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**

```
Faculty@MB419-2 MINGW64 ~
$ git init project1
Initialized empty Git repository in C:/Users/Faculty/project1/.git/
```

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

```
Faculty@MB419-2 MINGW64 ~
$ cd project1

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ touch helloworld.c
```

Step 3. Add the file to the staging area:

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        helloworld.c

nothing added to commit but untracked files present (use "git add" to track)
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git add helloworld.c

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   helloworld.c
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git commit -m "first comment"
[main (root-commit) 757126d] first comment
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 helloworld.c
```

Others commands:

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git diff helloworld.c
diff --git a/helloworld.c b/helloworld.c
index e69de29..e0e8a48 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -0,0 +1,6 @@
+#include <stdio.h>
+
+int main() {
+  printf("Hello, World!\n");
+  return 0;
+}
\ No newline at end of file
```

After changes add helloworld.c to repo:

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git add .

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   helloworld.c
```

Final commit:

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git commit -m "sec commit"
[main 2c2d1c8] sec commit
 1 file changed, 6 insertions(+)
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git log
commit 2c2d1c802f5dbbe323c28f296027a55b1277b919 (HEAD -> main)
Author: Supriya.hs <supriyahs.gat@gmail.com>
Date:   Mon Apr 7 12:58:00 2025 +0530

    sec commit

commit 757126dcf2f3f22f7b7fba28b7fbde6fa245ce0b
Author: Supriya.hs <supriyahs.gat@gmail.com>
Date:   Mon Apr 7 12:51:55 2025 +0530

    first comment
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch feature-branch
```

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch
  feature-branch
* main
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'
```

```
Faculty@MB419-2 MINGW64 ~/project1 (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git checkout -b new-branch
Switched to a new branch 'new-branch'

Faculty@MB419-2 MINGW64 ~/project1 (new-branch)
$ git status
On branch new-branch
nothing to commit, working tree clean
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (feature-branch)
$ git checkout main
Switched to branch 'main'
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git merge feature-branch
Updating 2c2d1c8..315fb22
Fast-forward
 img_hello_world.jpg | Bin 0 -> 1922 bytes
 index.html.txt      |  16 ++++++++++++++++
 2 files changed, 16 insertions(+)
 create mode 100644 img_hello_world.jpg
 create mode 100644 index.html.txt
```

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"

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git add .

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git commit -m "merge feature-branch into main
> "
On branch main
nothing to commit, working tree clean
```

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.

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch -d feature-branch
Deleted branch feature-branch (was 315fb22).
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch
* main
  new-branch
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch
  branch2
* main
  new-branch

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git checkout branch2
M       helloworld.c
M       index.html.txt
Switched to branch 'branch2'
```

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

$ git status

```
Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git status
On branch branch2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   helloworld.c
        modified:   index.html.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        P_TT.png
        git_lab3/
        login.html

no changes added to commit (use "git add" and/or "git commit -a")
```

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.

```
Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git stash
Saved working directory and index state WIP on branch2: 315fb22 added html file
```

List the Git Stash Entries: $git stash list

```
Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git stash list
stash@{0}: WIP on branch2: 315fb22 added html file
stash@{1}: WIP on main: 315fb22 added html file
```

Show the changes recorded in the stash: **$ git stash show [stash_ID]**

Ex: $git stash show stash@{0}

```
Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git stash show stash@{0}
 helloworld.c   |  2 +-
 index.html.txt |  2 +-
 login.html     | 98 ++++++++++++++++++++++++++++++++++
+++++++++++++
 3 files changed, 100 insertions(+), 2 deletions(-)

Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git stash show stash@{1}
 index.html.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

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}

```
Faculty@MB419-2 MINGW64 ~/project1 (branch2)
$ git stash apply
error: Your local changes to the following files would be overwritten by merge:
        index.html.txt
Please commit your changes or stash them before you merge.
Aborting
On branch branch2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   helloworld.c
        modified:   index.html.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        P_TT.png
        git_lab3/

no changes added to commit (use "git add" and/or "git commit -a")
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git stash drop
Dropped refs/stash@{0} (249eae96dedbf63fb6d64757a852737fac88d9d9)
```

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

```
marij@BOSKO MINGW64 ~/Desktop/Git project (master)
$ git stash drop stash@{3}
Dropped stash@{3} (417bb2e1a0c744dfcae3845078fe05770d4fc503)
```

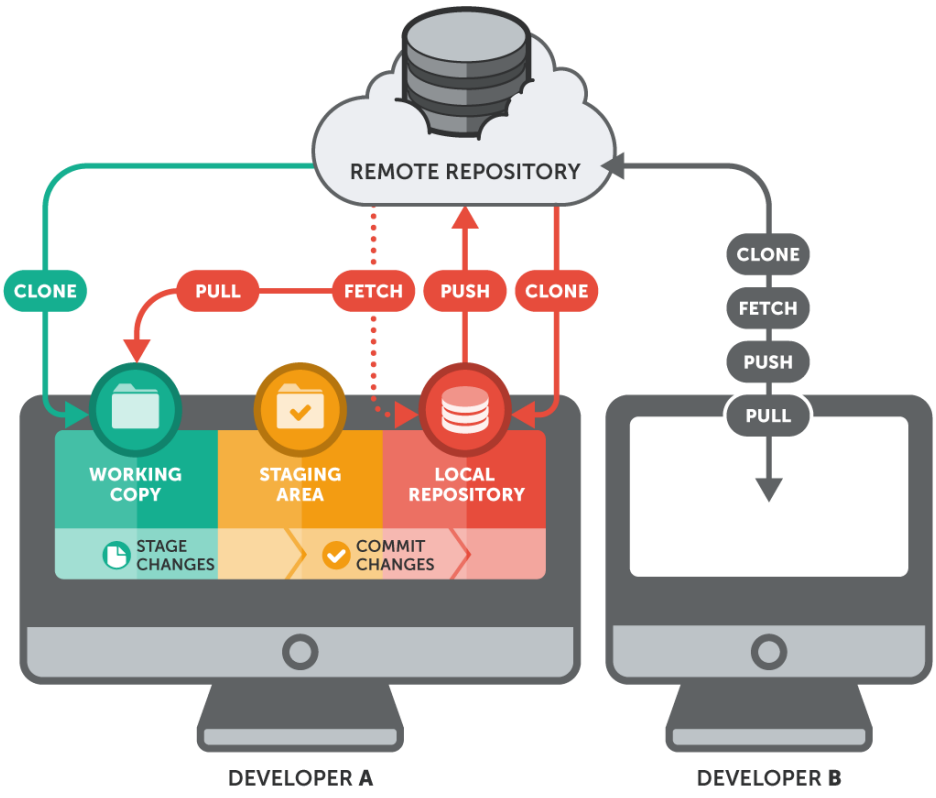
Drop All Git Stashes: $git stash clear

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.



