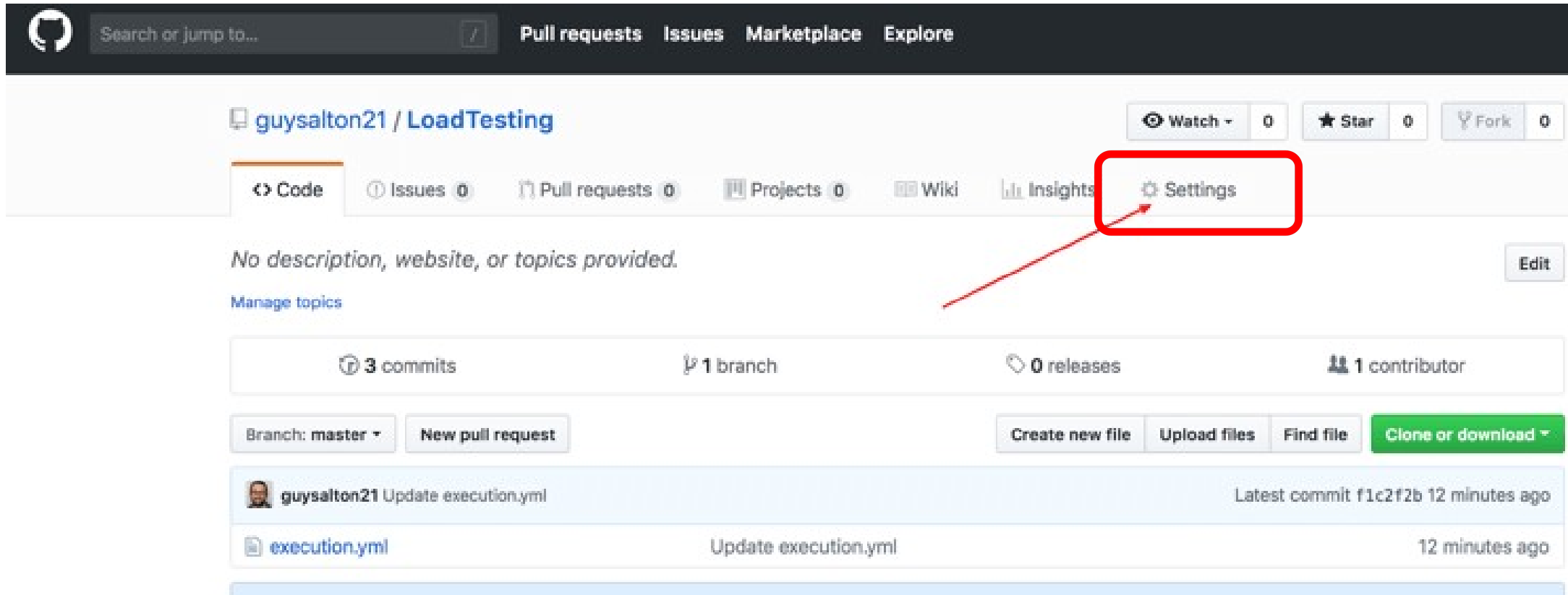
- One of the basic steps of implementing CI/CD is integrating your SCM (Source Control Management) tool with your CI tool.
- This saves you time and keeps your project updated all the time.
- One of the most popular and valuable SCM tools is GitHub.
- We will:
  - Schedule build
  - Pull code and data files from your GitHub repository to Jenkins machine
  - Automatically trigger each build on the Jenkins server, after each Commit on Git repository

# Github Repo

- Fork Github repo:
  - <https://github.com/atingupta2005/hello-world-maven>

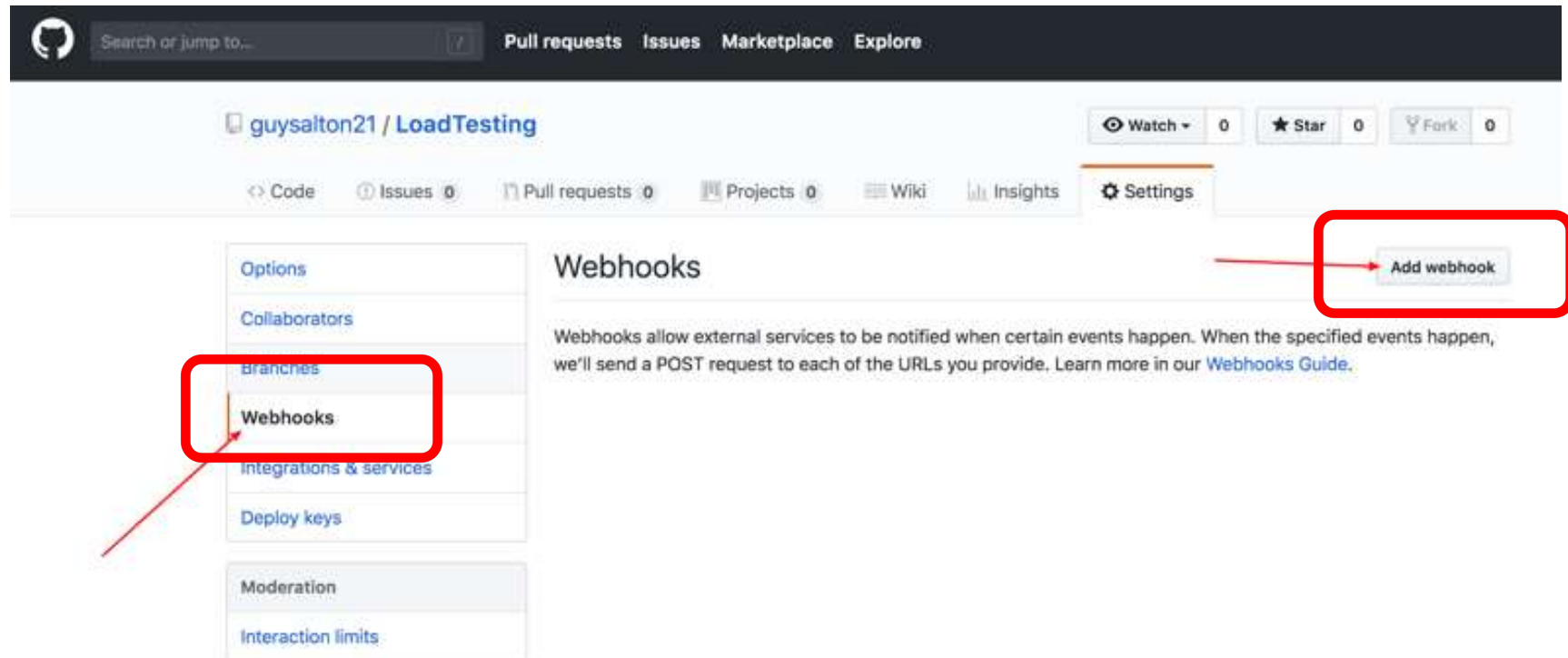
# Configuring GitHub

- Go to your GitHub repository and click on 'Settings'.



