**LAB PROGRAMS**

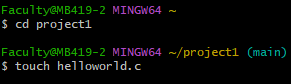
**1a. Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.**

**Solution:**

**Step 1.** Initialize a new Git repository: **$ git init project1**

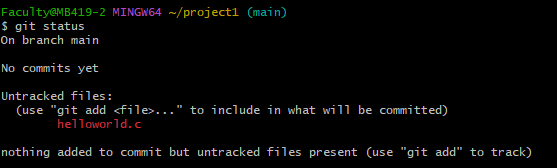
****

**Step 2.** Create a new file: touch/ gedit helloworld.c

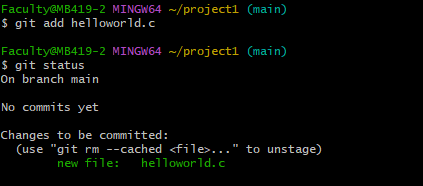


**Step 3.** Add the file to the staging area:

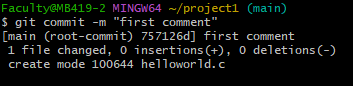
Before adding to staging area**: $ git status**

****

After adding to statging area: **$ git add helloworld.c**

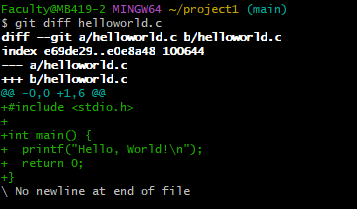
****

**Step 4:** Commit the changes with a message **: $ git commit -m "first comment"**

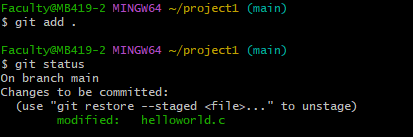


**Others commands:**

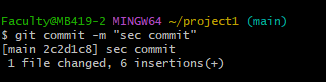
To see changes made in file : **$ git diff helloworld.c**

****

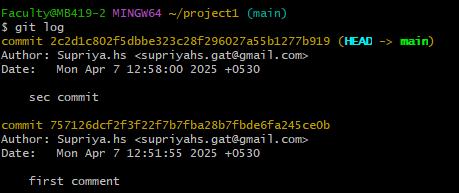
After changes add helloworld.c to repo:



Final commit:



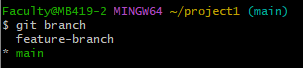
Check log on git**: $git log**

****

**1b. Creating and Managing Branches Create a new branch named “feature-branch.” Switch to the “master” branch. Merge the “feature-branch” into “master.”**

**Solution:**

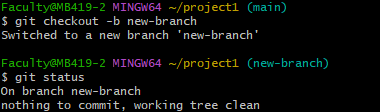
**Step 1:** Create a new branch named “feature-branch**”: $ git branch feature-branch**



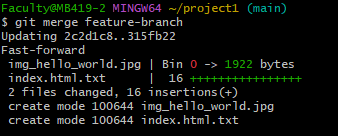
Alternatively, you can create and switch to the new branch in one step : **$ git checkout -b feature-branch**



**Note:** Using the **-b** option on checkout will create a new branch, and move to it, if it does not exist.



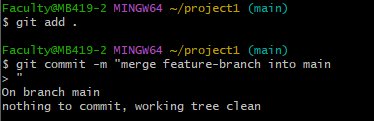
**Step 2:** Switch to the “master” branch **:$ git checkout master**

**Step 3:** Merge “feature-branch” into “master”: **$ git merge feature-branch**

**Step 4: Resolve any conflicts (if needed) and commit the merge:** If there are conflicts, Git will mark the conflicted files. Open each conflicted file, resolve the conflicts, and then.

**$ git add**

**$ git commit -m "Merge feature-branch into master”**



Now, the changes from “feature-branch” are merged into the “master” branch. If you no longer need the “feature-branch,” you can delete it. $ git branch -d feature-branch.



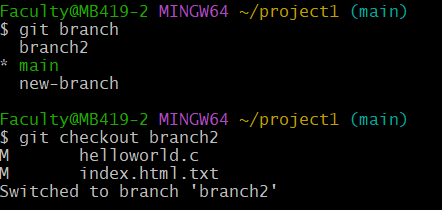
This assumes that the changes in “feature-branch” do not conflict with changes in the “master” branch. If conflicts arise during the merge.



**2a. Write the commands to stash your changes, switch branches, and then apply the stashed changes.**

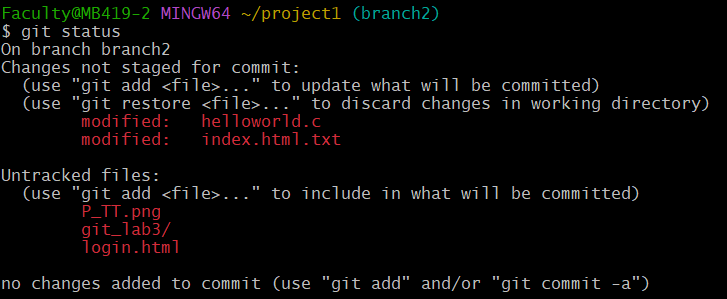
**Solution:**

**Step 1.** Check your current branch: **$ git branch**

****

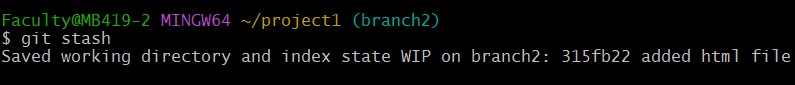
**Step 2.** Make some changes to files (uncommitted) You edited a file called Login.html:

# $ git status

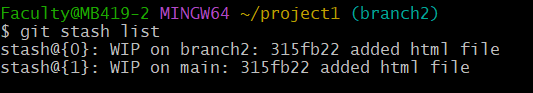
****

**Step 3.** Stash your changes: **$git stash save "Your stash message"**

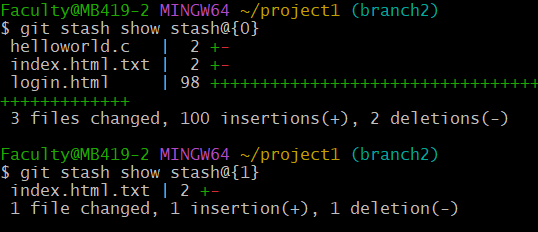
This command will save your local changes in a temporary area, allowing you to switch branches without committing the changes.



# List the Git Stash Entries: $git stash list

****

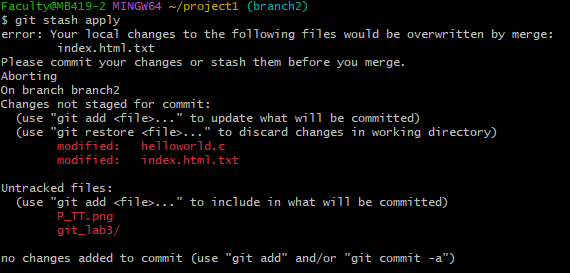
Show the changes recorded in the stash: **$ git stash show [stash\_ID] Ex: $git stash show stash@{0}**



**Step 2.** Switch to another branch: **$ git checkout your-desired-branch Step 3**. Apply the stashed changes: **$git stash apply**

If you have multiple stashes and want to apply a specific stash, you can use:

# $git stash apply stash@{1}

****

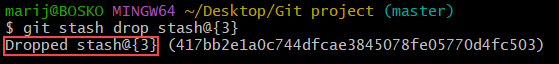
After applying the stash, your changes are reapplied to the working directory.

**Step 4**. Remove the applied stash (optional):

If you no longer need the stash after applying it, you can remove it**: $git stash drop**

****

To remove a specific stash: **$git stash drop stash@{3}**

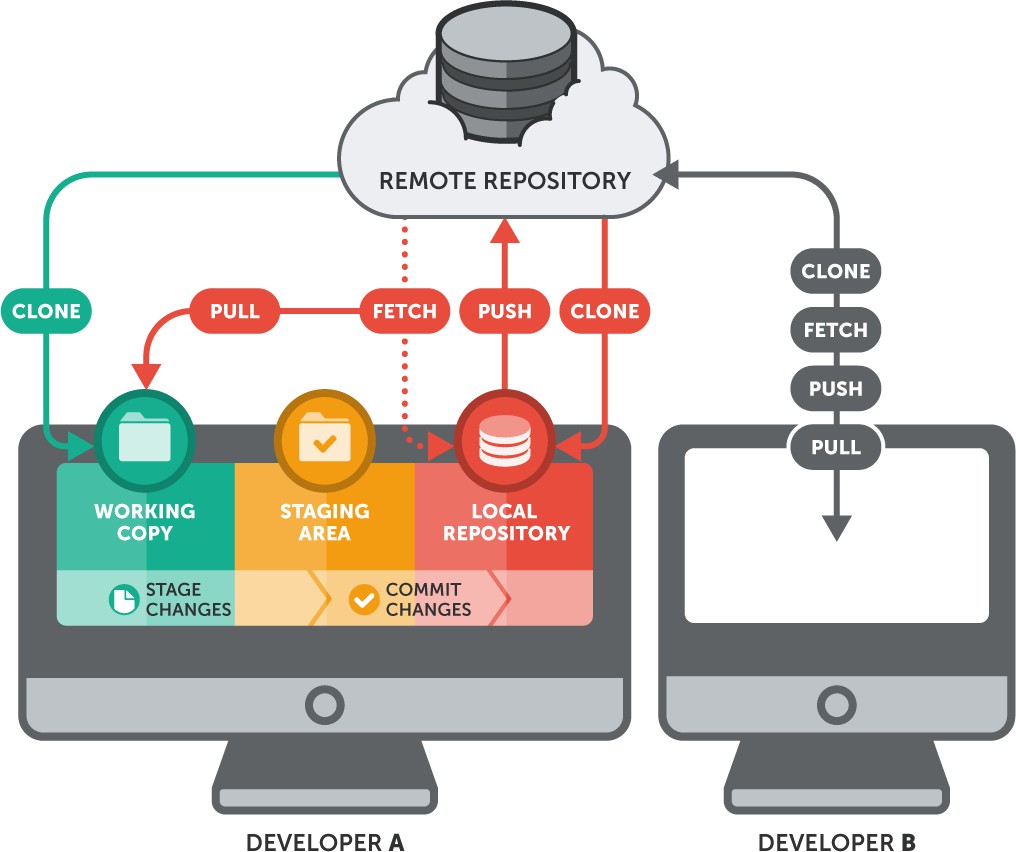
****

**Drop All Git Stashes: $**git stash clear

**Warning:** Running the git stash clear command schedules all entries for deletion. Once deleted, there is no way of recovering the entries.

**2b. Clone a remote Git repository to your local machine. Solution:**

**The git remote** command lets you create, view, and delete connections to other repositories. Remote connections are more like bookmarks rather than direct links into other repositories.

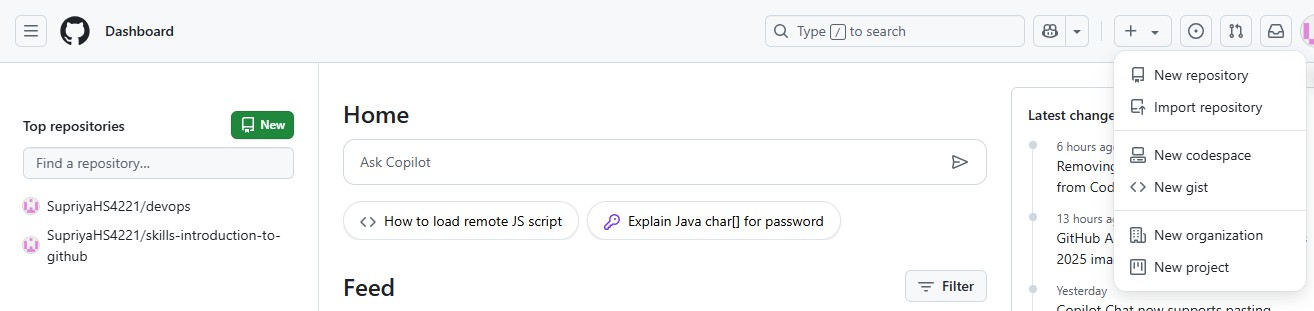


To clone a remote Git repository to your local machine, you can use the git clone command. Here’s the general syntax: **$git clone <repository\_url>**

**Create a GitHub Account:** Open your browser and go to: [https://github.com](https://github.com/) Create GitHub account with necessary details.

# Create a New GitHub Repository:

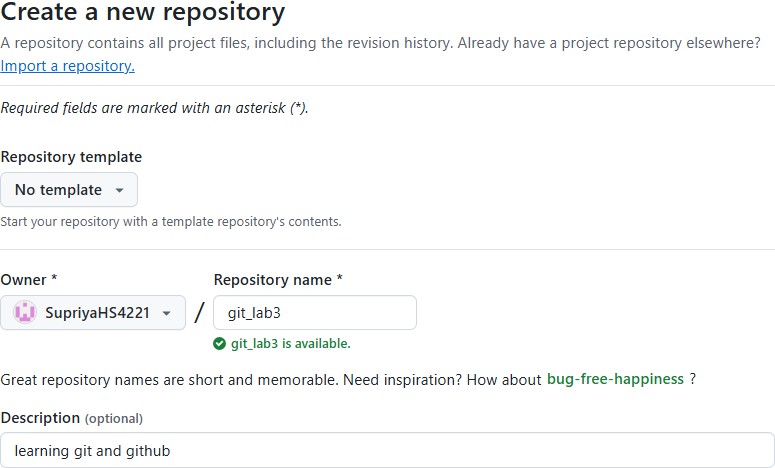
* Click the "+" icon (top-right) → Select "New repository".

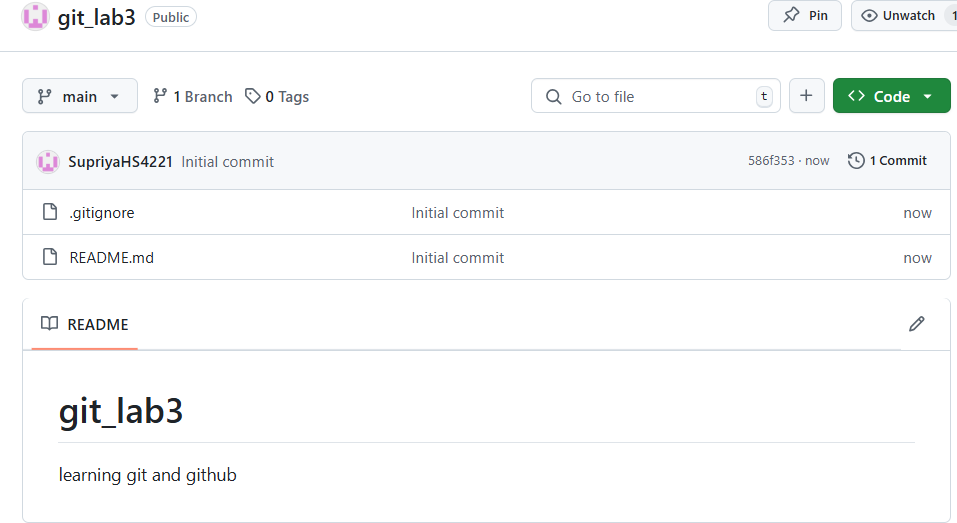


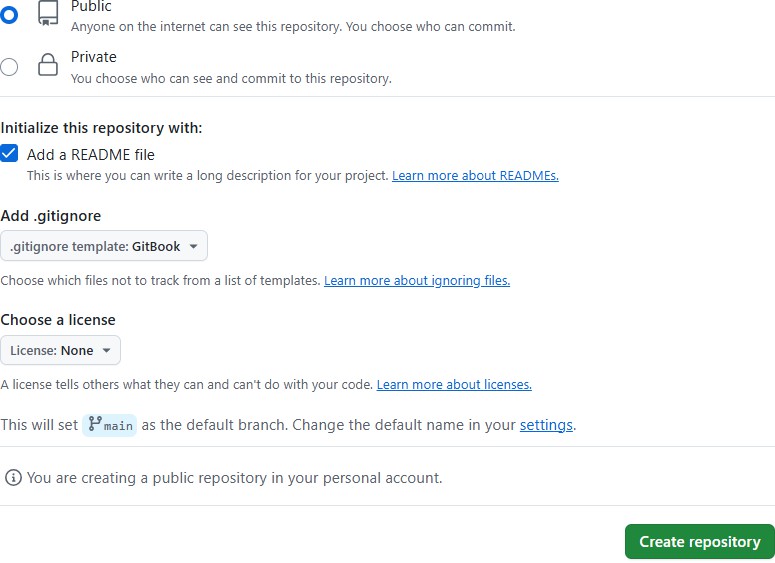
* Enter a repository name (e.g., my-project).
* Add a **description**.
* Choose **Public** or **Private** visibility.