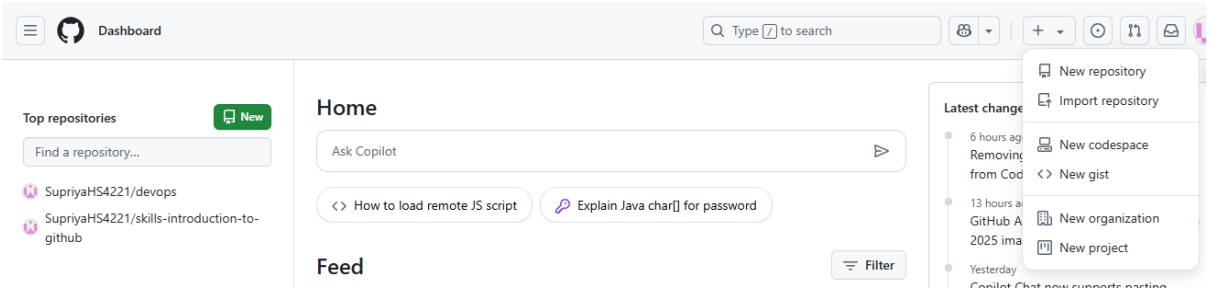
To clone a remote Git repository to your local machine, you can use the git clone command. Here's the general syntax: **$git clone <repository_url>**

Create a GitHub Account: Open your browser and go to: https://github.com

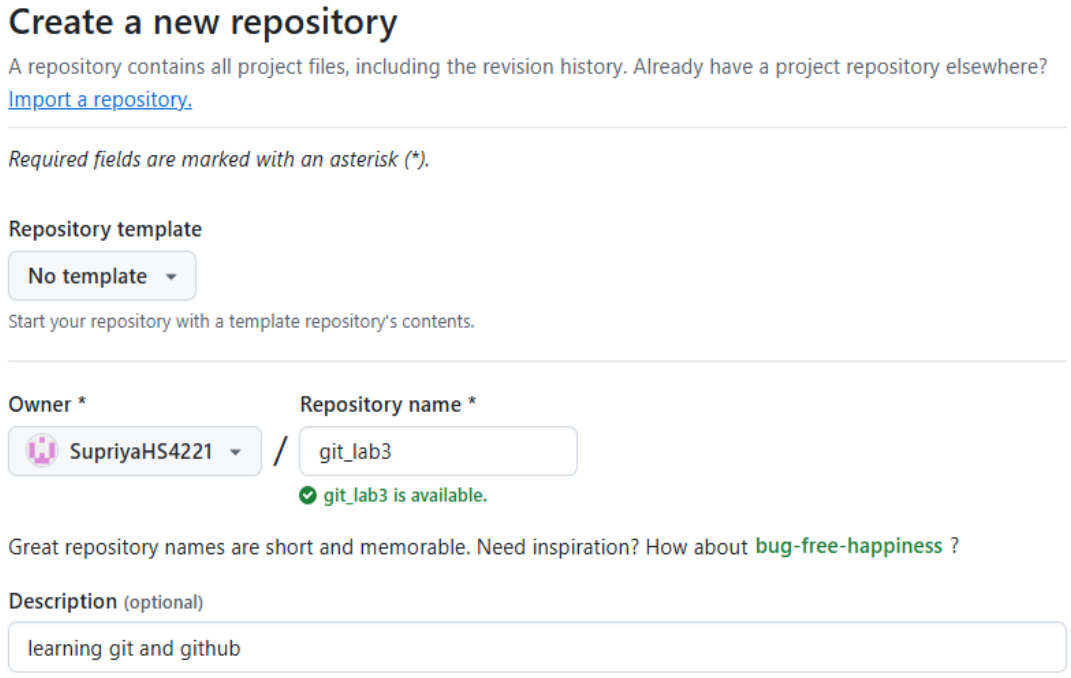
Create GitHub account with necessary details.