# Configuring GitHub

- Step 2: Click on Webhooks and then click on 'Add webhook'.



# Configuring GitHub

- Step 3: in the 'Payload URL' field, paste your Jenkins environment URL.
- At the end of this URL add /github-webhook/
- In the 'Content type' select 'application/json' and leave the 'Secret' field empty.

The screenshot shows the GitHub repository settings for 'guysalton21 / LoadTesting'. The 'Settings' tab is selected, and the 'Webhooks' section is active. The 'Add webhook' form is displayed with the following fields:

- Payload URL \***: `http://jenkins-demo.blazemeter.com:8080/github-webhook/` (highlighted with a red rectangle)
- Content type**: `application/json` (highlighted with a red rectangle)
- Secret**: (empty field)



# Configuring GitHub

- Step 4: in the 'Which events would you like to trigger this webhook?' 'Let me select individual events.'
- Then, check 'Pull Requests' and 'Push'
- At the end of this option, make sure webhook'.

Which events would you like to trigger this webhook?

- ☐ Just the push event.
- ☐ Send me everything.
- ☒ Let me select individual events.

☒ Check runs

Check run is created, requested, rerequested, or completed.

☒ Check suites

Check suite is requested, rerequested, or completed.

☒ Commit comments

Commit or diff commented on.

☒ Branch or tag creation

Branch or tag created.

☒ Branch or tag deletion

Branch or tag deleted.

☒ Deployments

Repository deployed.

☒ Deployment statuses

Deployment status updated from the API.

☒ Forks

Repository forked.

☒ Wiki

Wiki page updated.

☒ Issue comments

Issue comment created, edited, or deleted.

☒ Issues

Issue opened, edited, deleted, transferred, closed, reopened, assigned, unassigned, labeled, unlabeled, milestone, or demilestoned.

☒ Labels

Label created, edited or deleted.

☒ Collaborator add, remove, or changed

Collaborator added to, removed from, or has changed permissions for a repository.

☒ Milestones

Milestone created, closed, opened, edited, or deleted.

# Configuring GitHub

The screenshot shows the GitHub webhook configuration page. It features a list of event types on the left and right, each with a checkbox and a description. The 'Pull requests' checkbox is highlighted with a red box and an arrow pointing to it from above. The 'Pushes' checkbox is also highlighted with a red box and an arrow pointing to it from the left. At the bottom, the 'Active' checkbox is highlighted with a red box and an arrow pointing to it from the right. Below the 'Active' checkbox is a green 'Add webhook' button, which is also highlighted with a red box and an arrow pointing to it from the right.

|   |   |
|---|---|
| <input type="checkbox"/> <b>Visibility changes</b><br>Repository changes from private to public.  | <input checked="" type="checkbox"/> <b>Pull requests</b><br>Pull request opened, closed, reopened, edited, assigned, unassigned, review requested, review request removed, labeled, unlabeled, or synchronized. |
| <input type="checkbox"/> <b>Pull request reviews</b><br>Pull request review submitted, edited, or dismissed.                            | <input type="checkbox"/> <b>Pull request review comments</b><br>Pull request diff comment created, edited, or deleted.  |
| <input checked="" type="checkbox"/> <b>Pushes</b><br>Git push to a repository.  | <input type="checkbox"/> <b>Releases</b><br>Release published in a repository.  |
| <input type="checkbox"/> <b>Repositories</b><br>Repository created, deleted, archived, unarchived, publicized, or privatized.           | <input type="checkbox"/> <b>Repository imports</b><br>Repository import succeeded, failed, or cancelled.  |
| <input type="checkbox"/> <b>Repository vulnerability alerts</b><br>Vulnerability alert created, resolved, or dismissed on a repository. | <input type="checkbox"/> <b>Statuses</b><br>Commit status updated from the API.   |
| <input type="checkbox"/> <b>Team adds</b><br>Team added or modified on a repository.  | <input type="checkbox"/> <b>Watches</b><br>User stars a repository.   |

---

☒ **Active**  
We will deliver event details when this hook is triggered.

**Add webhook**

# Configuring GitHub

- We're done with the configuration on GitHub's side! Now let's move on to Jenkins.