# Check "Initialize this repository with a README".

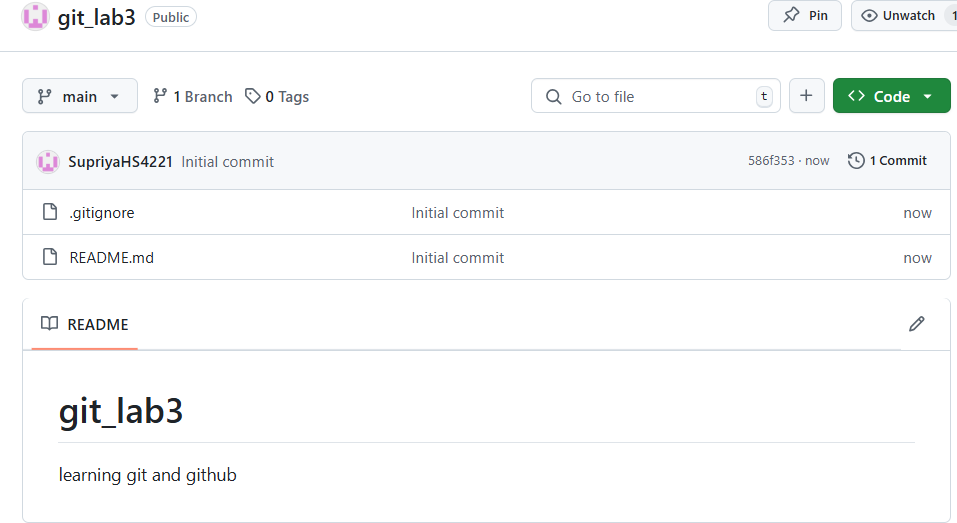
* Click the green **"Create repository"** button.





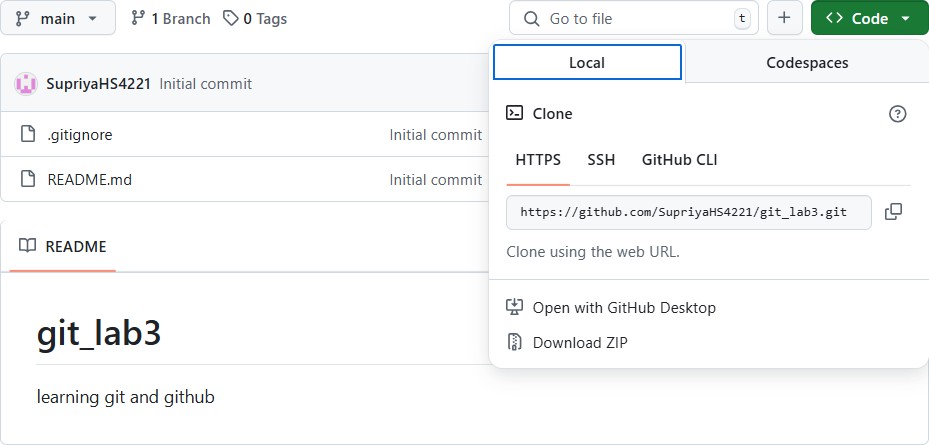


Finally, your repository will look like below:

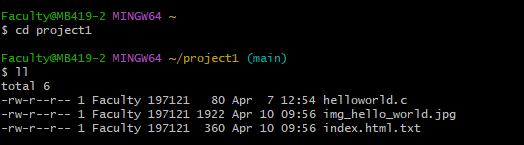


**Git Clone over HTTPS using the Command Line:**

* On the left sidebar, select Search or go to and find the project you want to clone.
* On the project’s overview page, in the upper-right corner, select Code, then copy the URL for Clone with HTTPS.



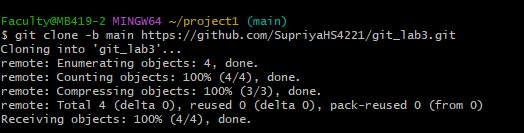
**Open a terminal and go to the directory where you want to clone the files:**



Run the following command. Git automatically creates a folder with the repository name and downloads the files there.

# $git clone <copied URL>

Replace <repository\_url> with the actual URL of the Git repository you want to clone. For example: **$ git clone** [**https://github.com/example/repo.git**](https://github.com/example/repo.git)

****

This command will create a new directory with the name of the repository and download all the files from the remote repository into that directory.

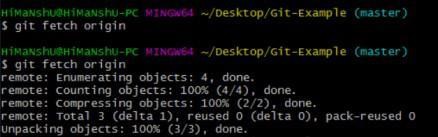
After running the git clone command, you’ll have a local copy of the remote repository on your machine, and you can start working with the code.

**3a. Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.**

**Solution:**

**Fetch the latest changes from the remote repository: $ git fetch**

Use **$git fetch** to retrieve new work done by other people. Fetching from a repository grabs all the new remote-tracking branches and tags without merging those changes into your own branches.



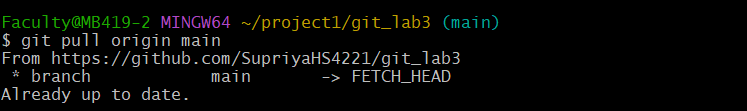
# Pulling changes from a remote repository:

**$git pull** is a convenient shortcut for completing both git fetch and git merge in the same command:

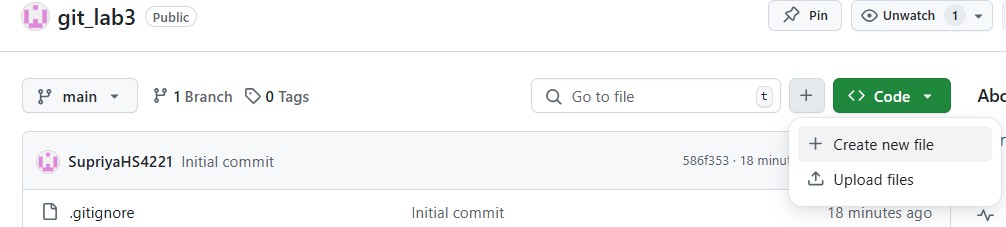
# $git pull <remote> <branch>

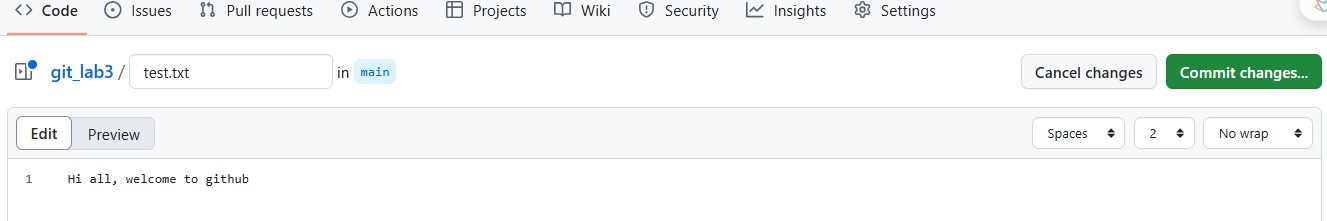
<remote> → Usually origin (the default remote)

<branch> → Usually main or master

**Ex: $git pull origin main**

**Fetch and Rebase in Git:**

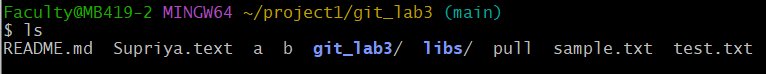
**Step 1: Creating new file in github :**

Write some info in the file

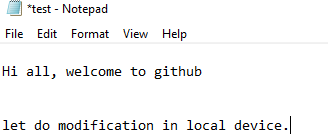
Now something got updated in remote repository.

**Step 2:** Open Terminal / Git Bash and Check Your Current Branch: **$git branch Step 3**: Let’s do git pull to fetch changes made in remote repo to local repo:

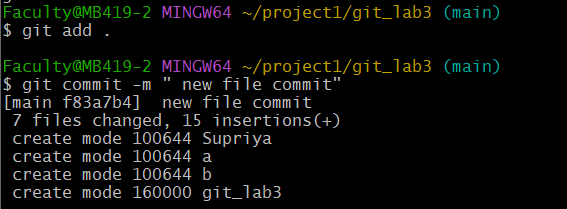
**Step 4**: let’s check for the update local repo with ls Command

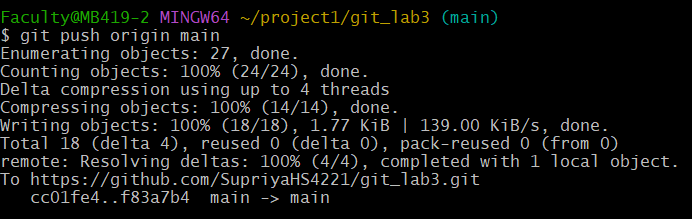


**Step 5**: Let’s do modify test.txt file locally

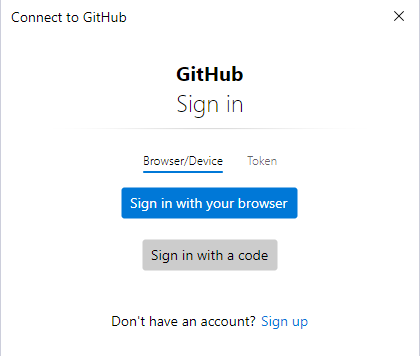


**Step 6:** Commit the changed file



**Step 7:** Push content to remote repository : **$git push**

Kindly provide the authentication needed.

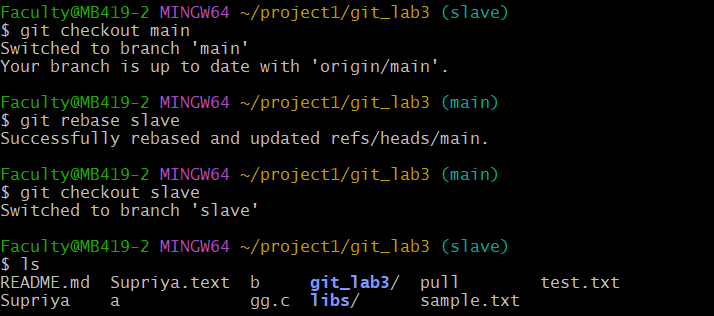


Now the changes made in local will be reflecting in GitHub.

# Rebase your local branch onto the updated remote branch:

Assuming you are currently on the branch you want to update (replace your-branch with the actual name of your branch):

# $git rebase origin your-branch

This command applies your local commits on top of the changes fetched from the remote branch.

# Push the rebased branch to the remote repository:

After successfully rebasing your local branch, you may need to force-push the changes to the remote repository**: $ git push origin your-branch --force**

Be cautious with force-pushing, especially if others are working with the same branch, as it rewrites the commit history.

Now, your local branch is rebased onto the updated remote branch. Keep in mind that force-pushing should be done with caution, especially on shared branches, to avoid disrupting collaborative work.

**3b. Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.**

**Solution:**

To merge “feature-branch” into “master” and provide a custom commit message, you can use the following command: **$ git merge feature-branch -m "Your custom commit message"**

Replace “Your custom commit message” with the actual message you want to use for the merge commit. This command performs the merge and creates a new commit on the “master” branch with the specified message. If there are no conflicts, Git will complete the merge automatically.

If conflicts occur during the merge, Git will pause and prompt you to resolve the conflicts manually. After resolving conflicts, you can continue the merge process with:

# $ git merge –continue

Alternatively, you can use an interactive merge to modify the commit message before finalizing the merge: **$ git merge feature-branch --no-ff –e**

This opens an editor where you can edit the commit message before completing the merge. Again, replace “feature-branch” with the name of your actual feature branch.

**4a. Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.**

**Solution:**

**There are two types of tags:**

* **Lightweight**: Used internally, Lightweight tags only point to specific commits and contain no extra information other than the tag name.

Create a lightweight tag using the following syntax:

**$git tag [tag\_name]**

* **Annotated:** Stored as full objects in the Git database, Annotated tags contain metadata, and they are used to describe a release without making a release commit.

Create an annotated tag using the following syntax:

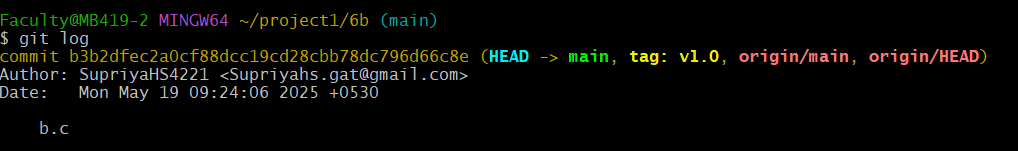
**$git tag -a [tag\_name] -m [message]**

To create a lightweight Git tag named “v1.0” for a specific commit in your local repository, you can use the following command:

**$git tag v1.0 <commit\_hash>**

Replace <commit\_hash> with the actual hash of the commit for which you want to create the tag. For example, if you want to tag the latest commit, you can use the following:

**$git tag v1.0 HEAD**

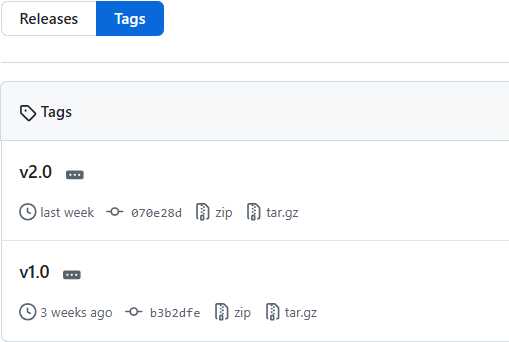
****

This creates a lightweight tag pointing to the specified commit. Lightweight tags are simply pointers to specific commits and contain only the commit checksum.

If you want to push the tag to a remote repository, you can use:

**$git push origin v1.0**

This command pushes the tag named “v1.0” to to the remote repository. Keep in mind that Git tags, by default, are not automatically pushed to remotes, so you need to explicitly push them if needed.