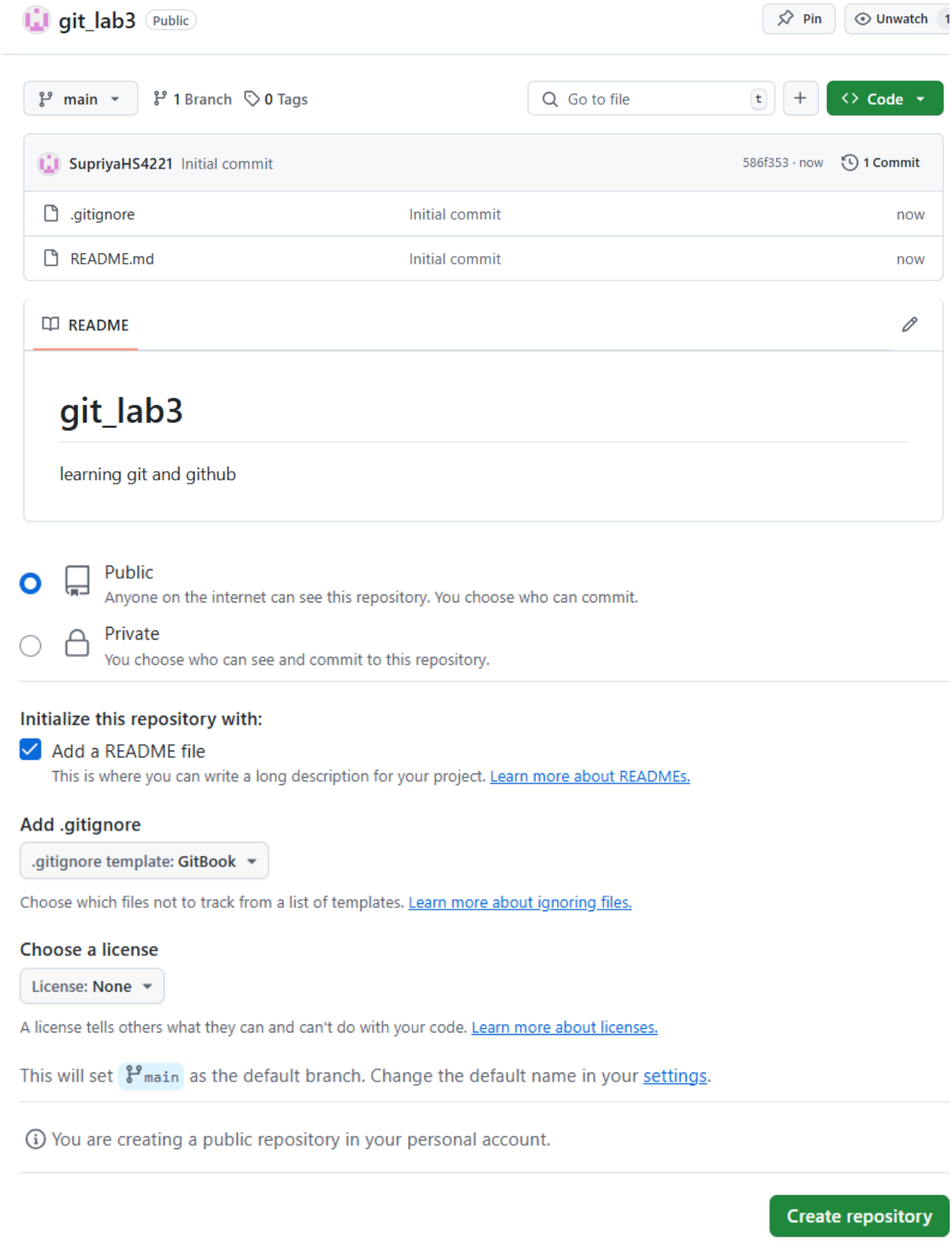
Create a New GitHub Repository:

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

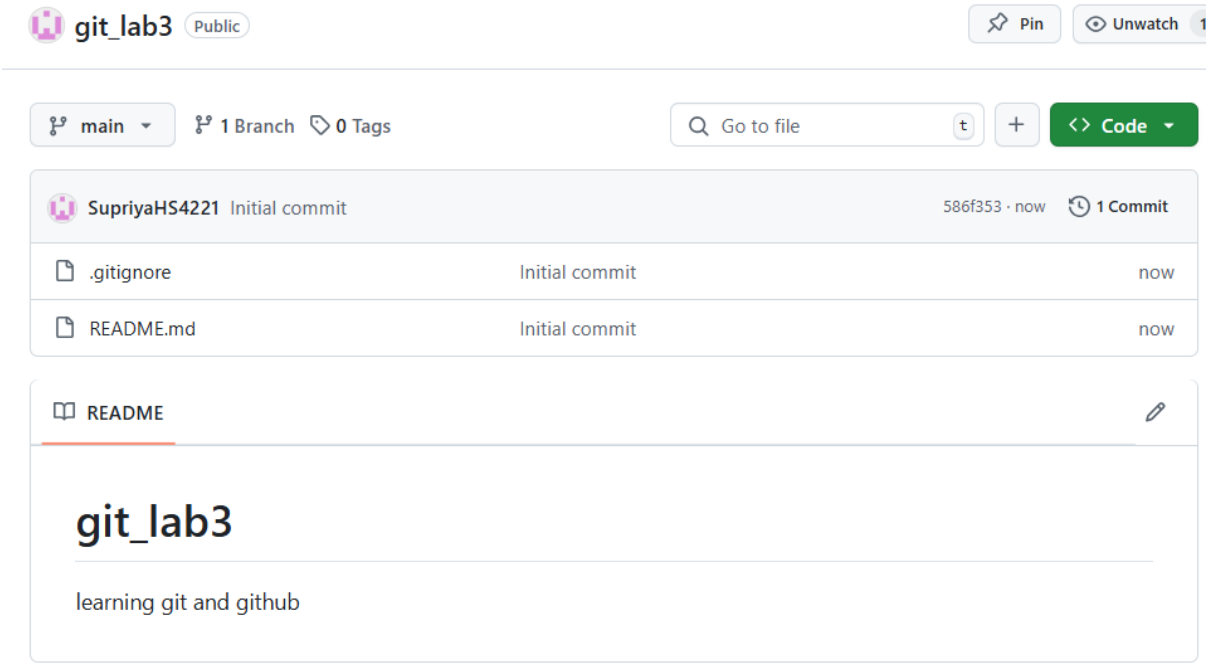


- Enter a repository name (e.g., my-project).
- Add a **description**.
- Choose **Public** or **Private** visibility.
- Check **"Initialize this repository with a README"**.
- Click the green **"Create repository"** button.