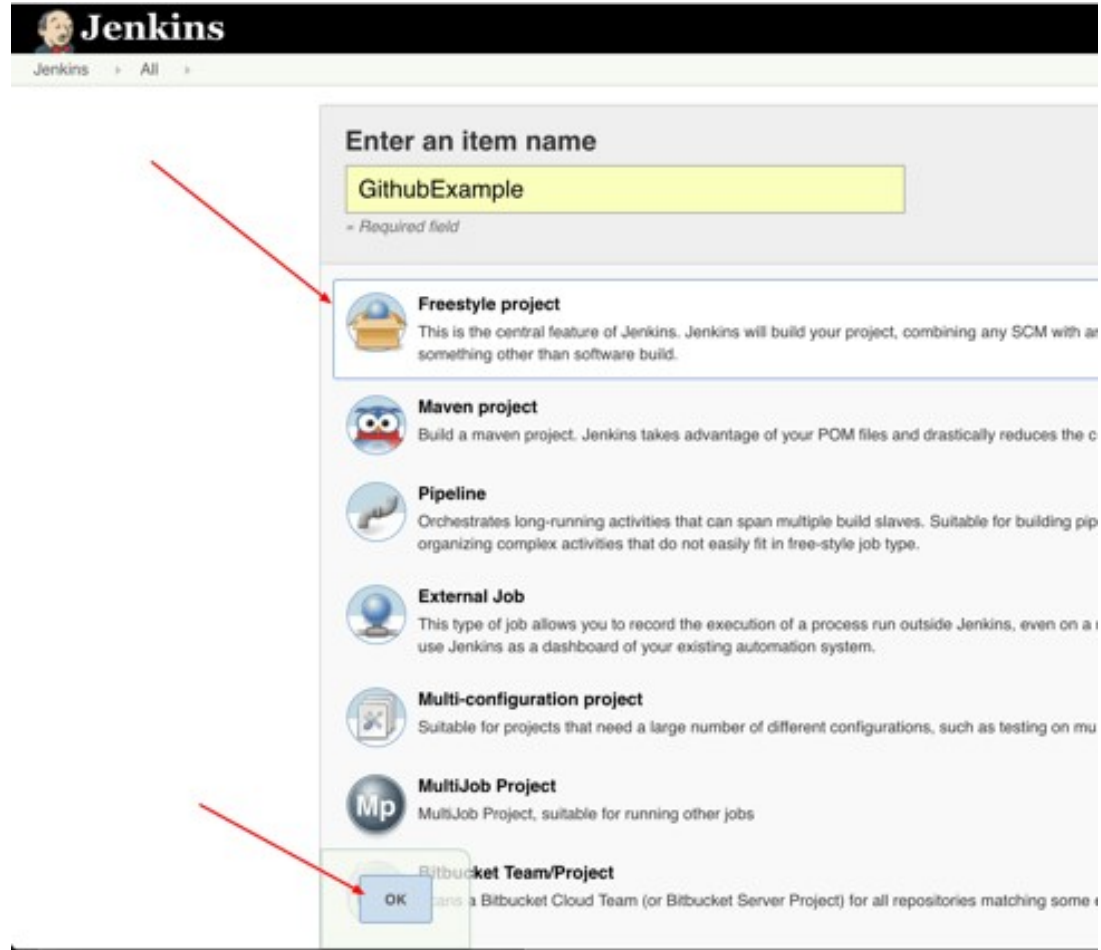
# Configuring Jenkins

- Step 5: In Jenkins, click on 'New Item' to create a new project.



# Configuring Jenkins

- Step 6: Give your project a name, then choose 'Freestyle project' and finally click on 'OK'.



The screenshot shows the Jenkins 'New Item' configuration page. At the top, the Jenkins logo and navigation links are visible. Below the header, there is a section titled 'Enter an item name' with a text input field containing 'GithubExample'. A red arrow points from the 'Freestyle project' option in the list below to the 'OK' button at the bottom of the page. The list of project types includes: Freestyle project, Maven project, Pipeline, External Job, Multi-configuration project, MultiJob Project, and Bitbucket Team/Project. Each option has a brief description.

**Enter an item name**

GithubExample

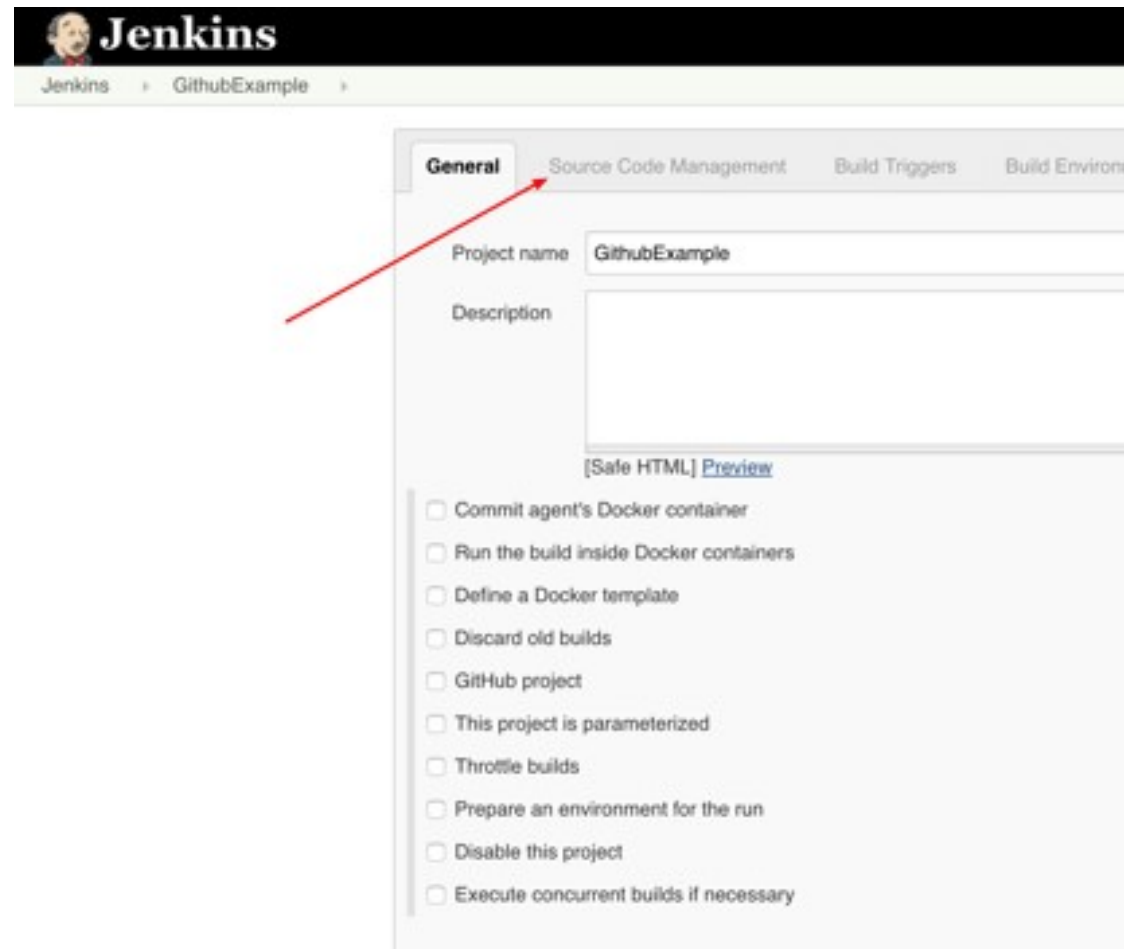
= Required field

- Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with anything other than software build.
- Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the complexity of your build.
- Pipeline**  
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines and organizing complex activities that do not easily fit in free-style job type.
- External Job**  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. Use Jenkins as a dashboard of your existing automation system.
- Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple operating systems.
- MultiJob Project**  
MultiJob Project, suitable for running other jobs
- Bitbucket Team/Project**  
Create a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some criteria.