**4b. Write the command to cherry-pick a range of commits from "source-branch" to the current branch**

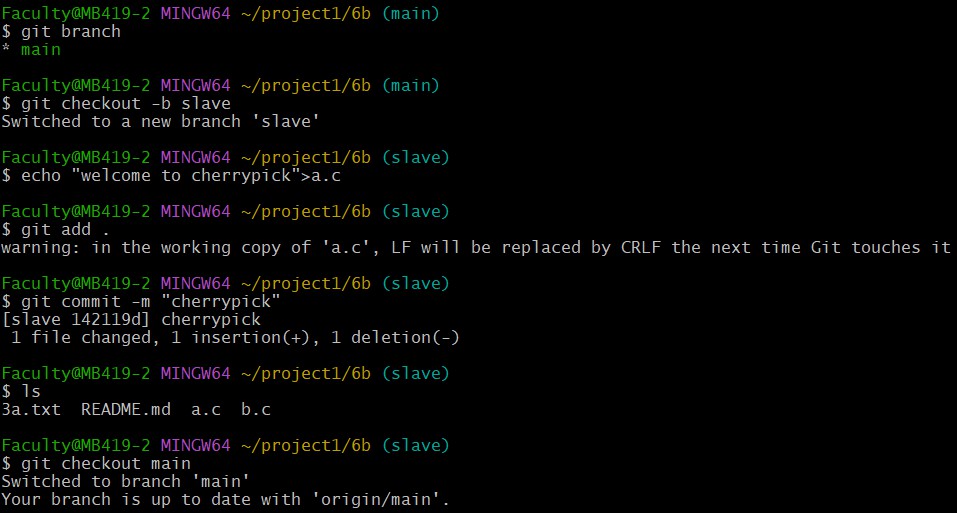
**Solution:**

To cherry-pick a range of commits from “source-branch” to the current branch, you can use the following command:

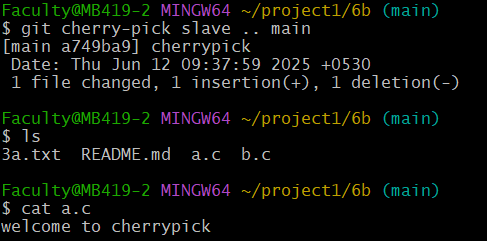
**$git cherry-pick <start-commit>^..<end-commit>**

Replace <start-commit> and <end-commit> with the commit hashes or references that define the range of commits you want to cherry-pick. The ^ (caret) symbol is used to exclude the starting commit itself from the range.

**Steps 1:** View all branches**: $git branch**

**Step 2:** Switch to source-branch and note commit hashes**: $git checkout –b <source-branch> Step 3:** Switch to the branch where you want to apply commits**:** $**git checkout main**

# Step 4: git cherry-pick slave .. main

****

If you encounter issues and need to abort the cherry-pick operation, you can use:

# $git cherry-pick --abort

Remember that cherry-picking introduces new commits based on the changes from the source branch, so conflicts may arise, and manual intervention might be required.

**5a. Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?**

**Solution:**

**Step 1:** To view the details of a specific commit, including the author, date, and commit message, you can use the following Git command:

# $git show <commit-id>

Replace <commit-id> with the actual commit hash or commit reference of the commit you want to inspect.

For example:

# $git show abc123

This command will display detailed information about the specified commit, including the author, date, commit message, and the changes introduced by that commit.

**5b. Write the command to list all commits made by the author "JohnDoe" between "2023-01- 01" and "2023-12-31."**

**Solution:**

To list all commits made by the author “JohnDoe” between “2023-01-01” and “2023-12- 31,” you can use the following git log command with the --author and --since / --until options:

# $git log --author=“ “ --since="2025-04-01" --until="2025-5-31"

****

**6a. Write the command to display the last five commits in the repository's history also undo the changes introduced by the commit with the ID "abc123".**

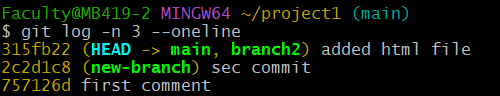
**Solution:**

To display the last five commits in the repository’s history, you can use the following git log command with the -n option:

# $git log -n 5

Adjust the number after the -n option if you want to see a different number of commits. If you want a more concise output, you can use the --oneline option:

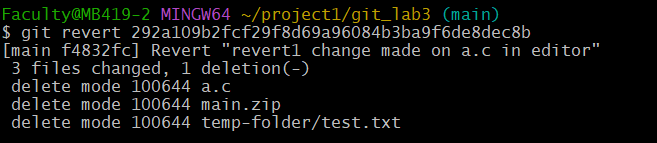
# $git log -n 5 –oneline

****

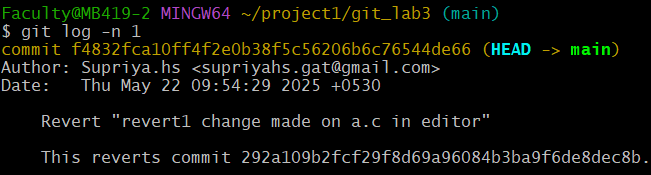
This provides a one-line summary for each commit, including the abbreviated commit hash and the commit message.

To undo the changes introduced by a specific commit with the ID “abc123,” you can use the git revert command. The git revert command creates a new commit that undoes the changes made in a previous commit. Here’s the command:

# $git revert abc123

****

After running this command, Git will open a **text editor** for you to provide a **commit message for the new revert commit.**



**6b. Write the command to Define custom Git aliases for common commands like git st for git status, git lg for log graph**

**Solution:**

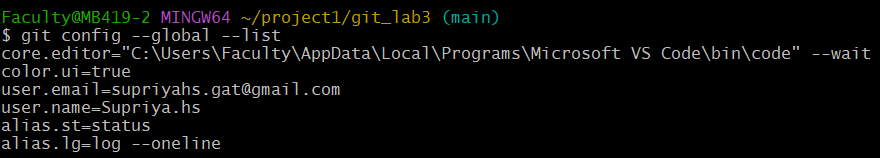
**Step 1: Open Git Bash or Terminal:** You’ll be using the **$git config** command to define your aliases

1. git st → shortcut for git status: **$git config --global alias.st status**
2. b) git lg → shortcut for log with graph view **$git config --global alias.lg "log --oneline**

**--graph"**

**Verify the aliases: $git config --global --list**

****



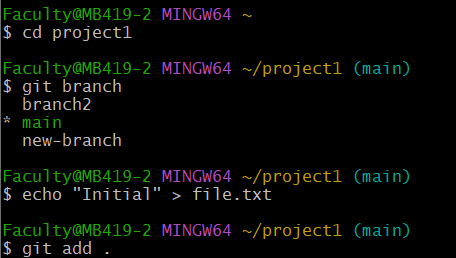
**7a. Simulate reverting entire merge commit using** git revert -m**. Solution:**

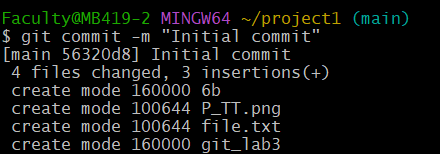
**Step 1: Create and Merge a Branch:**

git init

echo "Initial" > file.txt git add .

git commit -m "Initial commit"



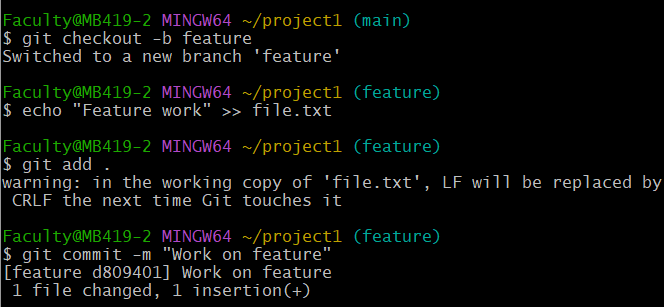


# # Create feature branch

git checkout -b feature

echo "Feature work" >> file.txt git add .

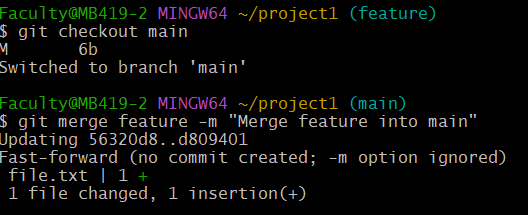
git commit -m "Work on feature"



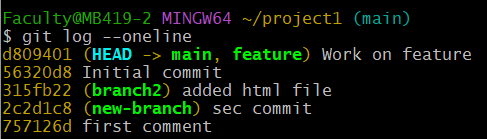
# # Switch back and merge

git checkout main

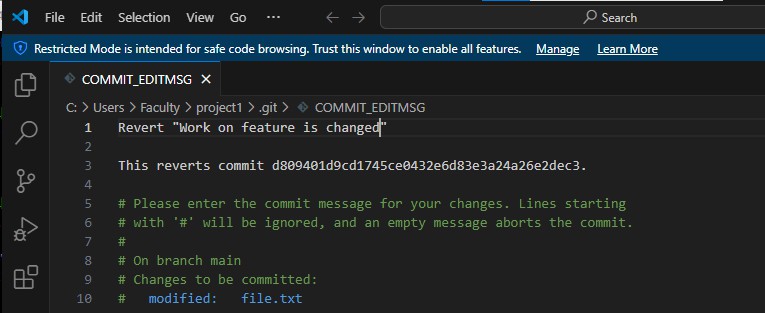
git merge feature -m "Merge feature into main"



**Step 2: Get the Merge Commit Hash: $git log --oneline**

****

**Step 3: Revert the Merge Commit:** Use -m to specify the mainline parent: **$git revert - m 1 f4d3c6b**

****

# A

-m 1 means you are keeping parent 1 (usually the branch you merged into, i.e., main).

# 7b. Accidentally delete a branch or reset a commit, then use git reflog to recover the lost commit.

**Solution:**

**git reflog** records **every move of HEAD** (including commits, checkouts, resets, rebases). It’s your undo safety net—even after deletes and resets.

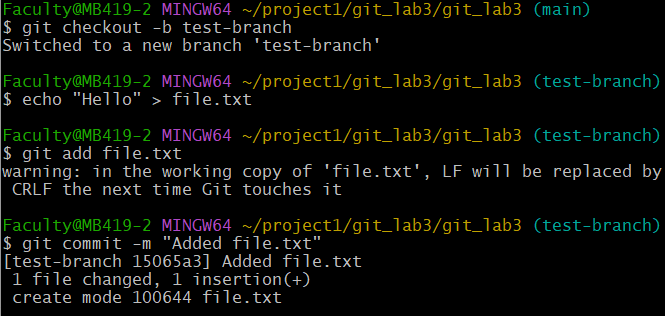
# Step-by-step to recover it:

* 1. Create and switch to a branch**:**

**$git checkout -b test-branch**

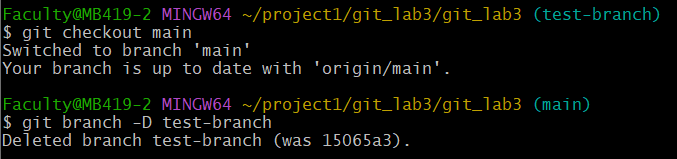
**$echo "Hello" > file.txt**

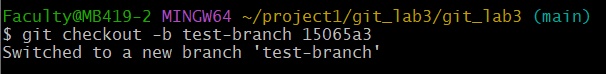
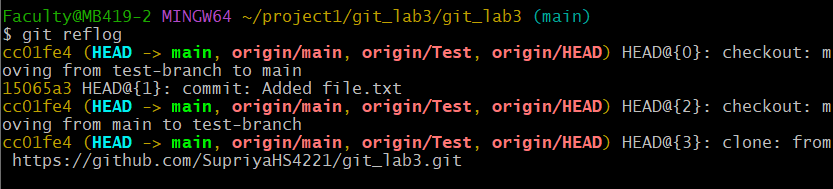
**$git add file.txt**

**$git commit -m "Added file.txt"**

* 1. Switch to another branch and delete test-branch**:**

# $git checkout main

**$git branch -D test-branch**

* 1. Recover with git reflog**: $git reflog**
  2. ****Restore the branch**: $git checkout -b test-branch 15065a3**

# 8a. Add several untracked files to the repo and then use git clean to remove them safely.

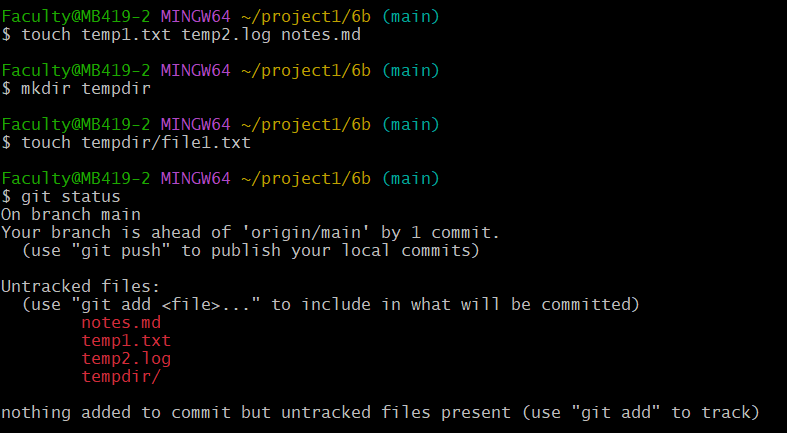
**Solution:**

1. **Add several untracked files:** Create a few untracked files for demonstration:

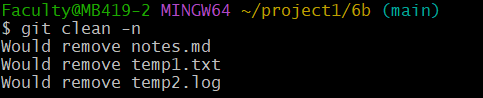
**$touch temp1.txt temp2.log notes.md**

**$mkdir tempdir**

**$touch tempdir/file1.txt**

****Verify they're untracked: **$git status**

Preview what will be removed using **git clean –n** This is a safe dry-run: **$git clean –n**



# 8b. Create a .zip or .tar archive of a specific branch or commit using git archive. Solution:

**Create a .zip Archive of a Branch:**

**Step1:** Open Terminal or Command Prompt Navigate to your Git repository folder:

**cd path\to\your\repository**

**Step 2:** Run the Archive Command: $**git archive --format=zip --output=main.zip main**

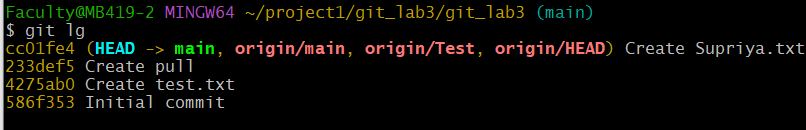
* --format=zip: sets the archive format.
* --output=main.zip: names the output file.
* main: is the name of the branch.

**Create a .tar Archive of a Branch**

**Step 1:** Open Terminal or Command Prompt: **cd path\to\your\repository**

**Step 2:** Run the Archive Command: **$git archive --format=tar --output=main.tar main**

**Create Archive of a Specific Commit**

**Step 1:** Find the Commit Hash: **$git log --oneline**

**Step 2:** Create a .zip Archive of That Commit: **$git archive --format=zip --output=commit.zip abc1234**



****

1. **Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use.**

**Solution:**

**What is Jenkins?**

Jenkins is an open-source automation server widely used in the field of Continuous Integration (CI) and Continuous Delivery (CD). It allows developers to automate the building, testing, and deployment of software projects, making the development process more efficient and reliable.

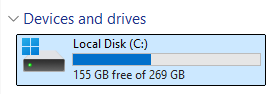
# Key features of Jenkins:

* + **CI/CD:** Jenkins supports Continuous Integration and Continuous Deployment, allowing developers to integrate code changes frequently and automate the deployment of applications.
  + **Plugins:** Jenkins has a vast library of plugins that can extend its capabilities. These plugins integrate Jenkins with version control systems (like Git), build tools (like Maven or Gradle), testing frameworks, deployment tools, and much more.
  + **Pipeline as Code:** Jenkins allows the creation of pipelines using Groovy-based DSL scripts or YAML files, enabling version-controlled and repeatable pipelines.
  + **Cross-platform:** Jenkins can run on various platforms such as Windows, Linux, macOS, and others.