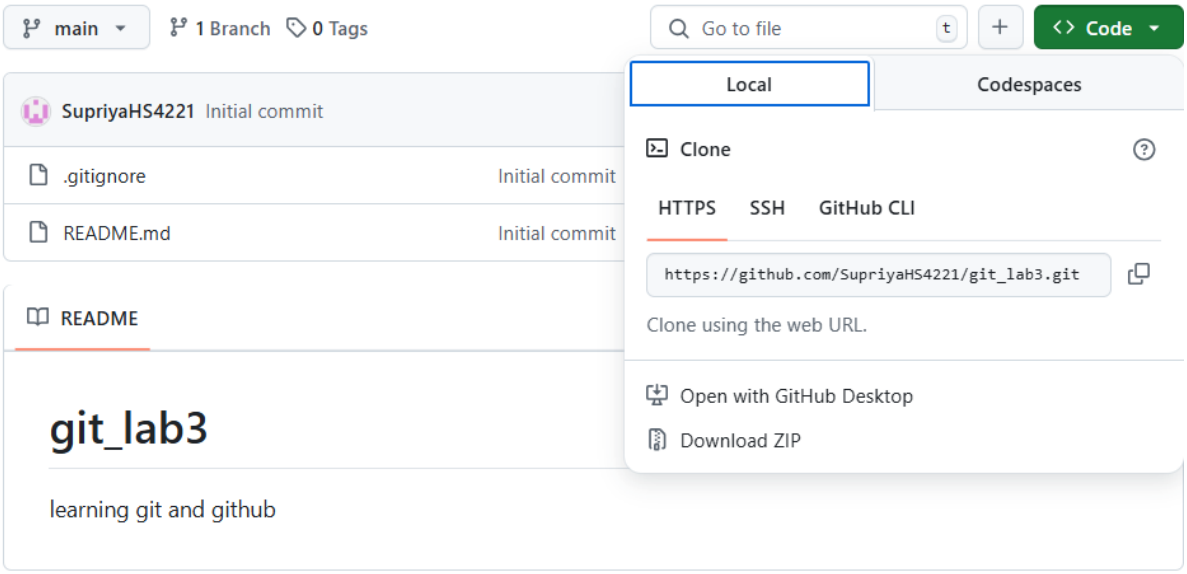


Finally, your repository will look like below:



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:

```
Faculty@MB419-2 MINGW64 ~
$ cd project1

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ ll
total 6
-rw-r--r-- 1 Faculty 197121   80 Apr  7 12:54 helloworld.c
-rw-r--r-- 1 Faculty 197121 1922 Apr 10 09:56 img_hello_world.jpg
-rw-r--r-- 1 Faculty 197121  360 Apr 10 09:56 index.html.txt
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git clone -b main https://github.com/SupriyaHS4221/git_lab3.git
Cloning into 'git_lab3'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.
```

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.

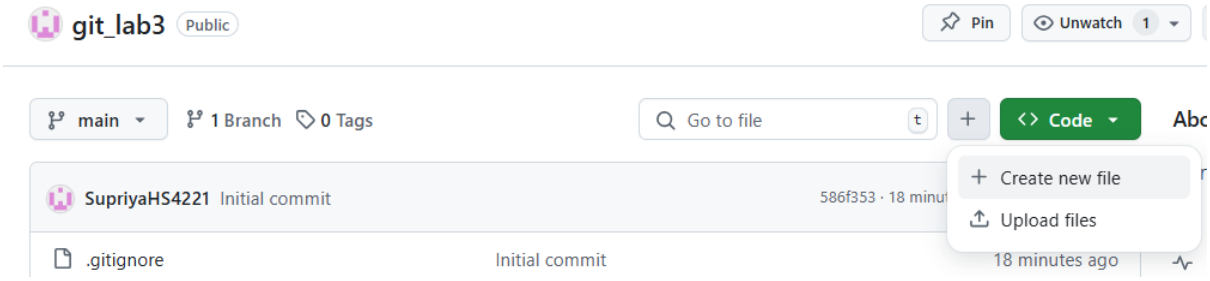
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ git fetch origin

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Pulling changes from a remote repository:
$git pull is a convenient shortcut for completing both git fetch and git merge in the same command:

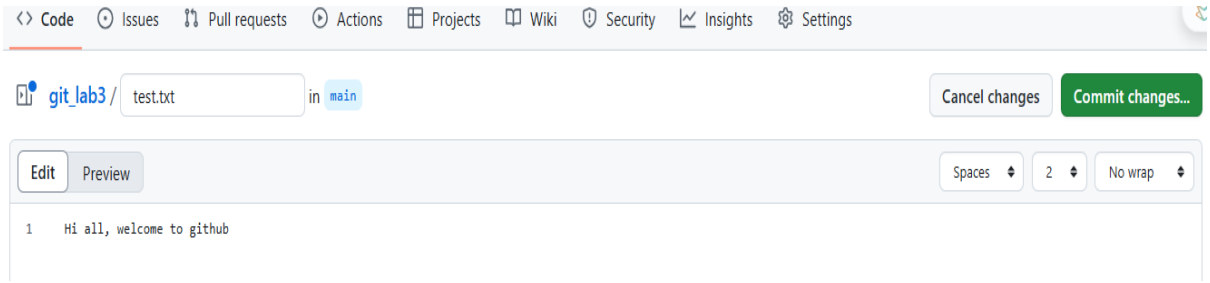
$git pull <remote> <branch>
<remote> → Usually origin (the default remote)
<branch> → Usually main or master
Ex: $git pull origin main

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git pull origin main
From https://github.com/SupriyaHS4221/git_lab3
 * branch            main       -> FETCH_HEAD
Already up to date.
```

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**

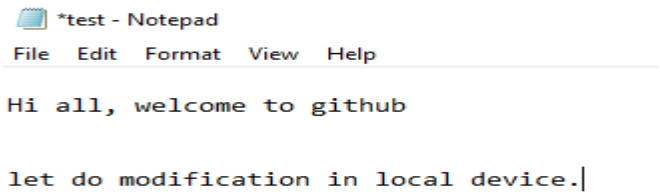
Step 3: Let's do git pull to fetch changes made in remote repo to local repo:

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git pull origin main
From https://github.com/SupriyaHS4221/git_lab3
 * branch            main       -> FETCH_HEAD
Already up to date.
```