OK

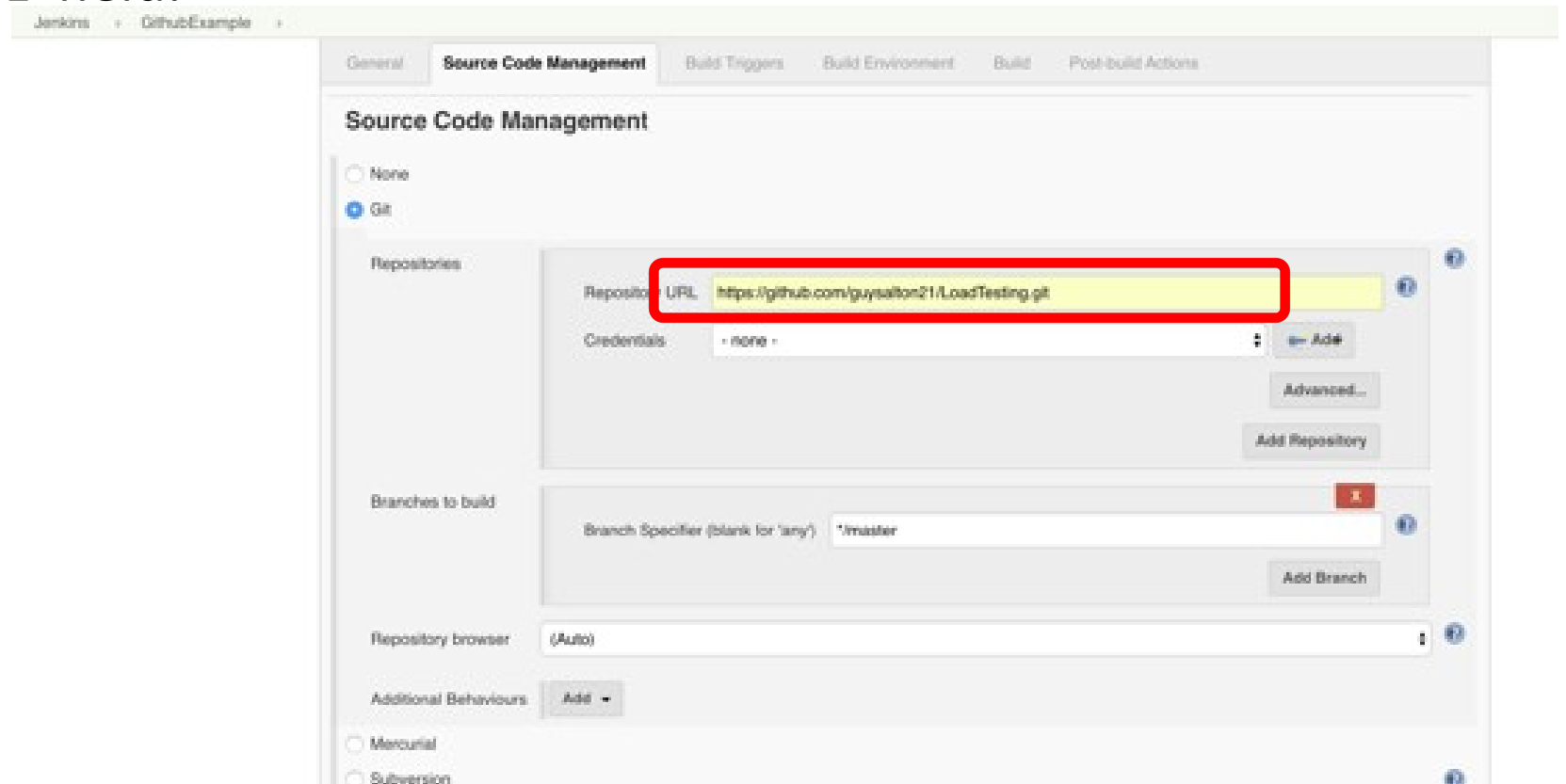
# Configuring Jenkins

- Step 7: Click on the 'Source Code Management' tab.



# Configuring Jenkins

- Step 8: Click on Git and paste your forked GitHub repository URL in the 'Repository URL' field.



The screenshot shows the Jenkins configuration interface for the 'Source Code Management' section. The 'Git' option is selected under the 'Source Code Management' heading. In the 'Repositories' section, the 'Repository URL' field is highlighted with a red rectangle and contains the text 'https://github.com/guysaiton21/LoadTesting.git'. Below this, the 'Credentials' field is set to 'none'. In the 'Branches to build' section, the 'Branch Specifier (blank for 'any')' field is set to '/master'. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. At the bottom, the 'Mercurial' and 'Subversion' options are unselected.

# Configuring Jenkins

- Step 9: Click on the 'Build Triggers' tab and then on the 'GitHub hook trigger for GITScm polling'. Or, choose the trigger of your choice.

The screenshot shows the Jenkins configuration interface for a project named 'GithubExample'. The 'Build Triggers' tab is active, displaying various options for triggering builds. The 'GitHub hook trigger for GITScm polling' option is selected, indicated by a blue checkmark and a red rectangular highlight. Other visible options include 'Mercurial', 'Subversion', 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', 'Build when a change is pushed to BitBucket', 'GitHub Branches', 'GitHub Pull Requests', and 'Poll SCM'. The 'Build Environment' section is partially visible at the bottom.