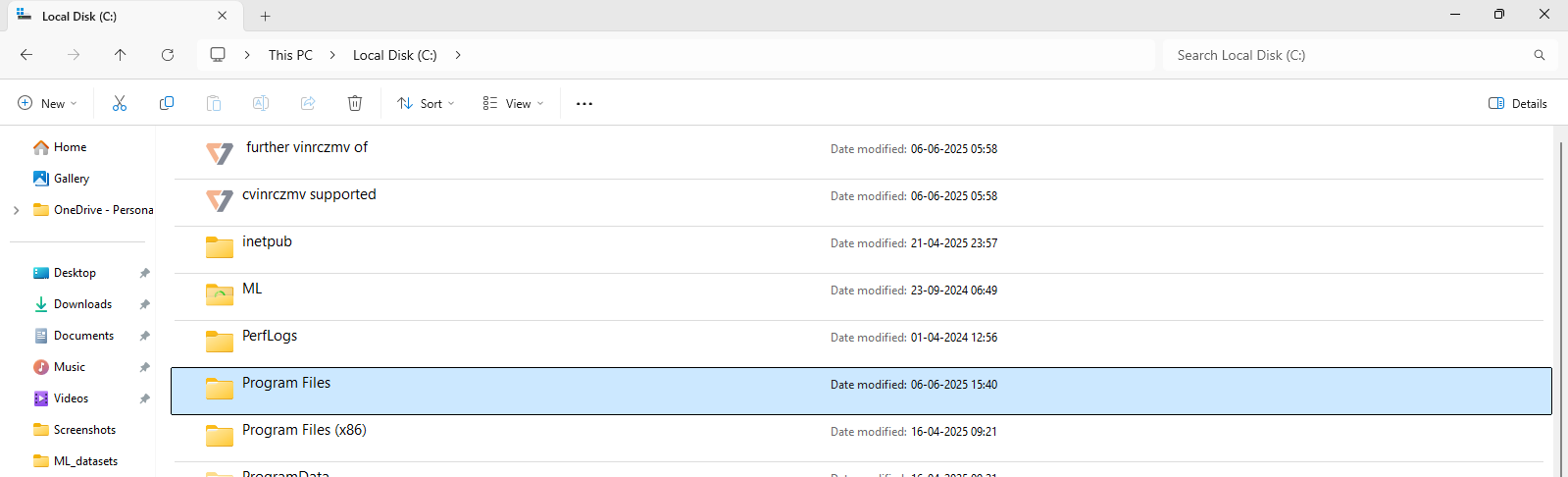
# Installing Jenkins

Jenkins can be installed on local machines, on a cloud environment, or even in containers.

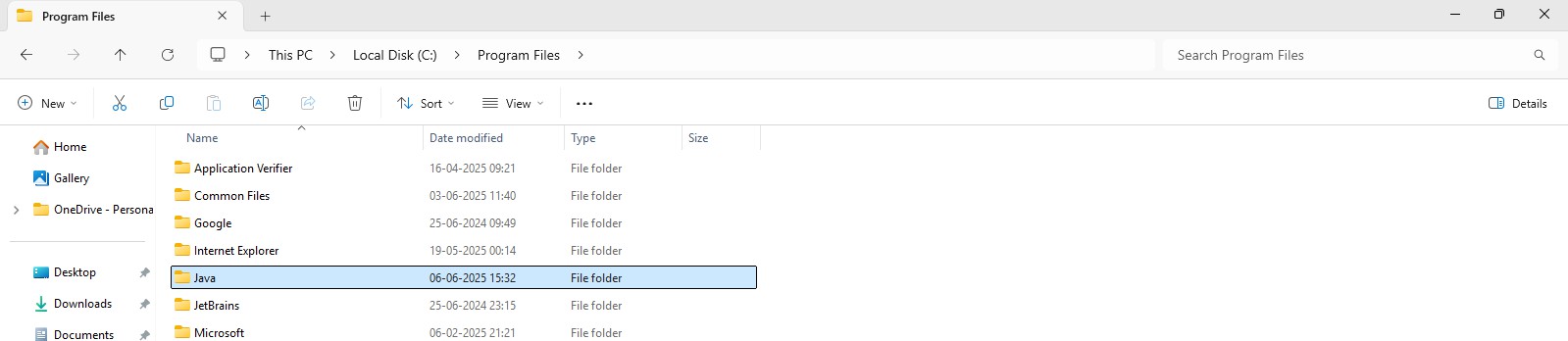
**Installation of Jenkins on Windows: Step 1:** check for program files in C: drive



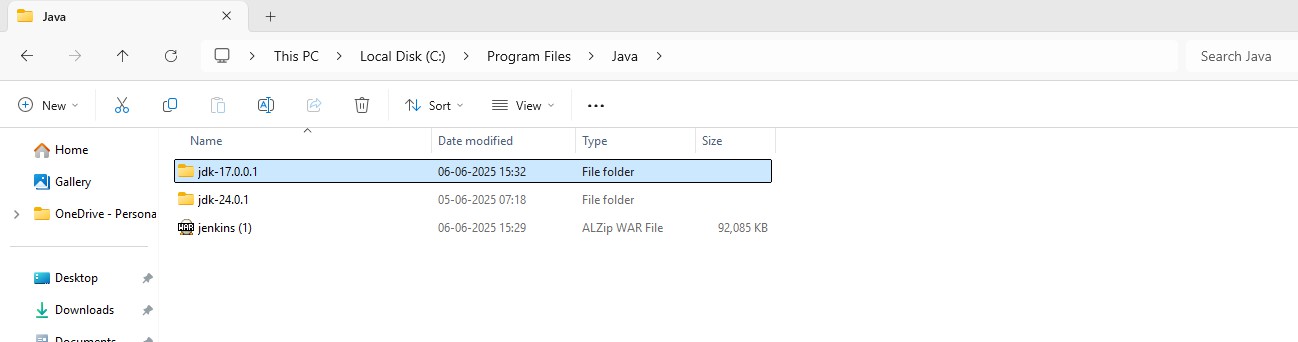
# Step 2:

****

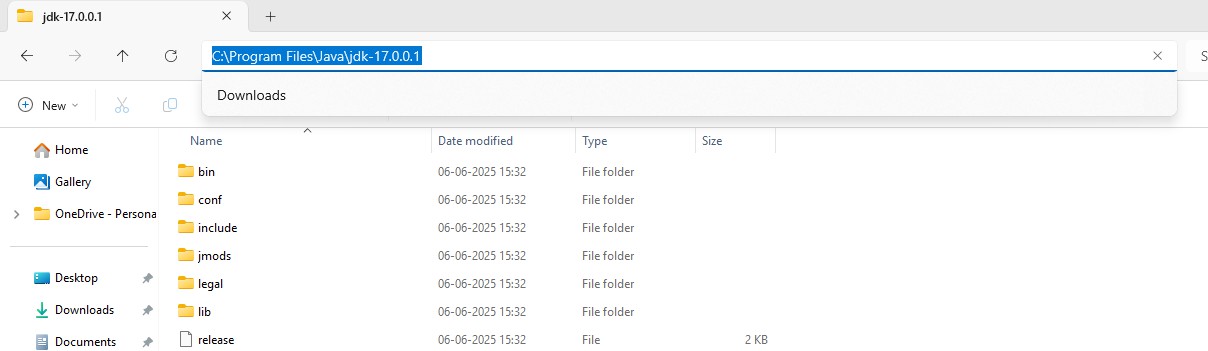
**Step 3:** Check for Java Folder



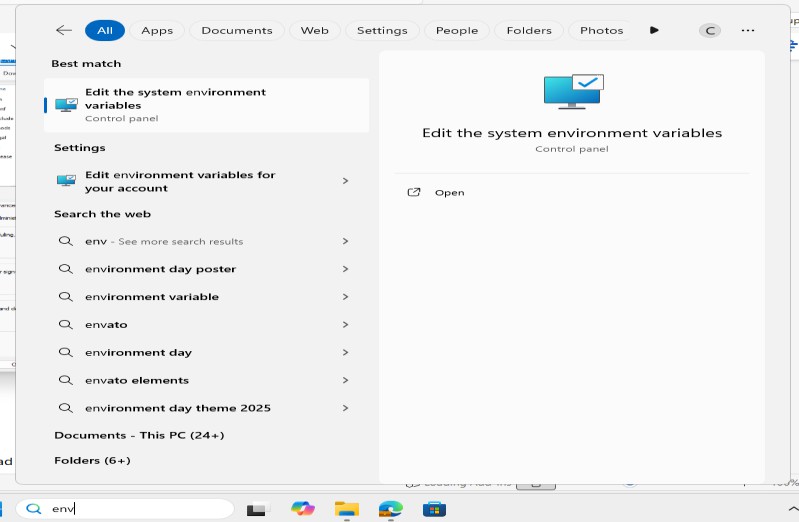
**Step 4:** Extract the Jdk 17 zip file here



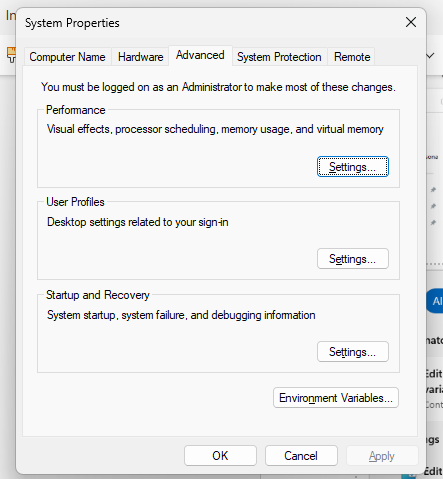
**Step 5:** copy the path of Jdk file



**Step 6:** Open Environment Variables: Press Windows + S, type Environment Variables, and click on Edit the system environment variables.

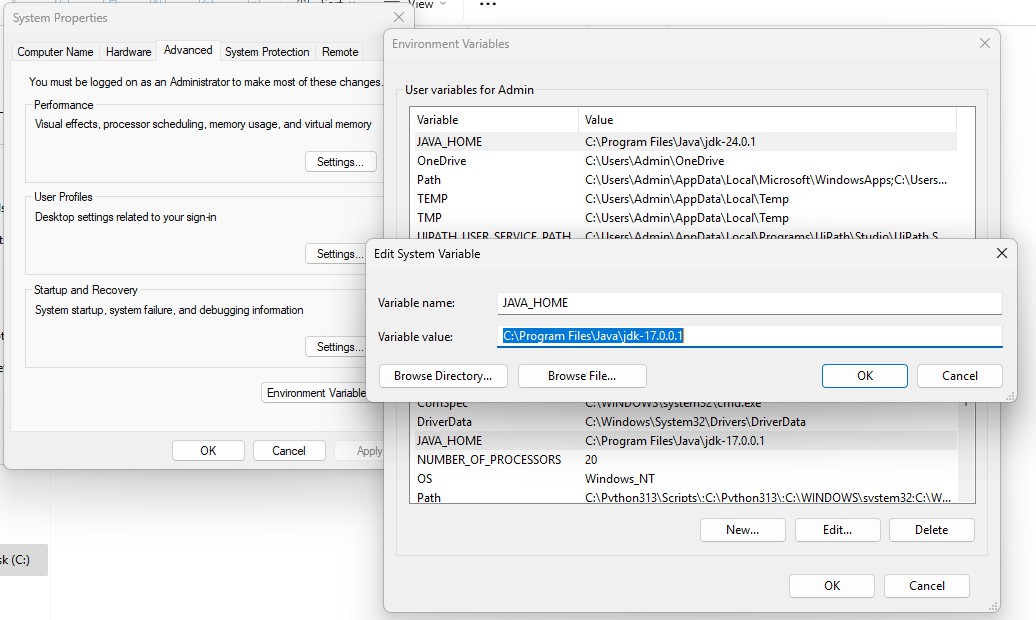
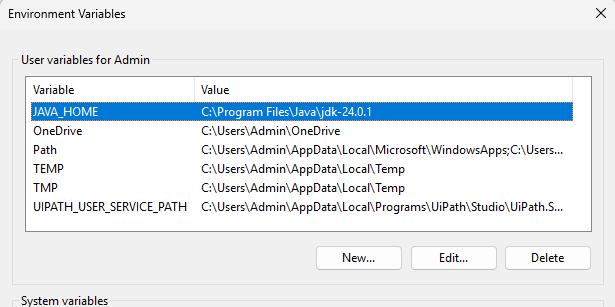


**Step 7:** In the System Properties window, click on Environment Variables.

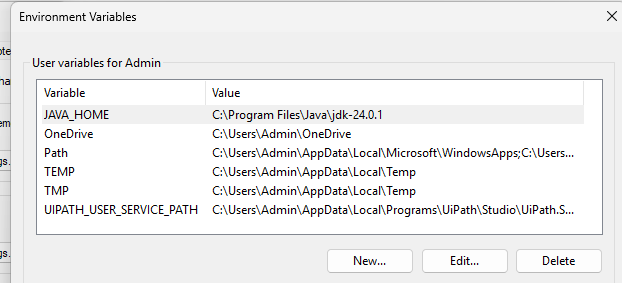


**Step 8:** Set JAVA\_HOME->Under System variables, click New. Variable Name**: JAVA\_HOME**

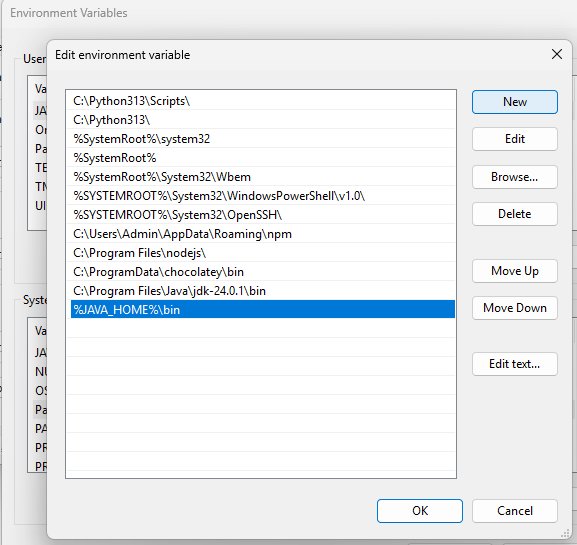
Variable Value: path to your JDK folder (e.g., C:\Program Files\Java\jdk-17)->Click OK.