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

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ ls
README.md  Supriya.text  a  b  git_lab3/  libs/  pull  sample.txt  test.txt
```

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



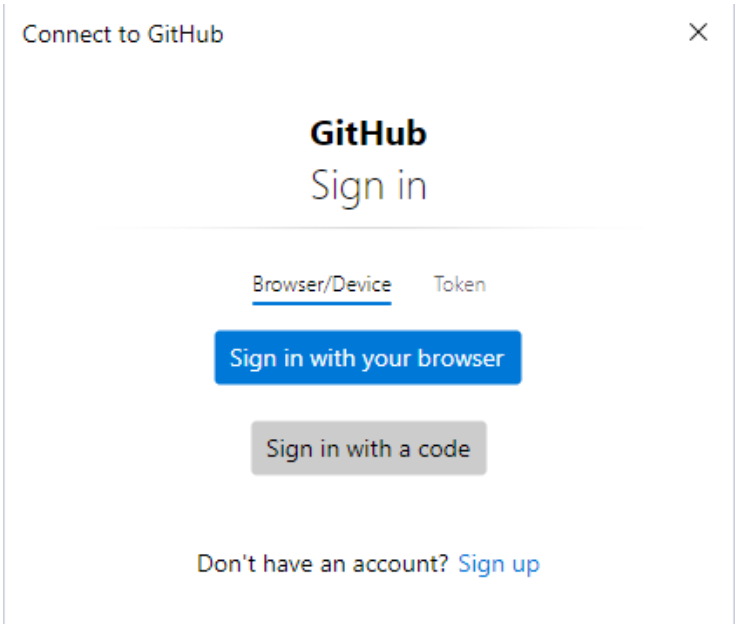
Step 6: Commit the changed file

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git add .

Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git commit -m " new file commit"
[main f83a7b4]  new file commit
 7 files changed, 15 insertions(+)
 create mode 100644 Supriya
 create mode 100644 a
 create mode 100644 b
 create mode 160000 git_lab3
```

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

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git push origin main
Enumerating objects: 27, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 4 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (18/18), 1.77 KiB | 139.00 KiB/s, done.
Total 18 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/SupriyaHS4221/git_lab3.git
   cc01fe4..f83a7b4  main -> main
```

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.

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (slave)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git rebase slave
Successfully rebased and updated refs/heads/main.

Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git checkout slave
Switched to branch 'slave'

Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (slave)
$ ls
README.md  Supriya.text  b     git_lab3/  pull        test.txt
Supriya    a             gg.c  libs/      sample.txt
```

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

```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git tag v1.0  b3b2dfec2a0cf88dcc19cd28cbb78dc796d66c8e
```

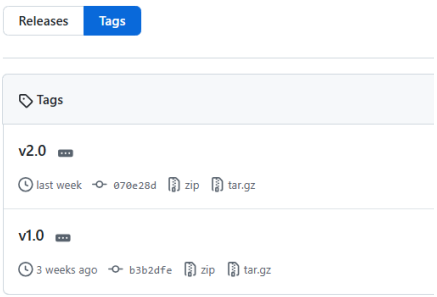
```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git log
commit b3b2dfec2a0cf88dcc19cd28cbb78dc796d66c8e (HEAD -> main, tag: v1.0, origin/main, origin/HEAD)
Author: SupriyaHS4221 <Supriyahs.gat@gmail.com>
Date:   Mon May 19 09:24:06 2025 +0530

    b.c
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git branch
* main

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git checkout -b slave
Switched to a new branch 'slave'

Faculty@MB419-2 MINGW64 ~/project1/6b (slave)
$ echo "welcome to cherrypick">a.c

Faculty@MB419-2 MINGW64 ~/project1/6b (slave)
$ git add .
warning: in the working copy of 'a.c', LF will be replaced by CRLF the next time Git touches it

Faculty@MB419-2 MINGW64 ~/project1/6b (slave)
$ git commit -m "cherrypick"
[slave 142119d] cherrypick
 1 file changed, 1 insertion(+), 1 deletion(-)

Faculty@MB419-2 MINGW64 ~/project1/6b (slave)
$ ls
3a.txt  README.md  a.c  b.c

Faculty@MB419-2 MINGW64 ~/project1/6b (slave)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Step 4: git cherry-pick slave .. main

```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git cherry-pick slave .. main
[main a749ba9] cherrypick
 Date: Thu Jun 12 09:37:59 2025 +0530
 1 file changed, 1 insertion(+), 1 deletion(-)

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ ls
3a.txt  README.md  a.c  b.c