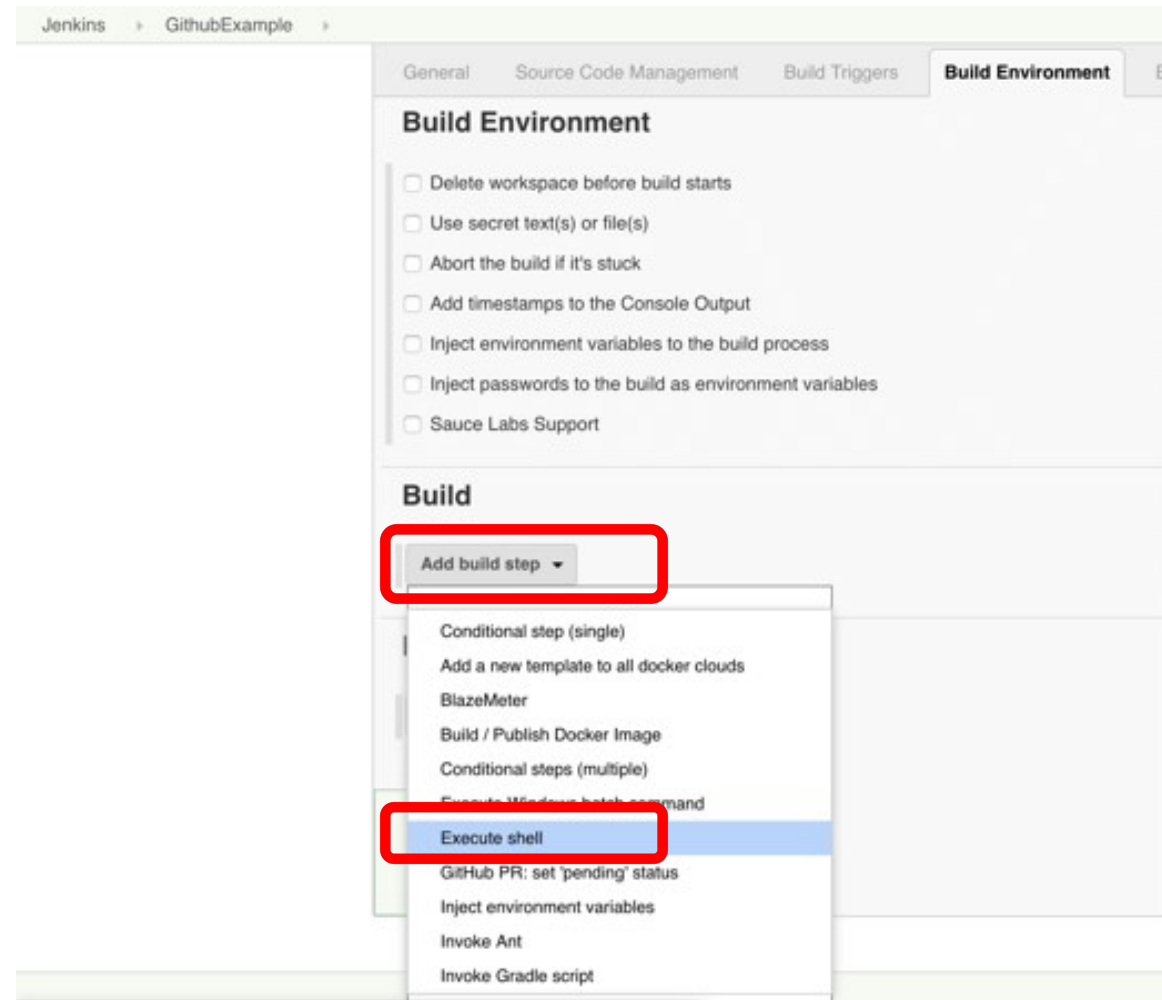
# Configuring Jenkins

- Your GitHub repository is integrated with your Jenkins project.
- You can now use any of the files found in the GitHub repository and trigger the Jenkins job to run with every code commit.

# Triggering the Jenkins Job to Run with Every Code Commit

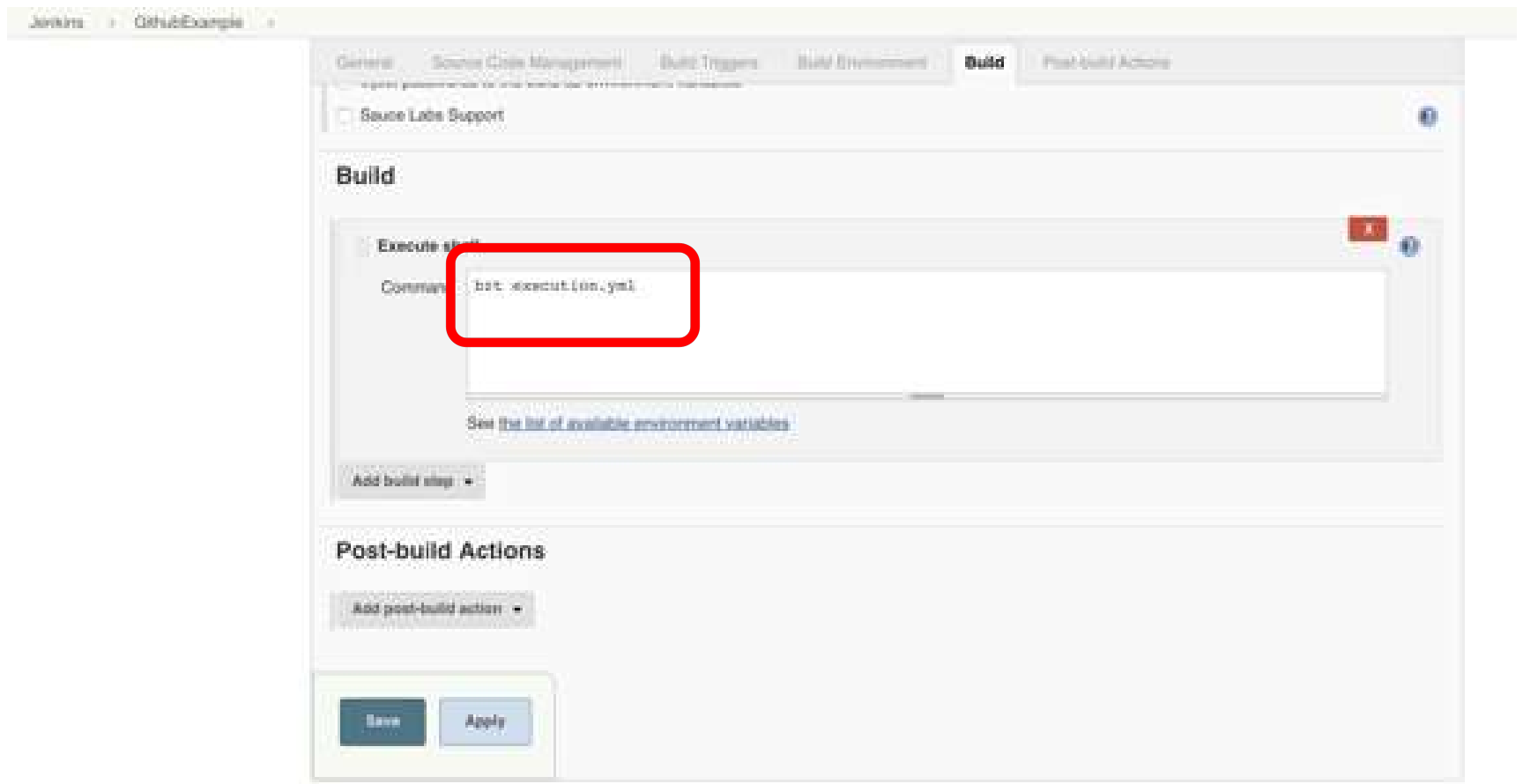
# Triggering the Jenkins Job to Run

- Step 10: Click on the 'Build' tab,
- Then click on 'Add build step' and
- Choose 'Execute shell'.