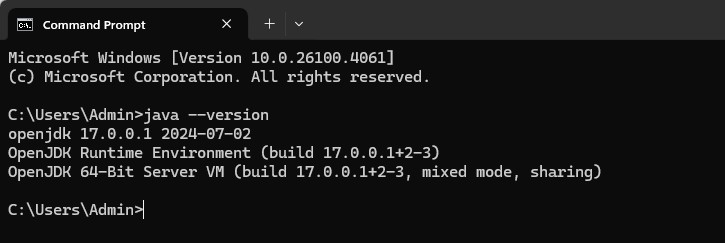
**Step 9:** Path variable->click on edit



**Step 10:** click on new-> write %JAVA\_HOME%\bin->click ok



**Step 11:** Go to to command prompt and check for java --version

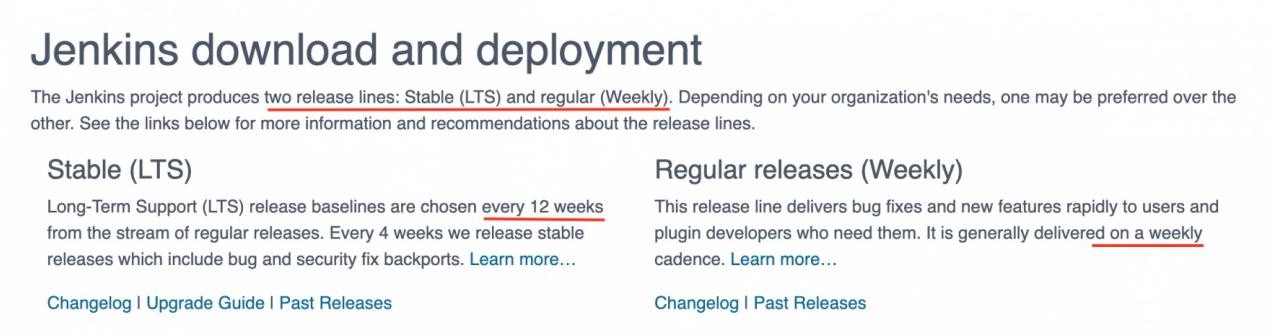


# Download Jenkins Installer for Windows

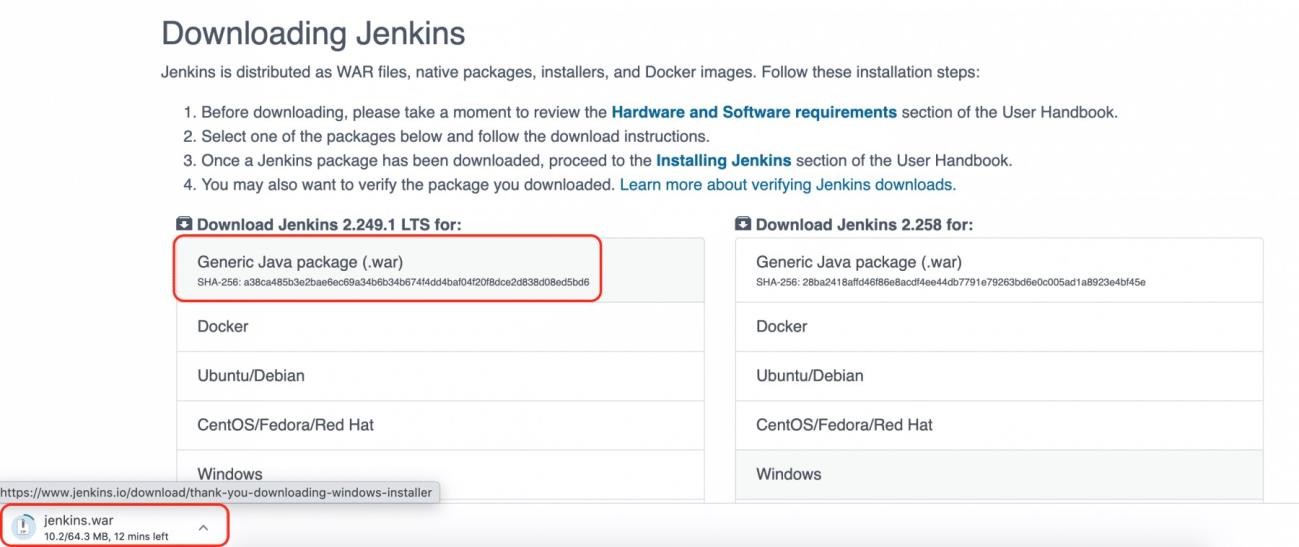
Step 1: Firstly, go to the official Jenkins website and click on the Download button.



**Step 2:** Secondly, after clicking on the Download button, we will be redirected to the download page. Additionally, here we can see all the download related information, as shown below:

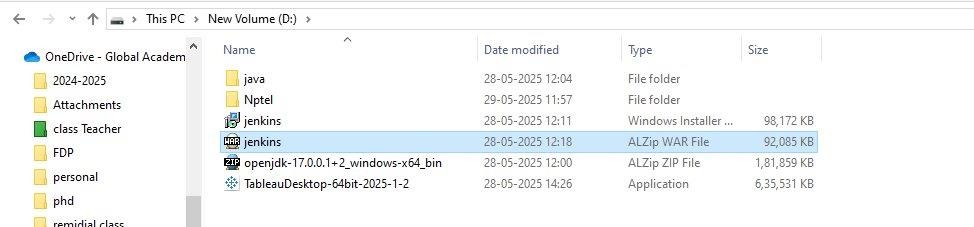


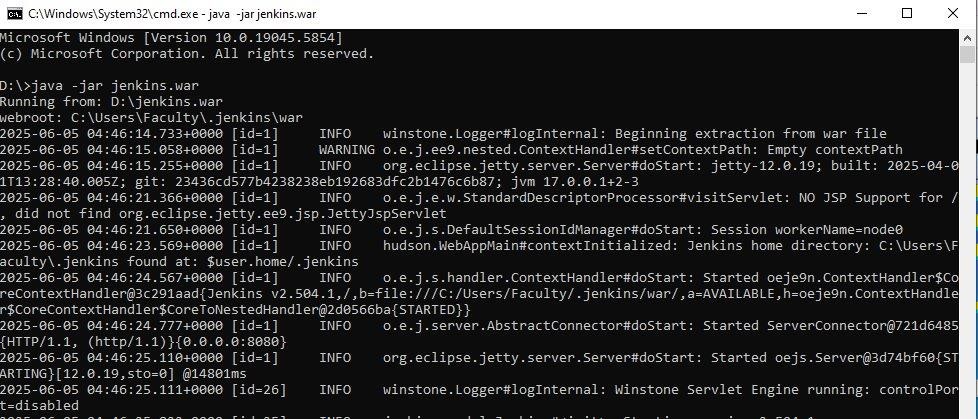
**Step 3:** Thirdly, we will see the list of operating systems for which Jenkins is available as an installer. Based on the operating system, we can select the corresponding option. Here, we are going with **Generic Java Package (.war)** file, a generic file that can set up Jenkins on all the operating systems that have JAVA installed. So click on Generic Java Package (.war) link highlighted in the image below. Consequently, it will download the file.



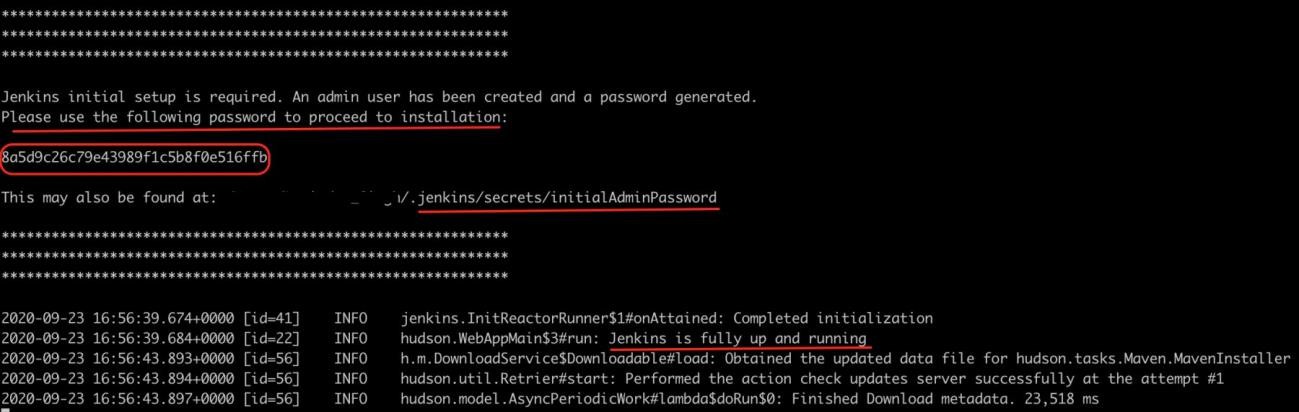
**Step 4:** Fourthly, place this jenkins.war file in any directory in your machine. Now open the command prompt and go to that directory and type the below command:

# Java –jar jenkins.war –y

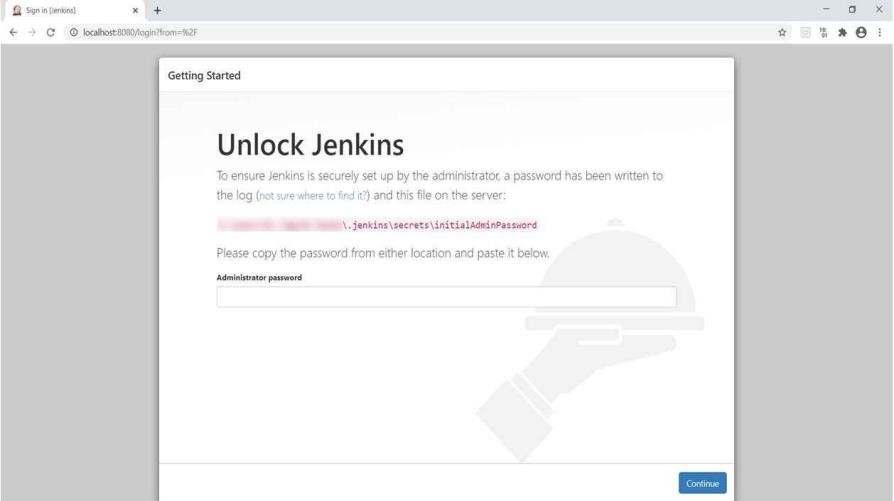
****



When Jenkins is fully up and running One time Password will be generated, copy the password



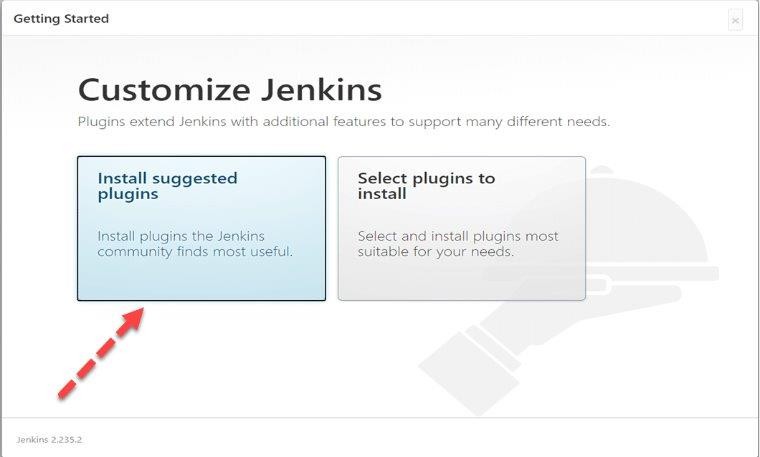
**Step 5:** Now open the browser and navigate to URL: [***http://localhost:8080/***](http://localhost:8080/)After that, press the *Enter* key, and you will see the below page for the first time by default.



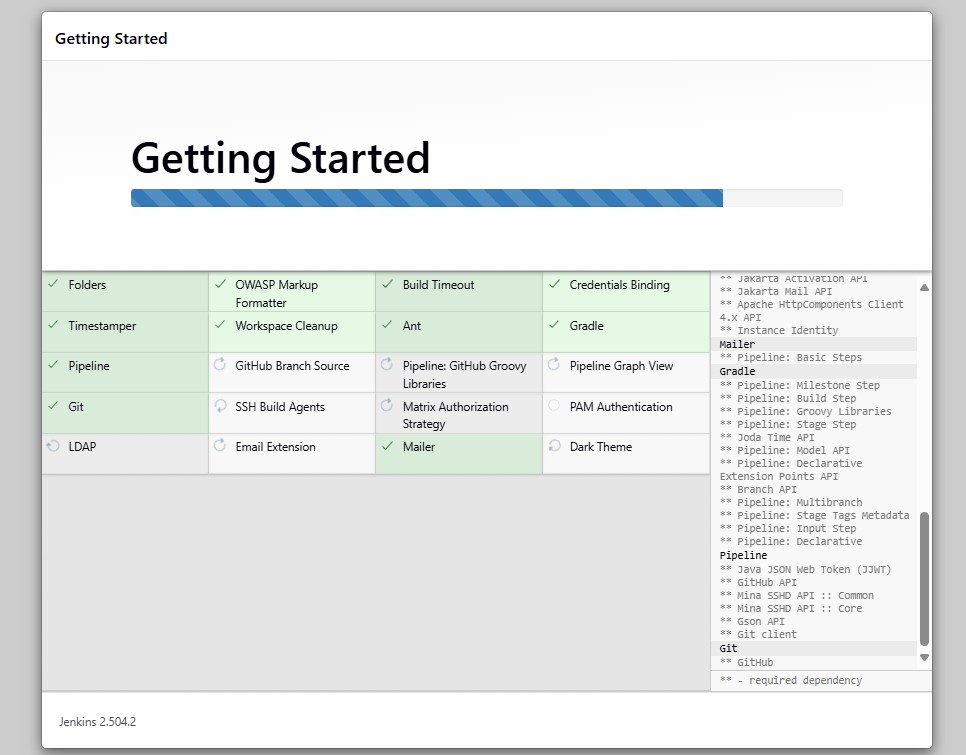
**Step 6*:*** Here, we need to put the password that we copied from the previous step and click the continue button.



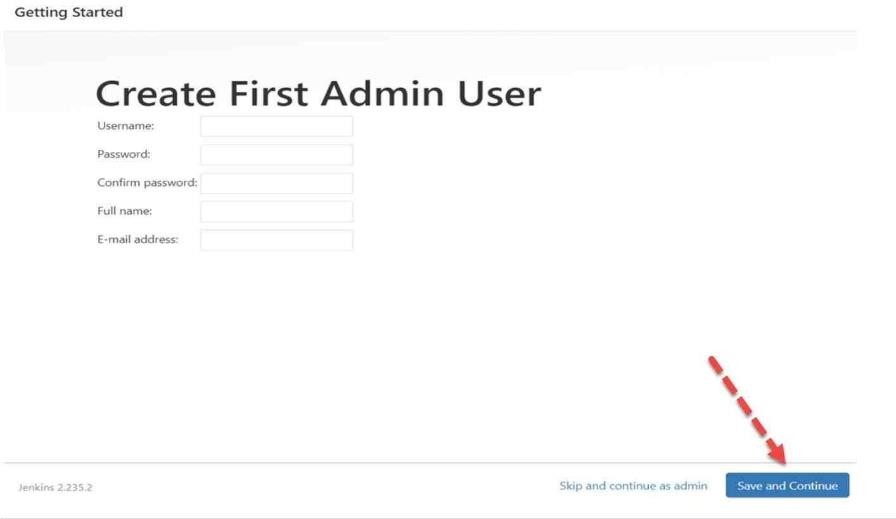
**Step 7**: will be redirected to the page to suggest you install suggested plugins. After that, click on “Install suggested plugins”. Kindly note that if users want to install only selected plugins required, it is recommended to select the option " Select plugins to install".