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ cat a.c
welcome to cherrypick
```

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:

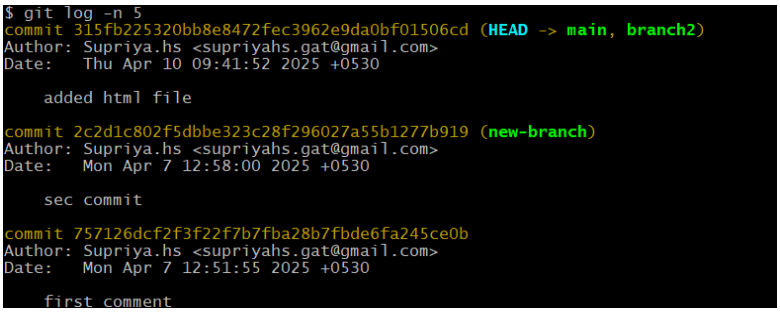
$git log --author=“ “ --since="2025-04-01" --until="2025-5-31"

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:

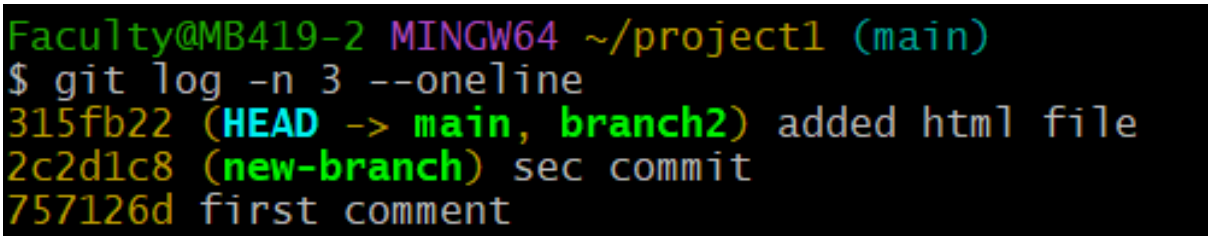
$git log -n 5



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:

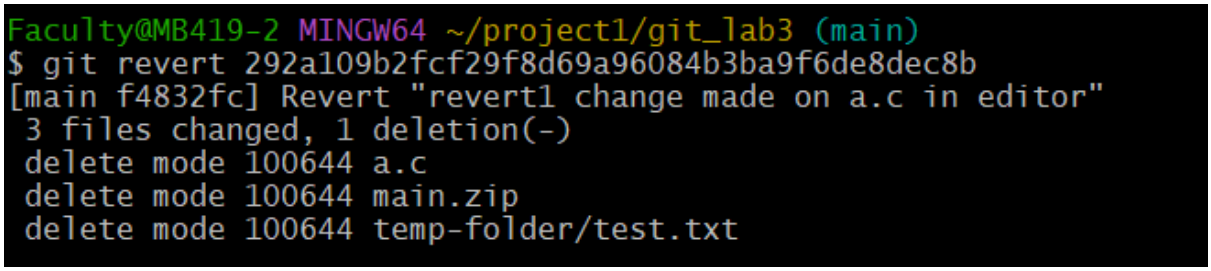
$git log -n 5 –oneline



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.**

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git log -n 1
commit f4832fca10ff4f2e0b38f5c56206b6c76544de66 (HEAD -> main)
Author: Supriya.hs <supriyahs.gat@gmail.com>
Date:   Thu May 22 09:54:29 2025 +0530

    Revert "revert1 change made on a.c in editor"

    This reverts commit 292a109b2fcf29f8d69a96084b3ba9f6de8dec8b.
```

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

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

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

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git config --global alias.st status
```

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git config --global alias.lg "log --oneline"
```

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3 (main)
$ git config --global --list
core.editor="C:\Users\Faculty\AppData\Local\Programs\Microsoft VS Code\bin\code" --wait
color.ui=true
user.email=supriyahs.gat@gmail.com
user.name=Supriya.hs
alias.st=status
alias.lg=log --oneline
```

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"

```
Faculty@MB419-2 MINGW64 ~
$ cd project1

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git branch
  branch2
* main
  new-branch

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ echo "Initial" > file.txt

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git add .
```

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git commit -m "Initial commit"
[main 56320d8] Initial commit
 4 files changed, 3 insertions(+)
 create mode 160000 6b
 create mode 100644 P_TT.png
 create mode 100644 file.txt
 create mode 160000 git_lab3
```

 # Create feature branch

git checkout -b feature

echo "Feature work" >> file.txt

git add .

git commit -m "Work on feature"

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git checkout -b feature
Switched to a new branch 'feature'

Faculty@MB419-2 MINGW64 ~/project1 (feature)
$ echo "Feature work" >> file.txt

Faculty@MB419-2 MINGW64 ~/project1 (feature)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by
 CRLF the next time Git touches it

Faculty@MB419-2 MINGW64 ~/project1 (feature)
$ git commit -m "Work on feature"
[feature d809401] Work on feature
 1 file changed, 1 insertion(+)
```

 # Switch back and merge

git checkout main

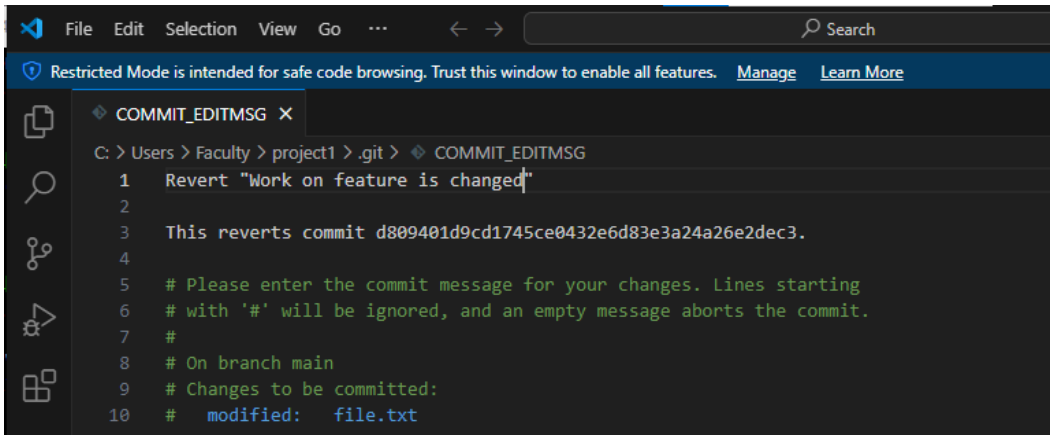
git merge feature -m "Merge feature into main"

```
Faculty@MB419-2 MINGW64 ~/project1 (feature)
$ git checkout main
M       6b
Switched to branch 'main'

Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git merge feature -m "Merge feature into main"
Updating 56320d8..d809401
Fast-forward (no commit created; -m option ignored)
 file.txt | 1 +
 1 file changed, 1 insertion(+)
```

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

```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git log --oneline
d809401 (HEAD -> main, feature) Work on feature
56320d8 Initial commit
315fb22 (branch2) added html file
2c2d1c8 (new-branch) sec commit
757126d first comment
```

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




```
Faculty@MB419-2 MINGW64 ~/project1 (main)
$ git revert -m 1 d809401
[main 0632d57] Revert "Work on feature is changed"
 1 file changed, 1 deletion(-)
```

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"

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git checkout -b test-branch
Switched to a new branch 'test-branch'

Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (test-branch)
$ echo "Hello" > file.txt

Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (test-branch)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by
 CRLF the next time Git touches it

Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (test-branch)
$ git commit -m "Added file.txt"
[test-branch 15065a3] Added file.txt
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

2. Switch to another branch and delete test-branch**:**
$git checkout main
$git branch -D test-branch

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (test-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git branch -D test-branch
Deleted branch test-branch (was 15065a3).
```