# Triggering the Jenkins Job to Run

- Step 11: To run sample commands - echo "Building Project"; echo "\$(pwd)"

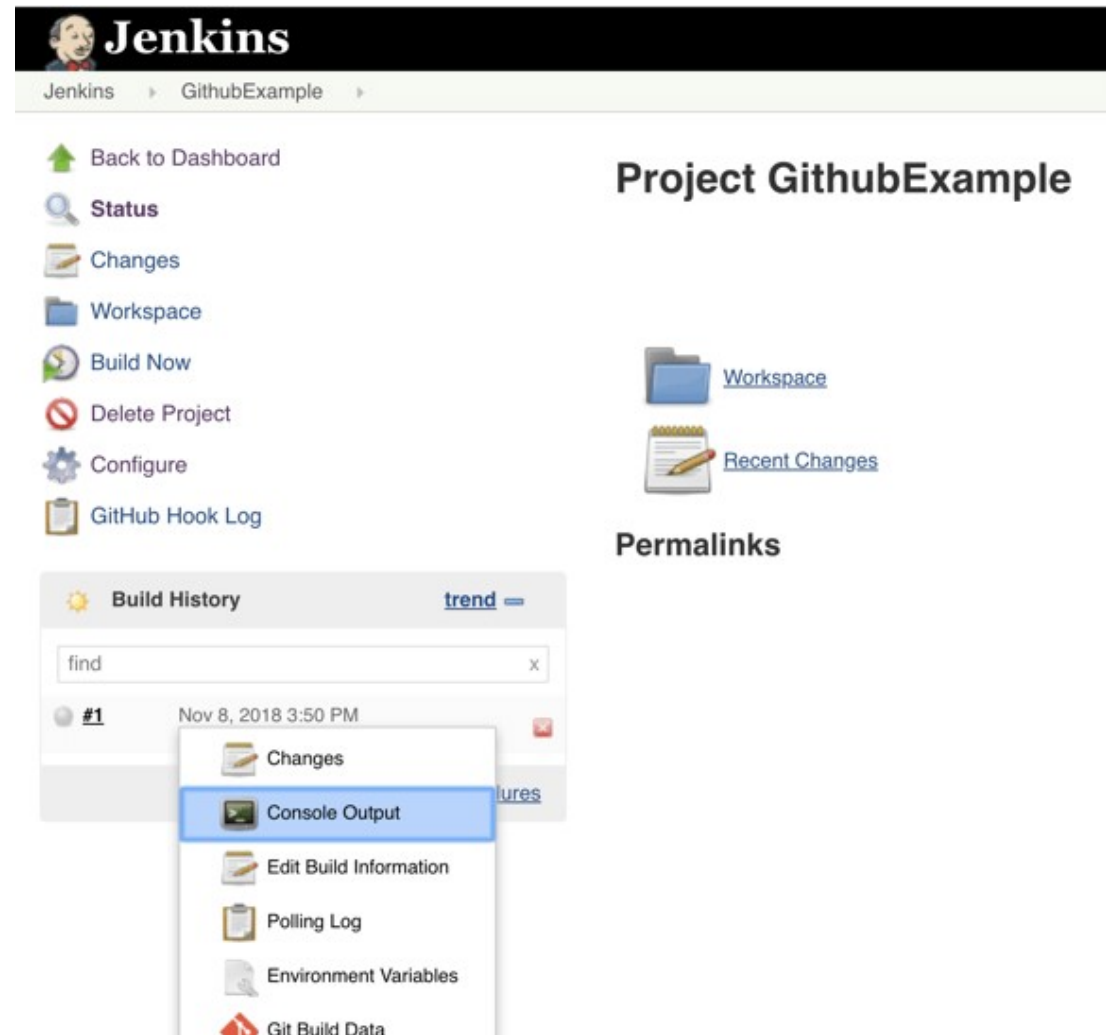


# Triggering the Jenkins Job to Run

- Step 12: Go back to your GitHub repository, edit the code and commit the changes.
  - We will now see how Jenkins ran the script after the commit.

# Triggering the Jenkins Job to Run

- Step 13: Go back to your Jenkins project and you'll see that a new job was triggered automatically from the commit we made at the previous step.
- Click on the little arrow next to the job and choose 'Console Output'.



# Triggering the Jenkins Job to Run

- Step 14: You can see that Jenkins was able to pull the latest code and run it!
- Every time you publish your changes to Github, GitHub will trigger your new Jenkins job.

# Code Packaging automation

Automation Maven test, Compile and Package



# Continuous Delivery Pipeline Using Jenkins

- Fetching the code from GitHub
- Compiling the source code
- Unit testing and generating the JUnit test reports
- Packaging the application into a WAR file and deploying it on the Tomcat server



# Source Code on Github

- <https://github.com/atingupta2005/java-servlet-hello>
- Clone
  - git clone <https://github.com/atingupta2005/java-servlet-hello>
  - cd java-servlet-hello
- Compile app
  - mvn clean install
- Package App
  - mvn clean package

# Step 1 — Compiling the Source Code

- Let's begin by first creating a Freestyle project in Jenkins
- Use Project Name – “Compile”
- When you scroll down you will find an option to add source code repository, select "git" and add the repository URL
- In that repository, there is a pom.xml file which we will use to build our project
- Consider the below screenshot:

# Step 1 — Compiling the Source Code

General **Source Code Management** Build Triggers Build Environment Build Post-build Actions