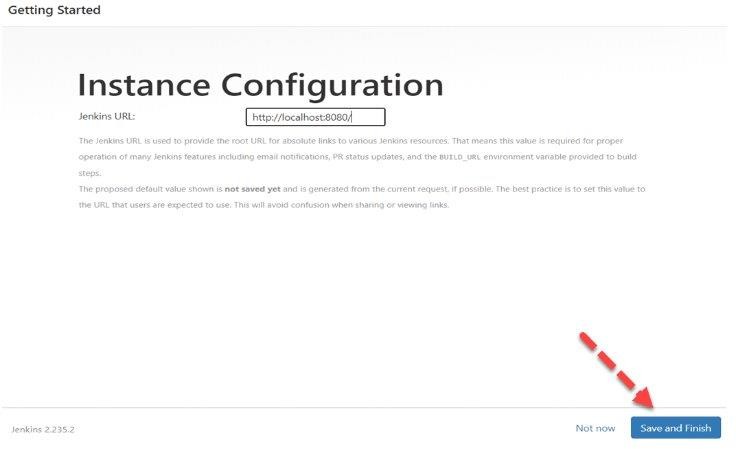
**Step 8:** After clicking on the suggested plugin button, the standard plugin installation will be started automatically, as displayed in the below screen:



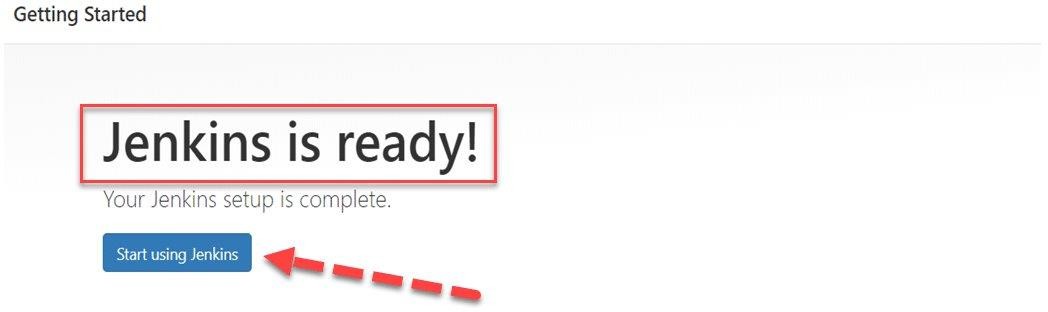
**Step 9**: Finally, after installation of all suggested plugins, you will be redirected to the User Account Page like below:



**Step 10:** After clicking on the ***“Save and Continue Button”,*** you will be redirected to the instance configuration screen. Here click on the ***“Save and Finish Button”.***

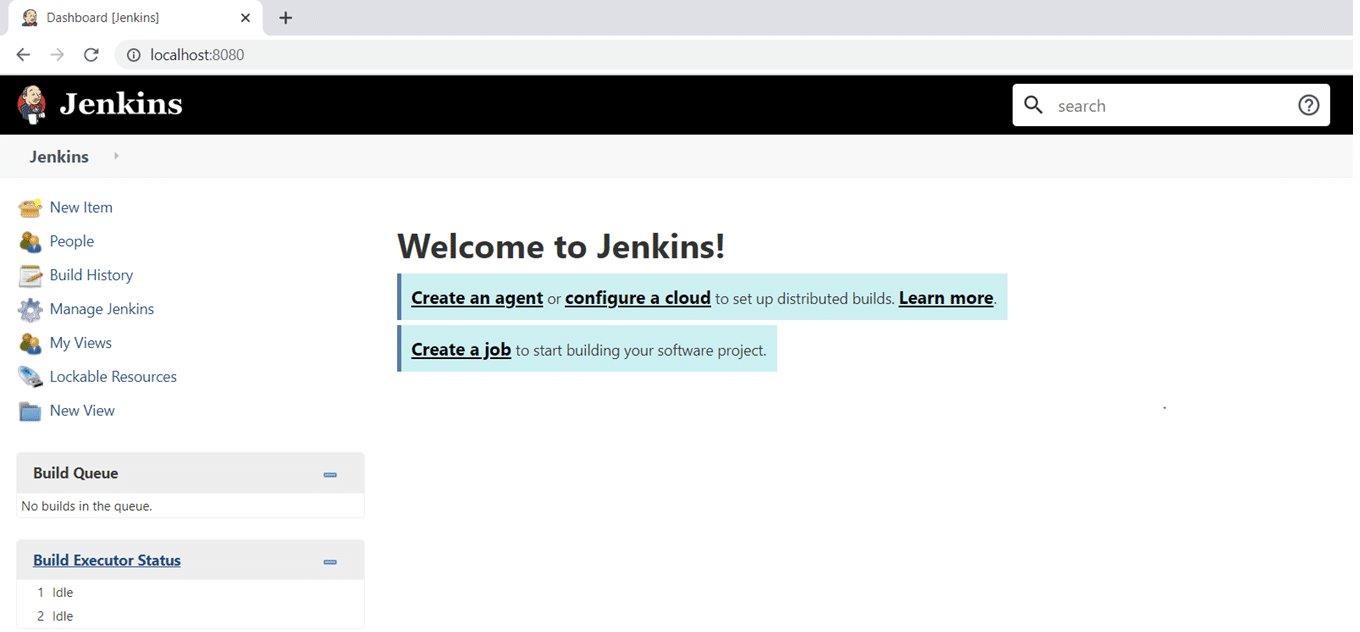
******

**Step 11:** After clicking, you will be redirected to a new screen that will display a message like “Jenkins is Ready”. Now click on the “Start using Jenkins” button.



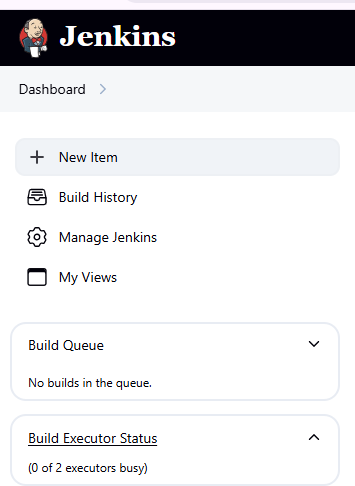
**Step 12:** After clicking on the ***“Start using Jenkins”*** button, you will be redirected to the

*Jenkins* Dashboard.

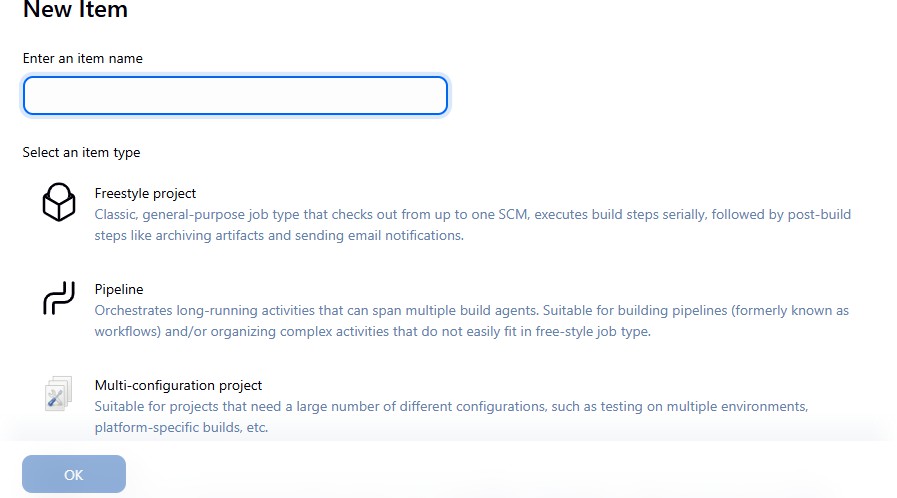


# Create a New Job

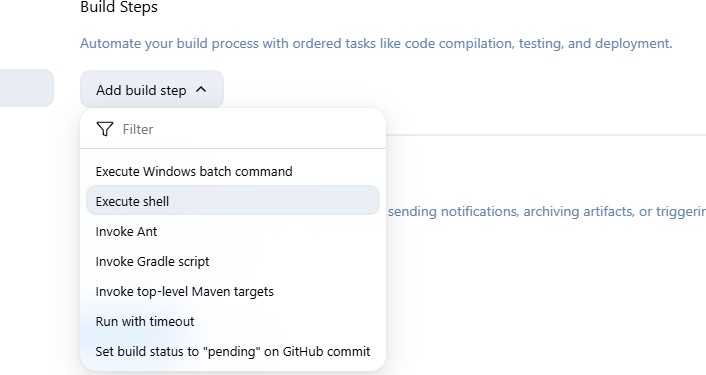
* Click **"New Item"**

****

* Enter job name
* Select job type (e.g., **Freestyle project**, **Pipeline**)
* Click **OK**

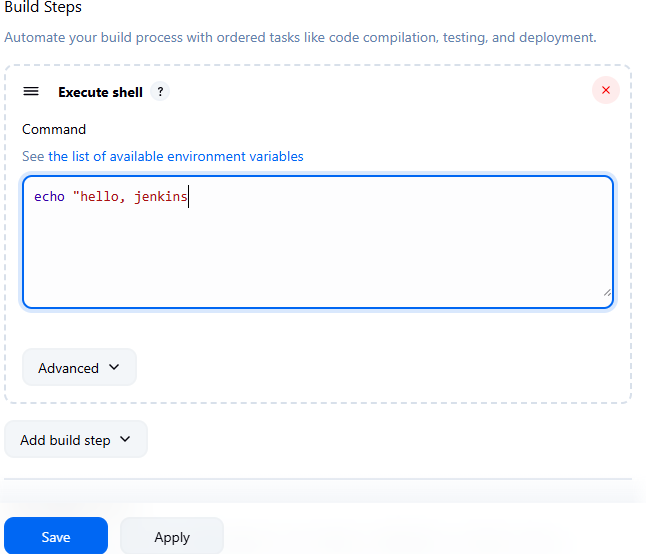
****

# Step 2: Configure Job

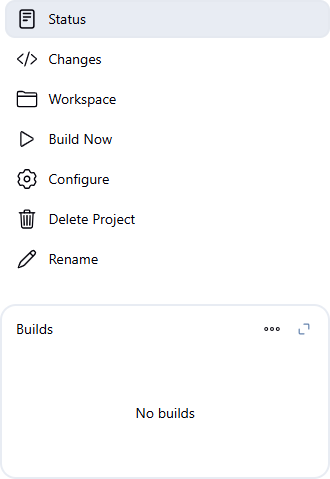
****

**Step 2: Save and Build**

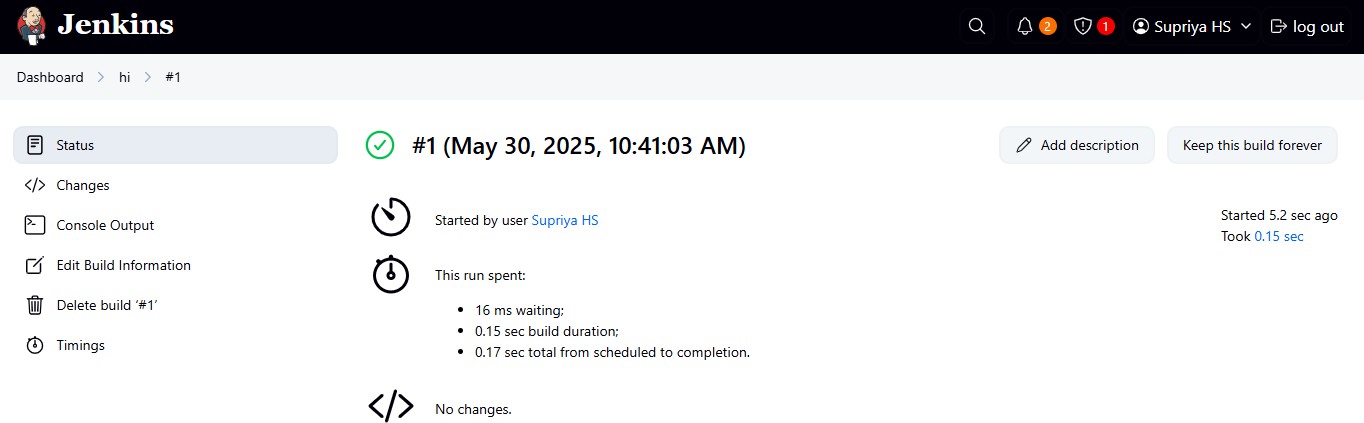
* Click **Save**

****

* Use **"Build Now"** to trigger job



Status of job



Click on console output:



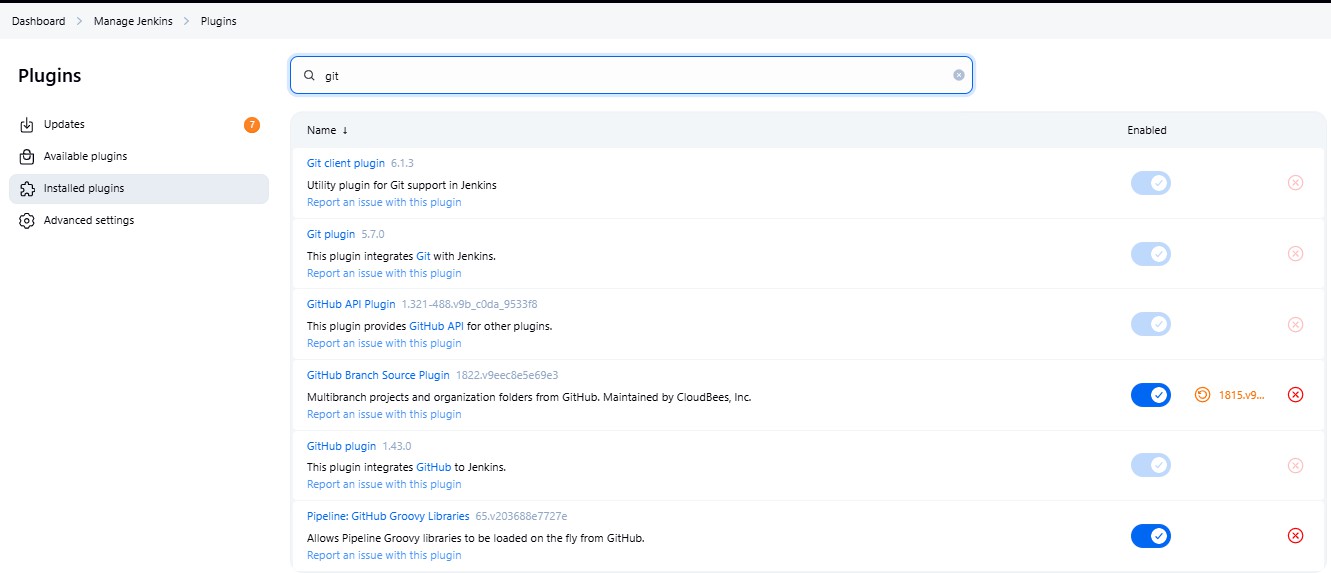
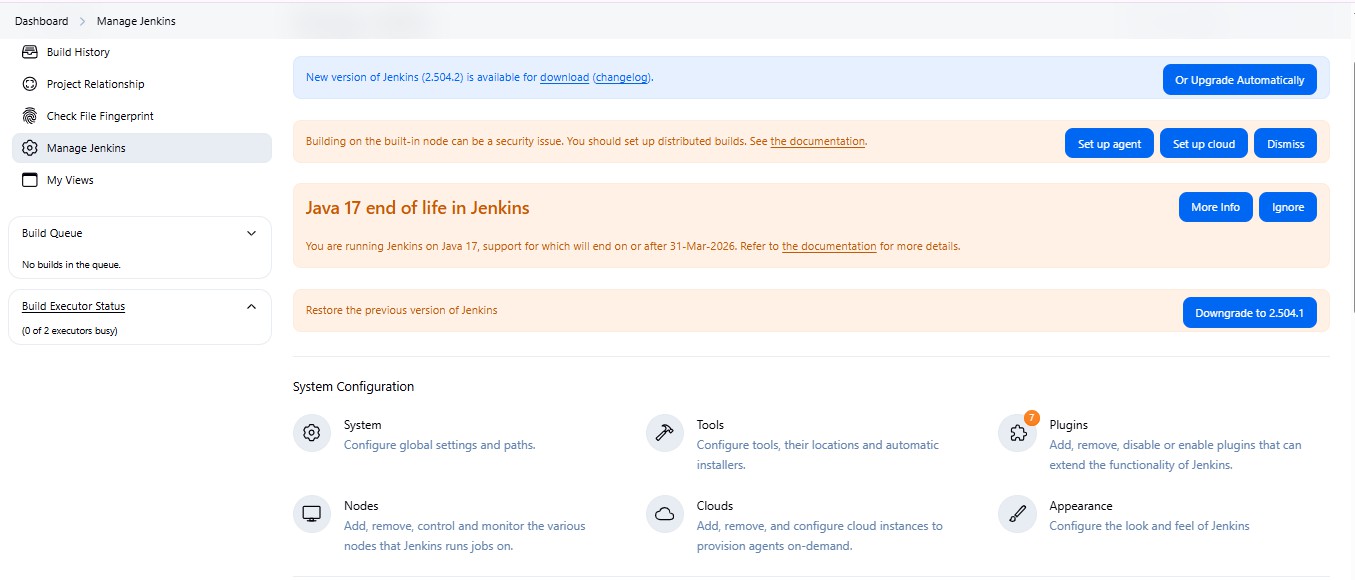
1. **Configure a Jenkins Freestyle project to integrate with a GitHub repository using Source Code Management (SCM), and build based on changes in the Git repository.**