3. Recover with git reflog**: $git reflog**

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git reflog
cc01fe4 (HEAD -> main, origin/main, origin/Test, origin/HEAD) HEAD@{0}: checkout: m
oving from test-branch to main
15065a3 HEAD@{1}: commit: Added file.txt
cc01fe4 (HEAD -> main, origin/main, origin/Test, origin/HEAD) HEAD@{2}: checkout: m
oving from main to test-branch
cc01fe4 (HEAD -> main, origin/main, origin/Test, origin/HEAD) HEAD@{3}: clone: from
 https://github.com/SupriyaHS4221/git_lab3.git
```

4. Restore the branch**: $git checkout -b test-branch 15065a3**

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git checkout -b test-branch 15065a3
Switched to a new branch 'test-branch'
```

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**

```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ touch temp1.txt temp2.log notes.md

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ mkdir tempdir

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ touch tempdir/file1.txt

Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        notes.md
        temp1.txt
        temp2.log
        tempdir/

nothing added to commit but untracked files present (use "git add" to track)
```

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

```
Faculty@MB419-2 MINGW64 ~/project1/6b (main)
$ git clean -n
Would remove notes.md
Would remove temp1.txt
Would remove temp2.log
```

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

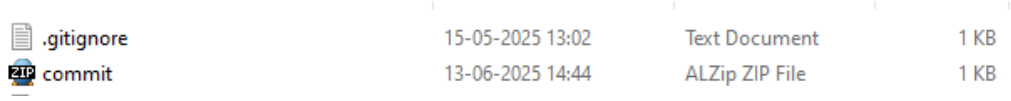
Step 1: Open Terminal or Command Prompt: **cd path\to\your\repository**
Step 2: Run the Archive Command: **$git archive --format=tar --output=main.tar main**

Create Archive of a Specific Commit
Step 1: Find the Commit Hash: **$git log --oneline**

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git lg
cc01fe4 (HEAD -> main, origin/main, origin/Test, origin/HEAD) Create Supriya.txt
233def5 Create pull
4275ab0 Create test.txt
586f353 Initial commit
```

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