## Source Code Management

☐ None  
☒ Git

Repositories

Repository URL **https://github.com/saurabh0010/game-of-life.git**

Credentials **- none -** **Add**

**Advanced...**  
**Add Repository**

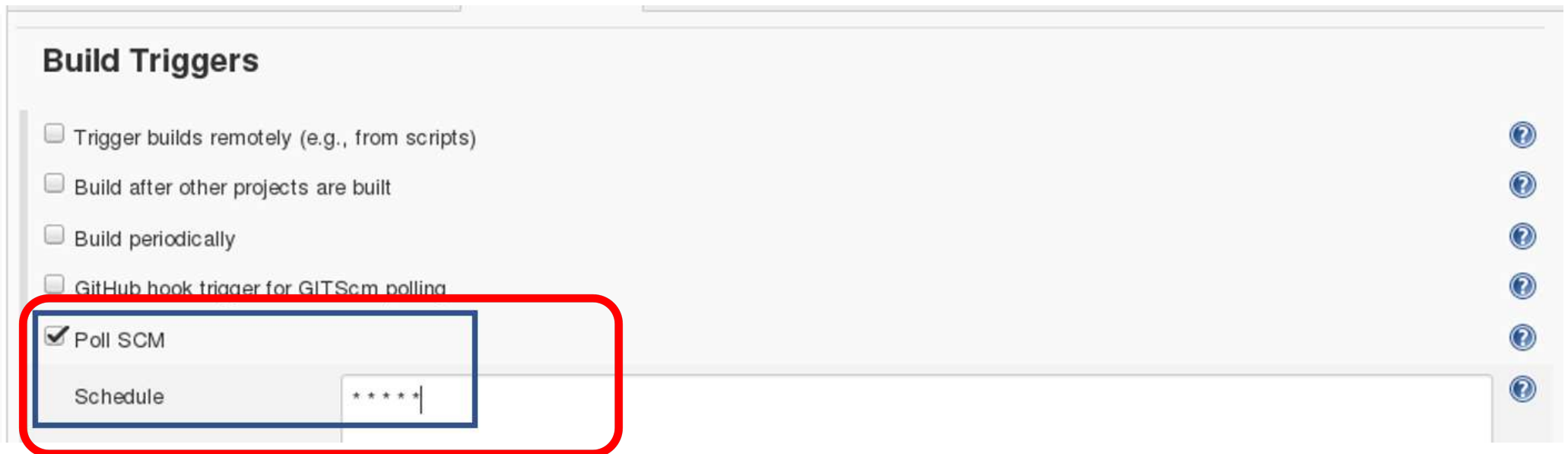
Branches to build

Branch Specifier (blank for 'any') **\*/master**

**Save** **Apply** **Add Branch**

# Step 1 — Compiling the Source Code

- Now we will add a Build Trigger
- Pick the poll SCM option
  - Basically, we will configure Jenkins to poll the GitHub repository after every 5 minutes for changes in the code
- Consider the below screenshot:



**Build Triggers**

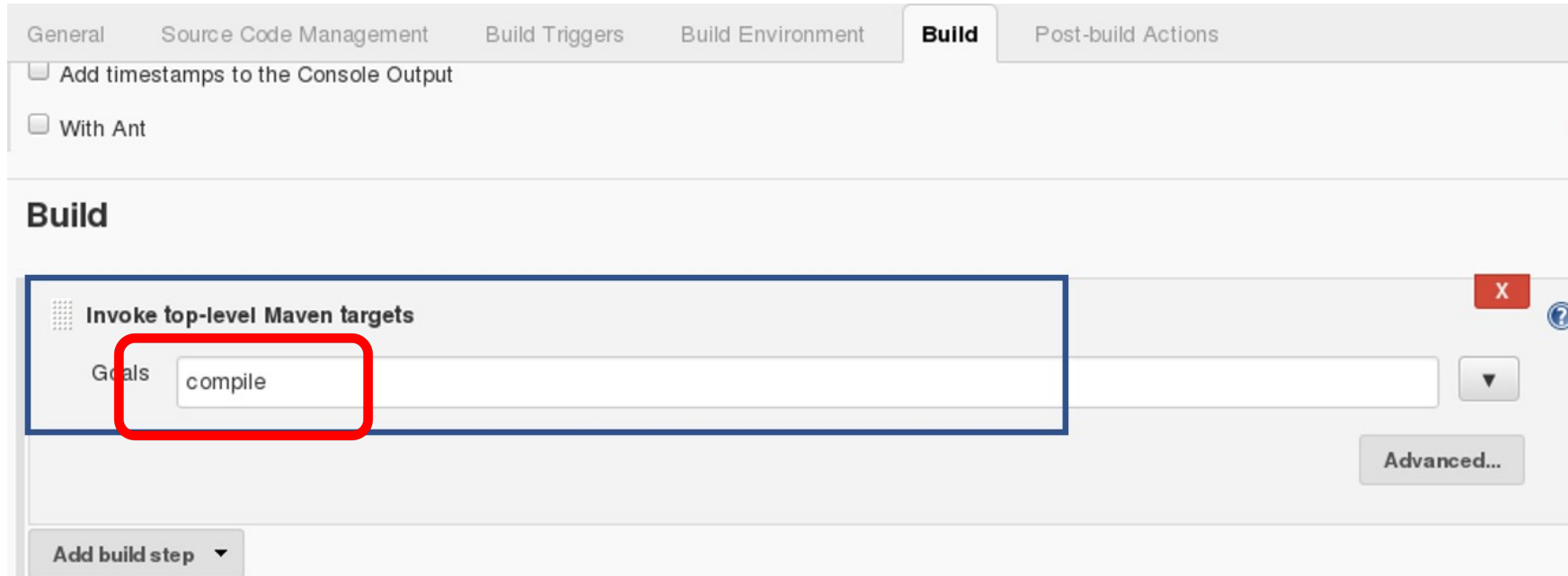
- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ GitHub hook trigger for GITScm polling
- ☒ Poll SCM

Schedule:

# Step 1 — Compiling the Source Code

- In the build tab, click on invoke top level maven targets and type the below command:

- compile



The screenshot shows the Jenkins configuration interface for a build job. The 'Build' tab is selected, and the 'Invoke top-level Maven targets' step is highlighted with a blue border. Within this step, the 'Goals' field contains the text 'compile', which is also highlighted with a red border. Other visible elements include the 'Add build step' button at the bottom left, the 'Advanced...' button at the bottom right, and various checkboxes at the top like 'Add timestamps to the Console Output' and 'With Ant'.

- This will pull source code from the GitHub repository and will also compile it.
- Click on Save and run the project.
- Now, click on the console output to see the result.

## Step 2 — Test the Source Code

- Now we will create one more Freestyle Project for unit testing.
- Project Name - **Test**
- Add the same repository URL in the source code management tab, like we did in the previous job.
- Now, in the "Build Trigger" tab click on the
  - "build after other projects are built".
- In the Build tab, click on invoke top level maven targets and use the below command:
  - `test`

**Build Triggers**

☐ Trigger builds remotely (e.g., from scripts)