**Solution:**

**Configure GitHub with Jenkins:**

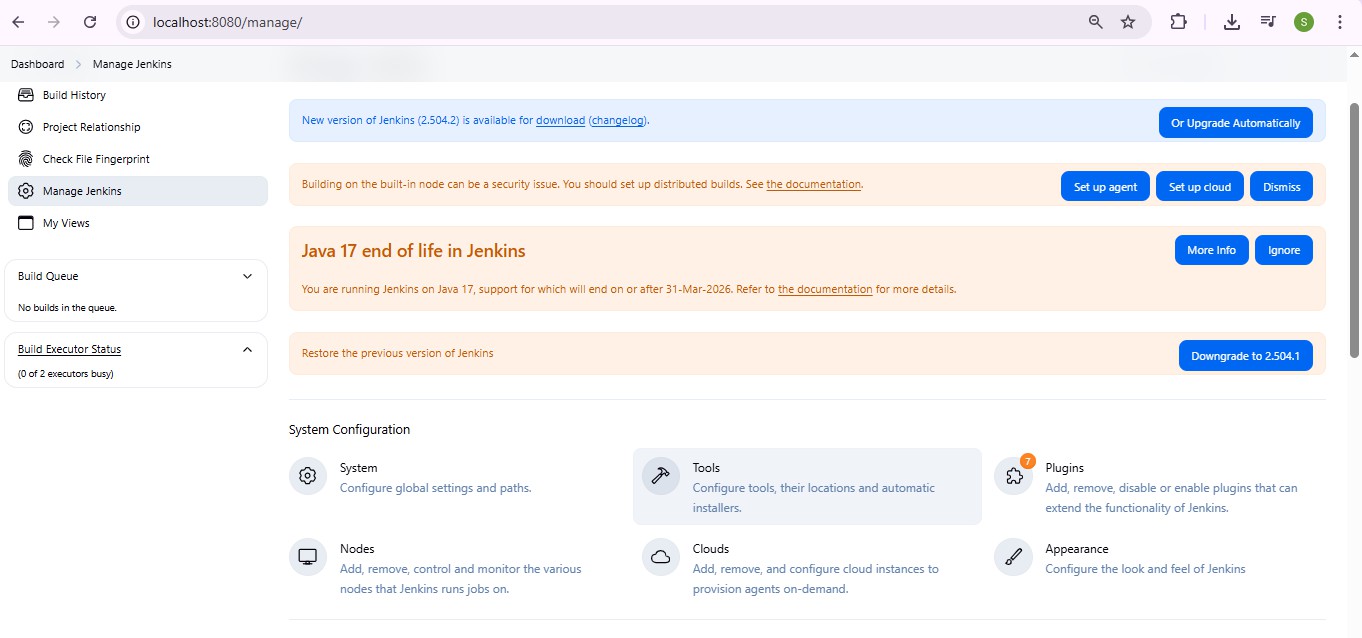
**Step 1: Install Required Jenkins Plugins**

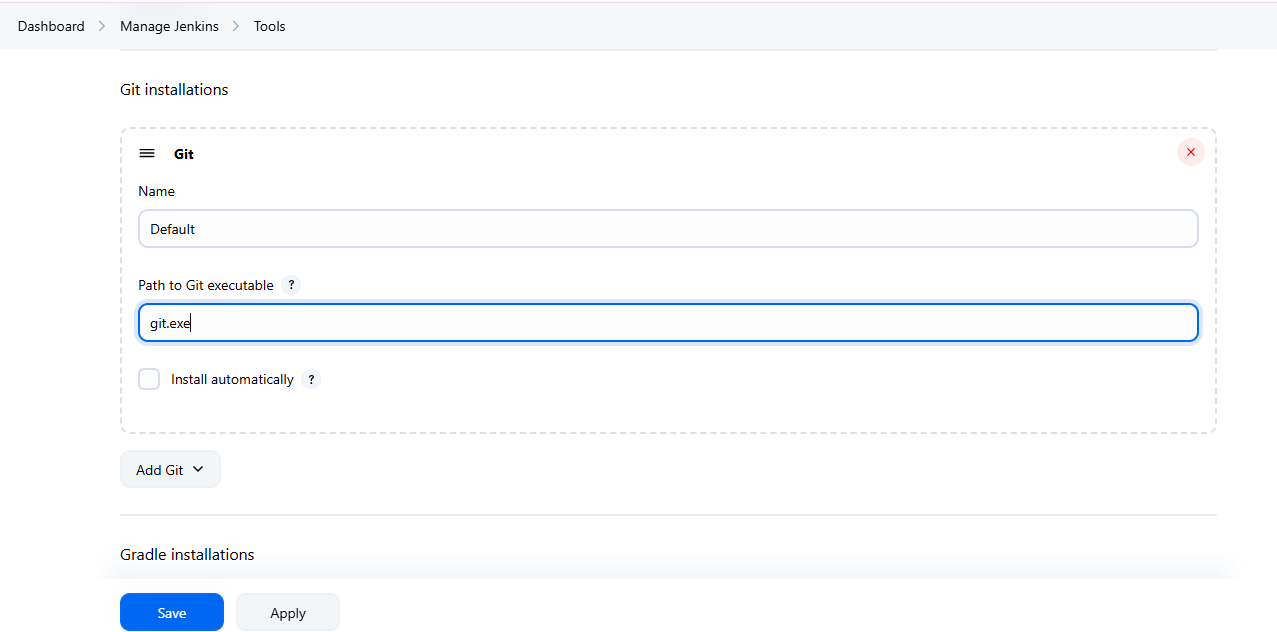
1. Go to Manage Jenkins → Manage Plugins
2. Under **Available** or **Installed** tabs, ensure the following are installed:
   * **Git plugin**
   * **GitHub plugin**
   * **GitHub Integration Plugin**
   * **GitHub Branch Source**
   * **Pipeline (if using scripted pipeline)**

****

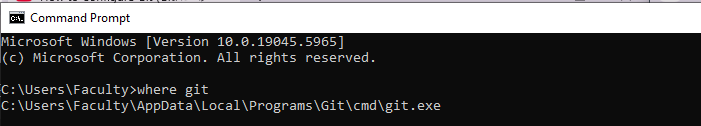
**Step 2: Add local git.exe file Jenkins**

1. Go to **Manage Jenkins → Manage Tool**



****

To get the path for git : **Go to CMD → type where git**

****

Paste the path click on Apply and save

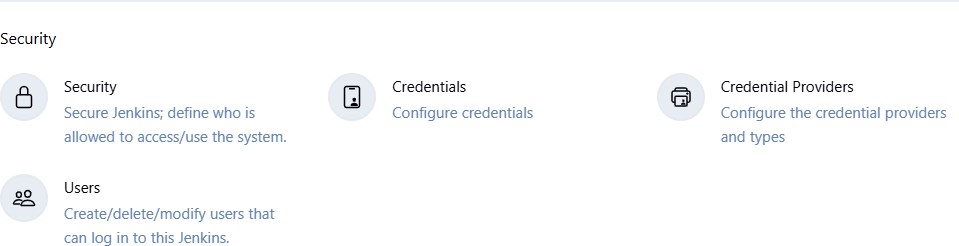
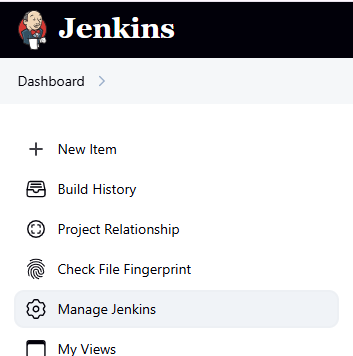
# Create Credentials in Jenkins

**Step 1.** Go to: [http://localhost:8080](http://localhost:8080/) (or your Jenkins server URL) > Login with your

credentials.

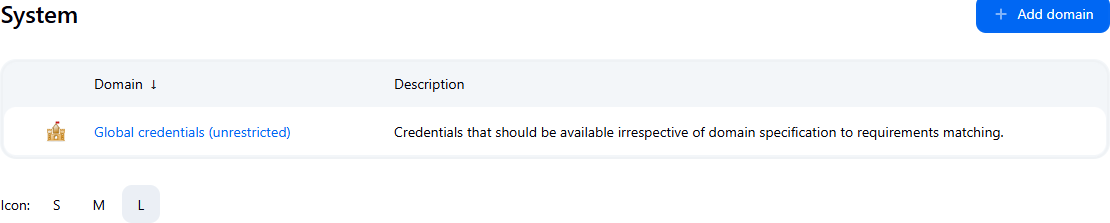
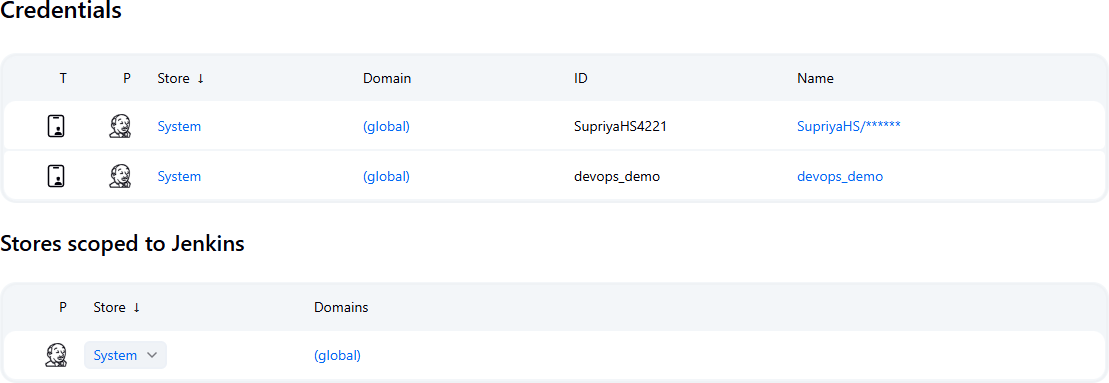
**Step 2.** Navigate to Credentials: In the left-hand panel, click on: Manage Jenkins → security

→ Credentials

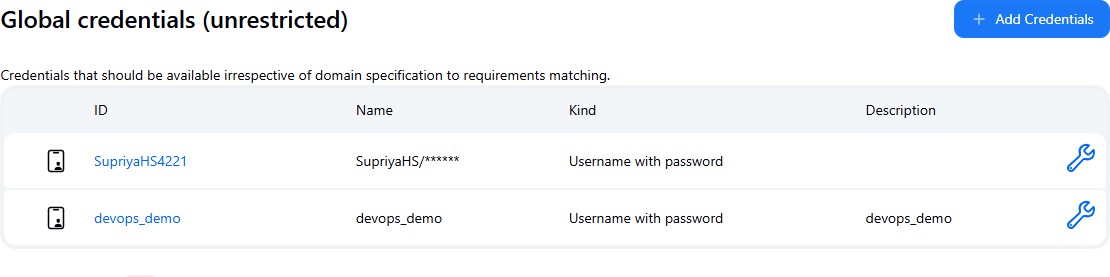


# Step 3. Choose Scope

* You’ll see:
  + **(global)**: Available to all jobs.
  + **Folder-specific** (if you've created folders to organize jobs).
* Click on **(global)** → Add Credentials.



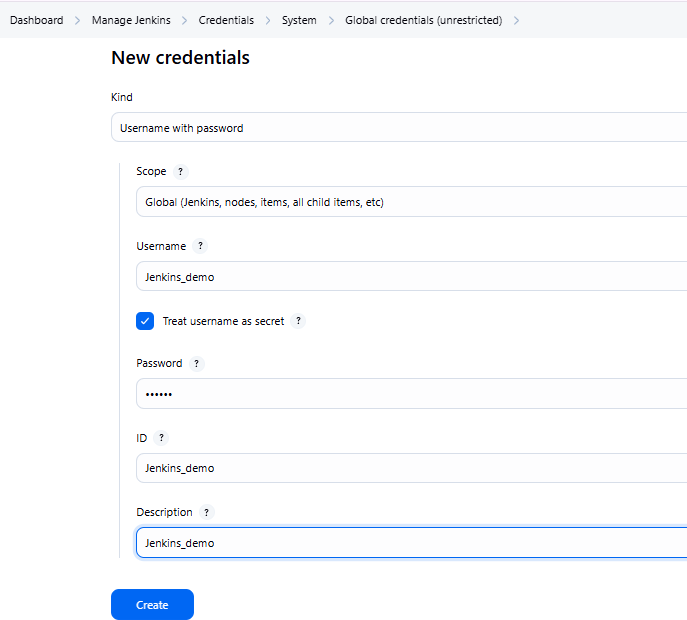
**Step 4**. On the left side, click **Add Credentials**.



**Step 5. Fill in Credential Details:** Choose the correct **Kind** based on what you need:

|  |  |
| --- | --- |
| **Kind** | **Use Case** |
| **Username with password** | Git over HTTPS, DB login |
| **Secret text** | API token, GitHub token |
| **SSH Username with private key** | Git over SSH |
| **Certificate** | TLS/SSL certificates |
| **Secret file** | Any file Jenkins jobs need |

Click **OK**.

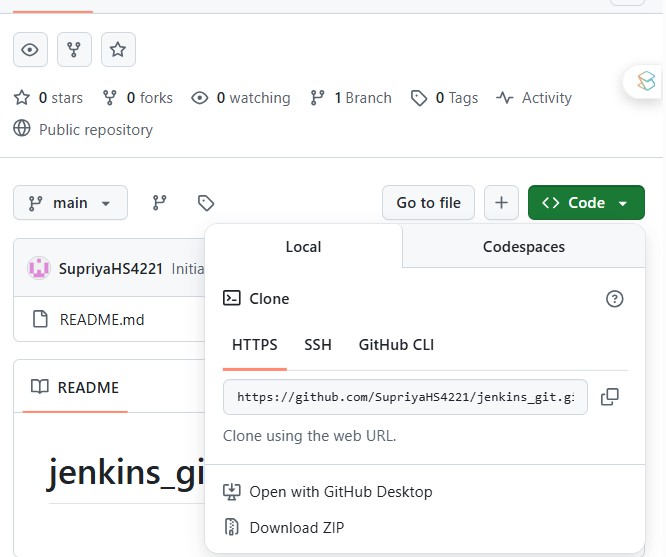


# Create the new job in Jenkins

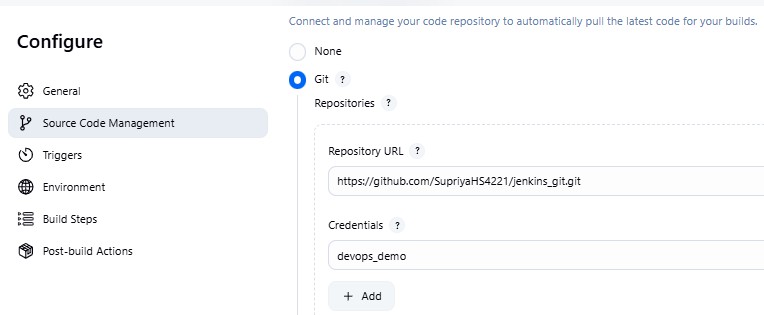
Step 1: **Create a new Freestyle project** in Jenkins named jenkins\_git\_project. Step 2: In the **Source Code Management (SCM)** section:

* Select **Git**.
* Enter the GitHub repository URL:

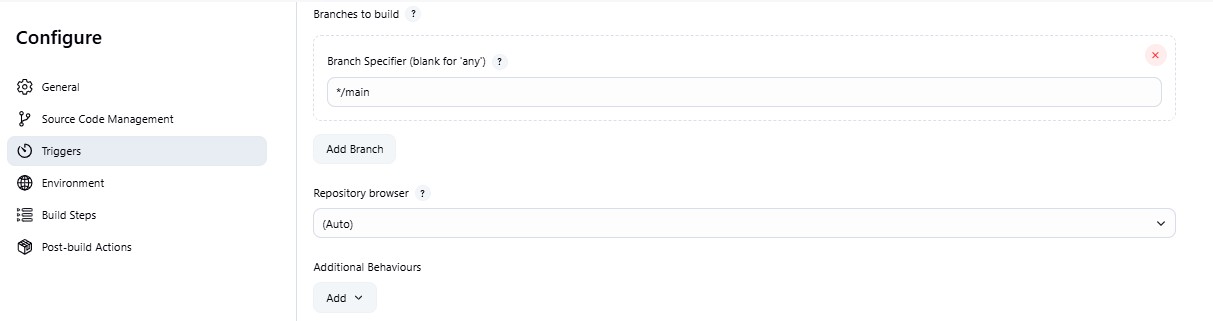
Example: https://github.com/YourUsername/jenkins\_git.git



* Add Git credentials

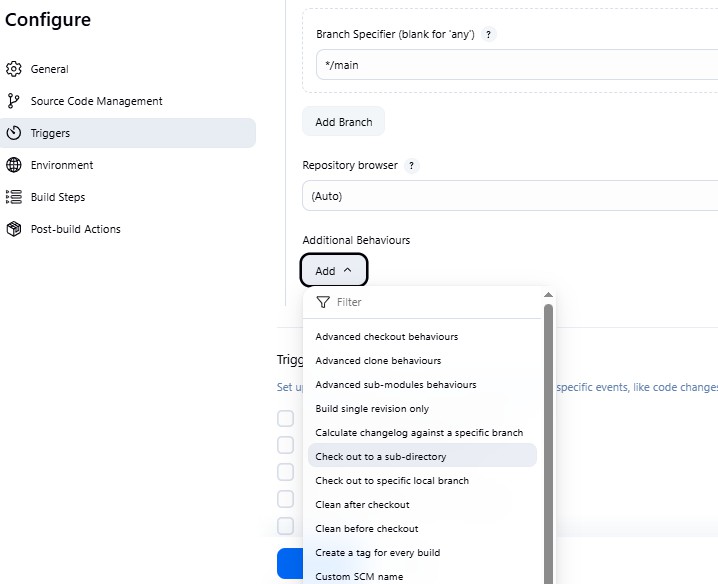


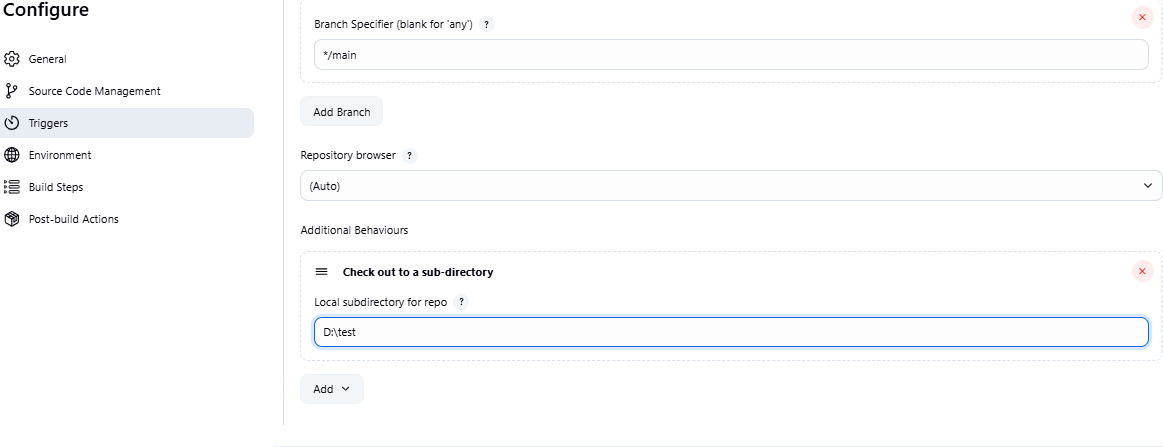
* Specify the **branch to build** (e.g., \*/main).



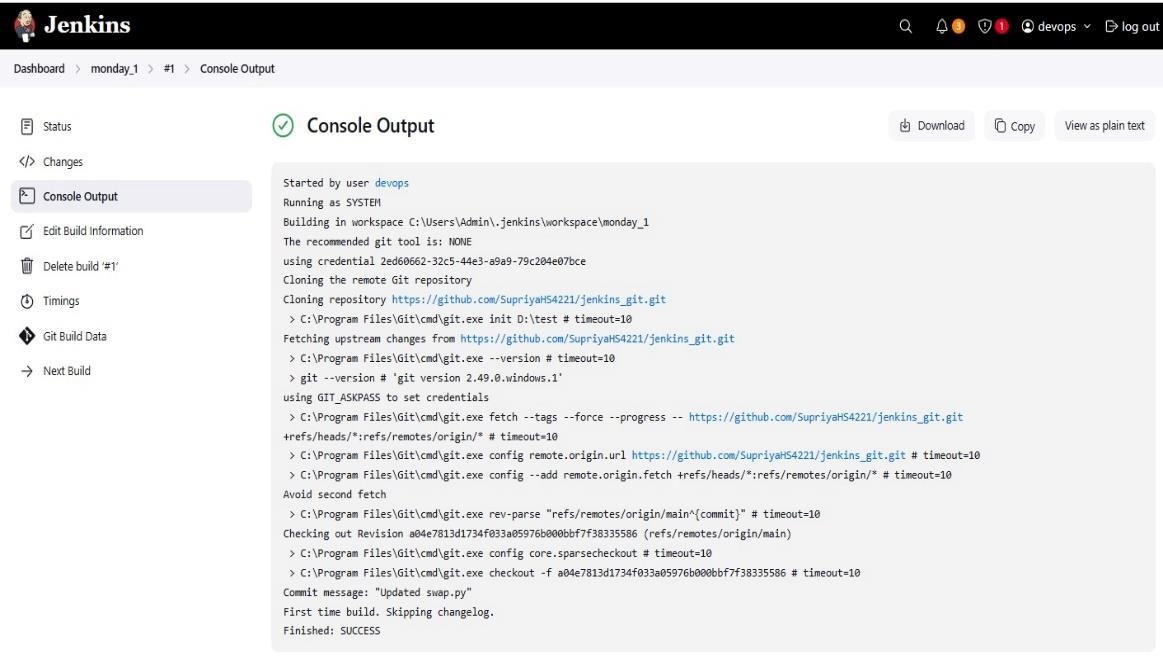
* Click on **"Add"** under **Additional Behaviours** and configure the following:

**Checkout to a specific local sub-directory** – Customize workspace structure.



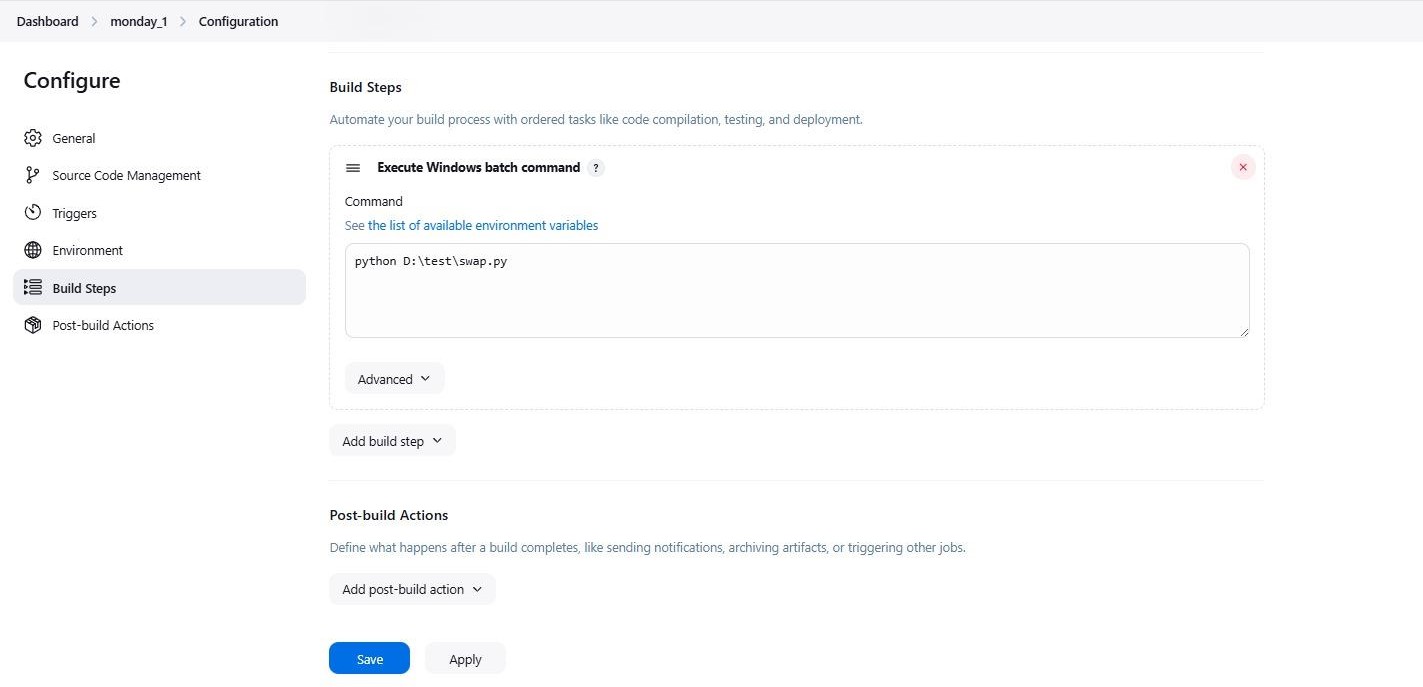


* Save the job and **run the build** manually to test.

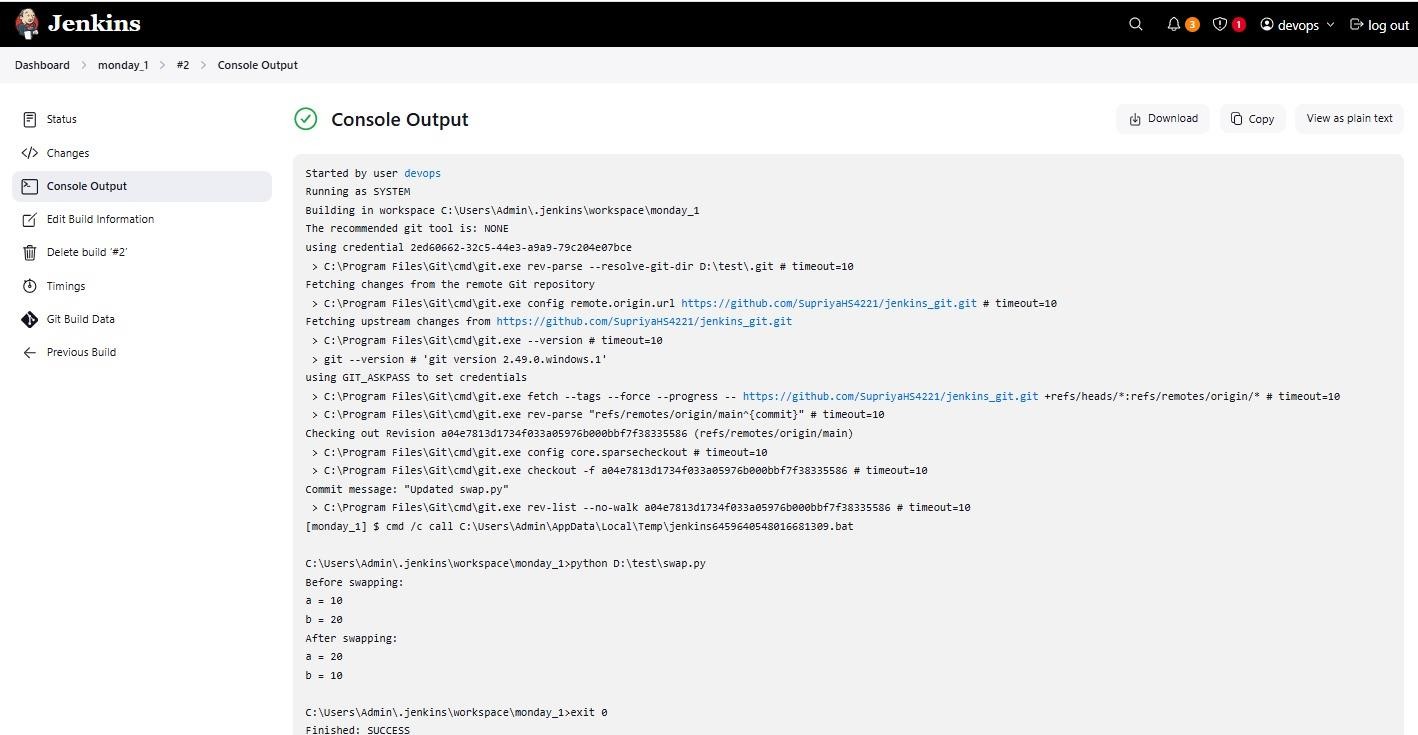


* Now go back to configure job, In the **Build section**, add a simple window batch command such as:

python localfilepath\pythonfile



* Save the job and **run the build** manually to test.



# VIVA QUESTION

1. **What is Git?**

Answer: Git is a distributed version control system that helps track changes in source code during software development. It allows multiple developers to work together efficiently.

# What is the difference between Git and GitHub?

Answer: Git is the version control system (tool). GitHub is a web-based platform for hosting Git repositories, providing collaboration and project management features.

# What is a repository in Git?

Answer: A repository (repo) is a directory or storage space where your project files and the entire version history of your project are stored.

# What is the purpose of git clone?

Answer: git clone is used to create a copy of a remote repository on your local machine, including all the files, branches, and commit history.

# Explain the purpose of git commit.

git commit is used to save your changes to the local repository. It captures the state of the project at a particular point in time, allowing you to track changes over time.

# What is the difference between git pull and git fetch?

git pull is used to fetch changes from a remote repository and automatically merge them into your current branch, while git fetch only retrieves the changes without merging them.

# What is a branch in Git?

A branch in Git is a separate line of development. It allows you to work on different features or fixes without affecting the main codebase (usually the master or main branch).

# How do you create and switch to a new branch?

$git checkout -b branch\_name

# . How do you connect a local repo to GitHub?

$git remote add origin <github-repo-url>

$git push -u origin master

# How do you integrate Git with Jenkins?

Install Git plugin in Jenkins. In the Jenkins job, select "Git" as SCM and enter the repository URL. Use credentials if required for GitHub access.

# What does git reset --hard HEAD~1 do?

It deletes the last commit permanently along with changes in the working directory and staging area.

# How can you undo a git push?

Use git revert <commit> to create a new commit that undoes changes, then push again.

# What's the difference between git merge and git rebase?

merge creates a new commit to combine histories; rebase rewrites commit history linearly.

# How do you resolve a merge conflict?

Edit conflicting files manually, mark them as resolved with git add, then commit.

# What happens if you delete the .git folder in a project?

You lose all version history; the folder is the heart of the Git repository.

# Can you commit without staging? How?

Yes, with git commit -a -m "msg" which stages and commits tracked file changes.

# How do you squash multiple commits into one?

Use git rebase -i HEAD~n and choose squash for desired commits.

# What's the difference between HEAD, HEAD~1, and ORIG\_HEAD?

HEAD: current commit, HEAD~1: previous commit, ORIG\_HEAD: last state of HEAD before a change.

# How can you make a branch from a previous commit?

git checkout -b new-branch <commit-hash>

# What is a detached HEAD state?

It's when you're not on any branch, just viewing a specific commit.

# What triggers a Jenkins job automatically?

Webhooks, SCM polling, timers (cron), or upstream builds.

# What if Jenkins fails due to a plugin error?

Remove or disable the plugin manually from the JENKINS\_HOME/plugins directory.

1. **What is the role of workspace in Jenkins?**

It stores the job’s files during execution like code, build outputs, etc.

# What’s the difference between build now and scheduled builds?

Build now triggers manually; scheduled builds are automated via cron.