☒ Build after other projects are built

Projects to watch: **Compile,**

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

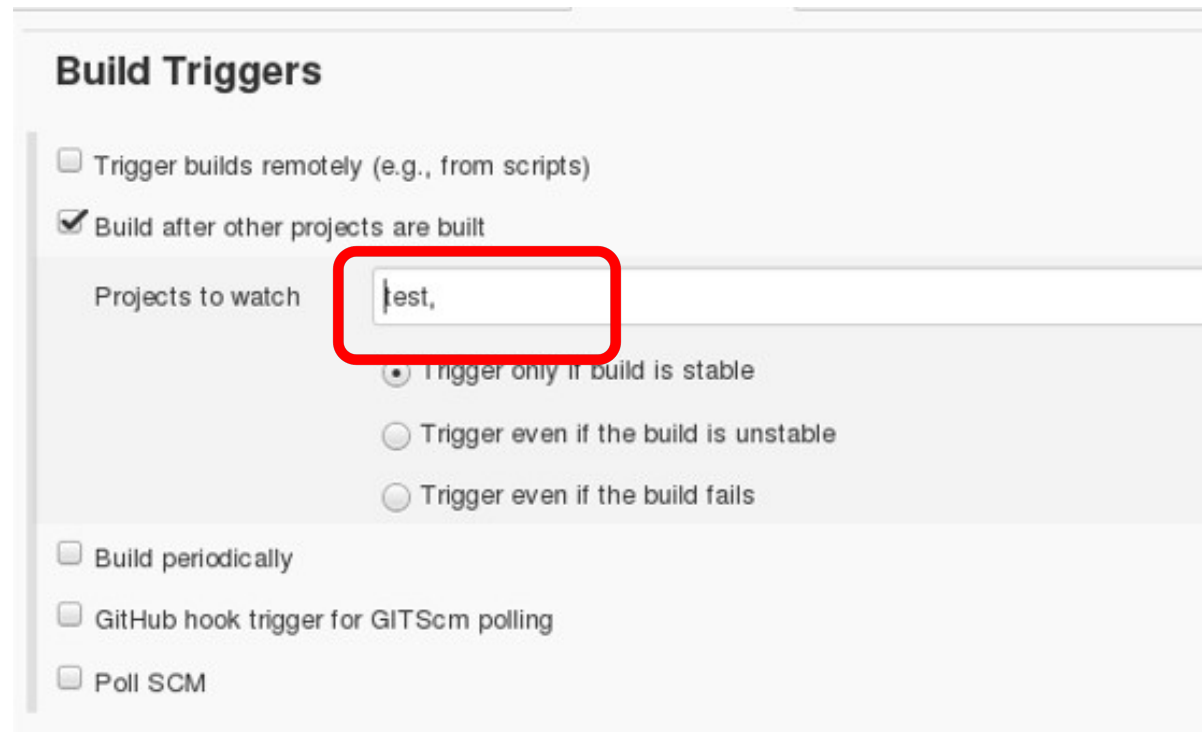
☐ Build periodically

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

## Step 3 — Creating a JAR File and Deploying

- Create one more freestyle project and add the source code repository URL.
- Then in the build trigger tab, select build when other projects are built, consider the below screenshot:
- Project Name – “**Create Jar**”



**Build Triggers**

☐ Trigger builds remotely (e.g., from scripts)

☒ Build after other projects are built

Projects to watch:

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☐ Build periodically

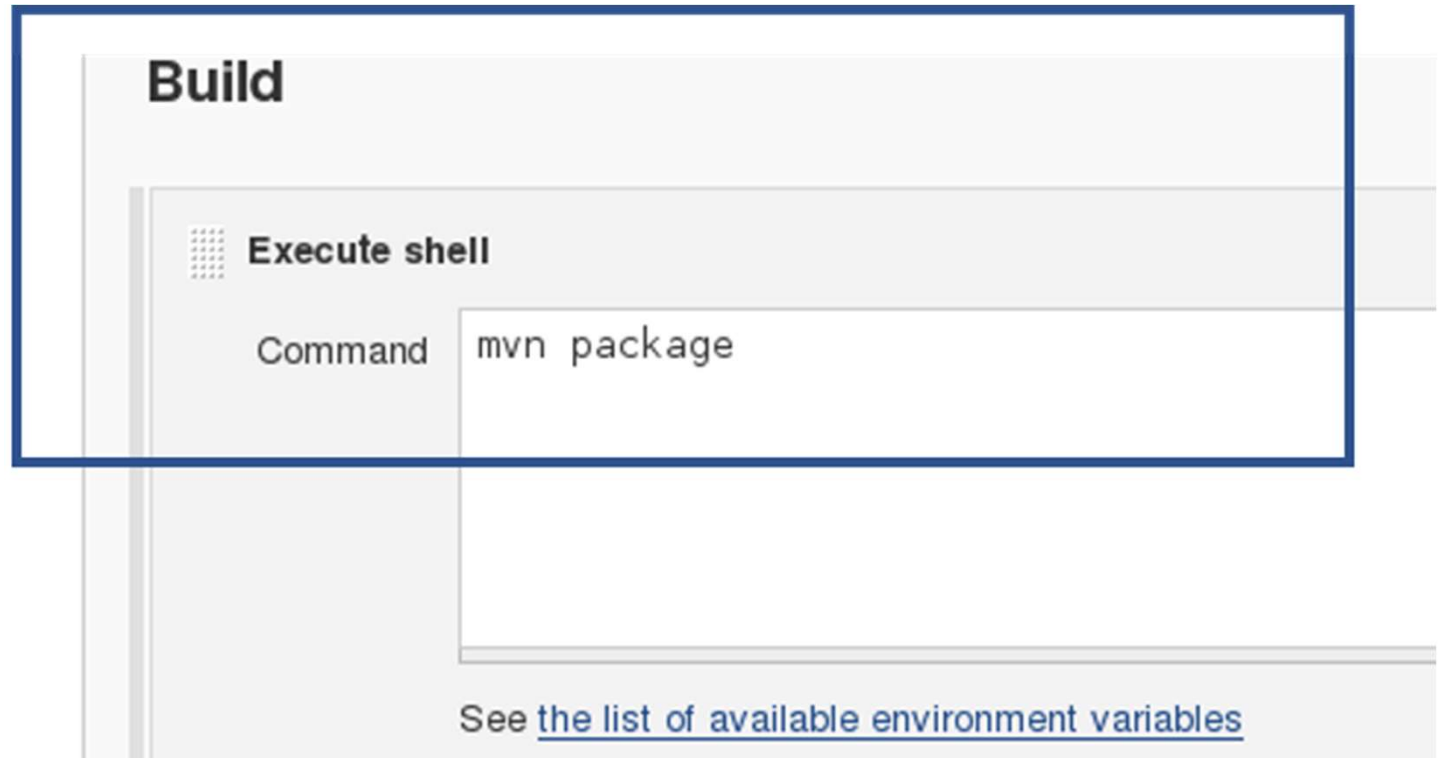
☐ GitHub hook trigger for GITScm polling

☐ Poll SCM



# Step 3 — Creating a JAR File and Deploying

- In the build tab, select shell script. Type the below command to package the application:
  - `mvn package`



# Important URLs

- GitHub projects:
  - <https://github.com/atingupta2005/hello-world-maven>
    - Console Based – To build jar file
  - <https://github.com/atingupta2005/java-servlet-hello>
    - Web Based to build war file

*Thanks*