```
Faculty@MB419-2 MINGW64 ~/project1/git_lab3/git_lab3 (main)
$ git archive --format=zip --output=commit.zip 233def5
```



9. 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:

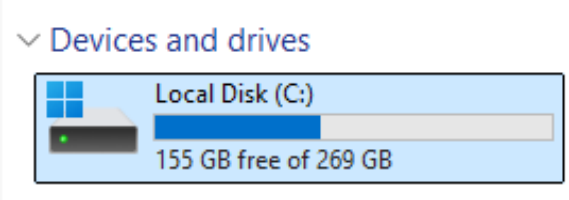
- **CI/CD:** Jenkins supports Continuous Integration and Continuous Deployment, allowing developers to integrate code changes frequently and automate the deployment of applications.
- **Plugins:** Jenkins has a vast library of plugins that can extend its capabilities. These plugins integrate Jenkins 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

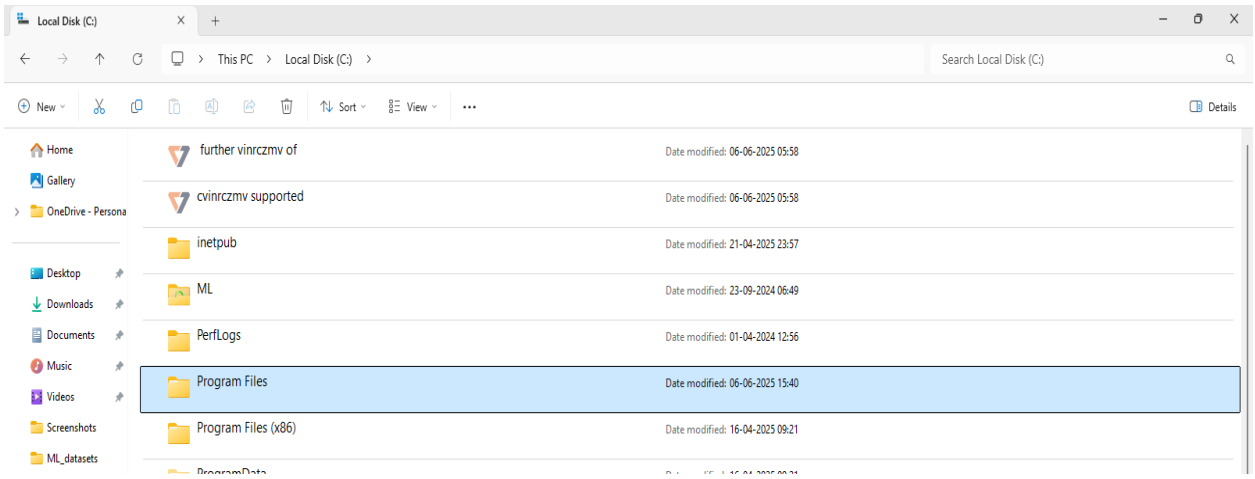
Jenkins can be installed on local machines, on a cloud environment, or even in containers.

Installation of Jenkins on Windows:

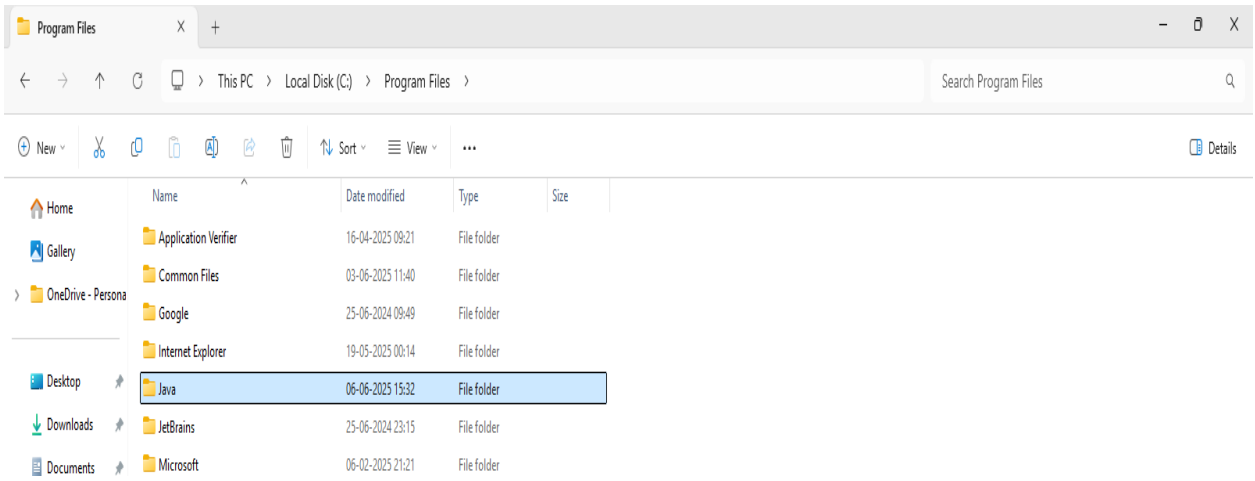
Step 1: check for program files in C: drive



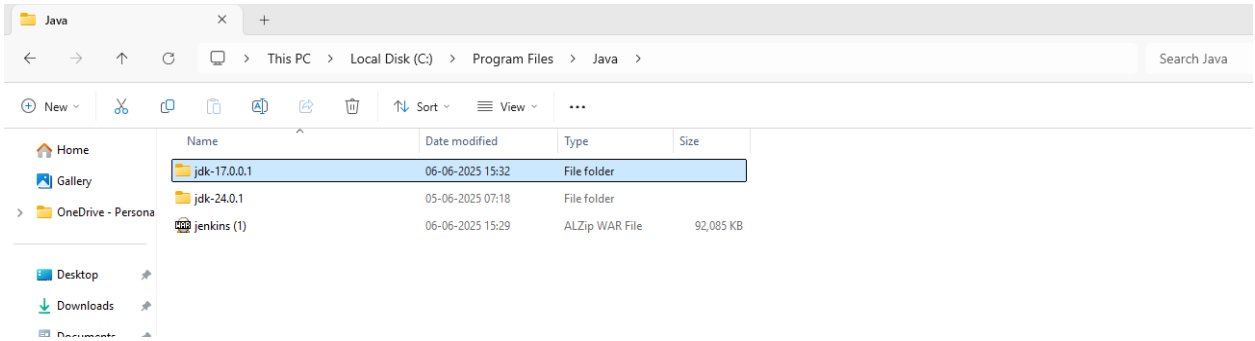
Step 2:



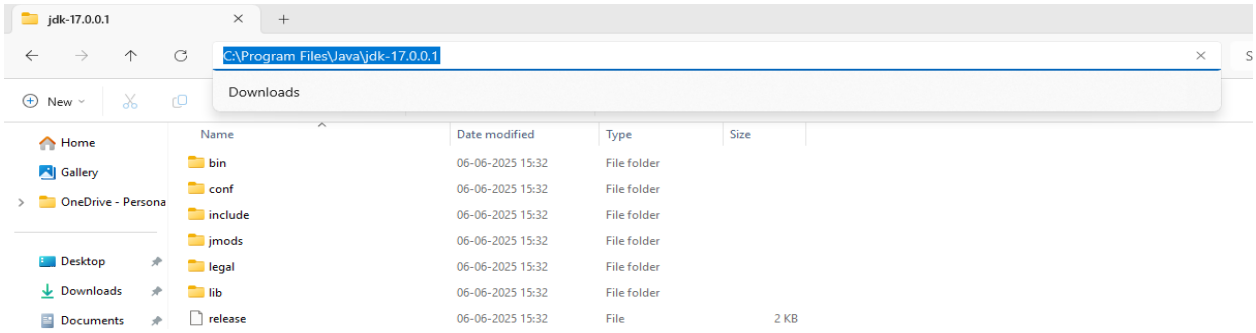
Step 3: Check for Java Folder



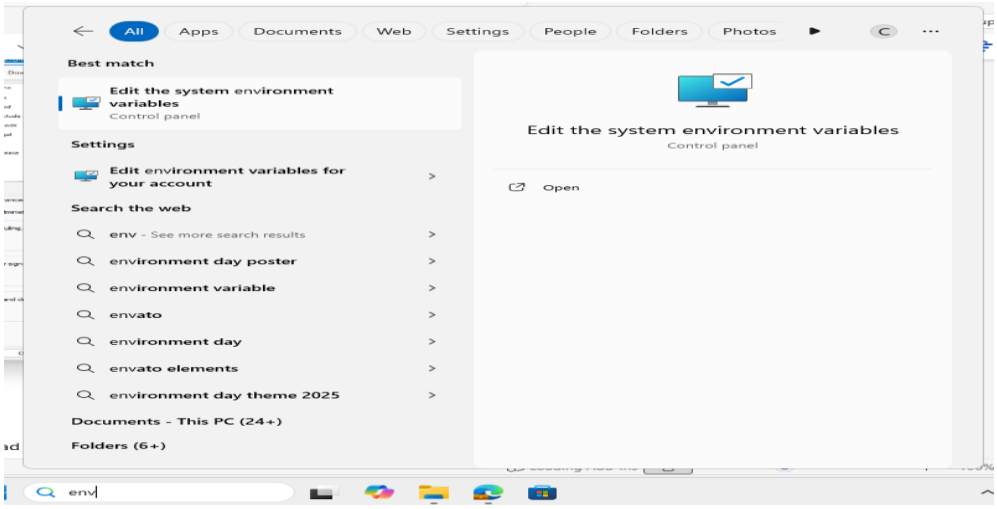
Step 4: Extract the Jdk 17 zip file here



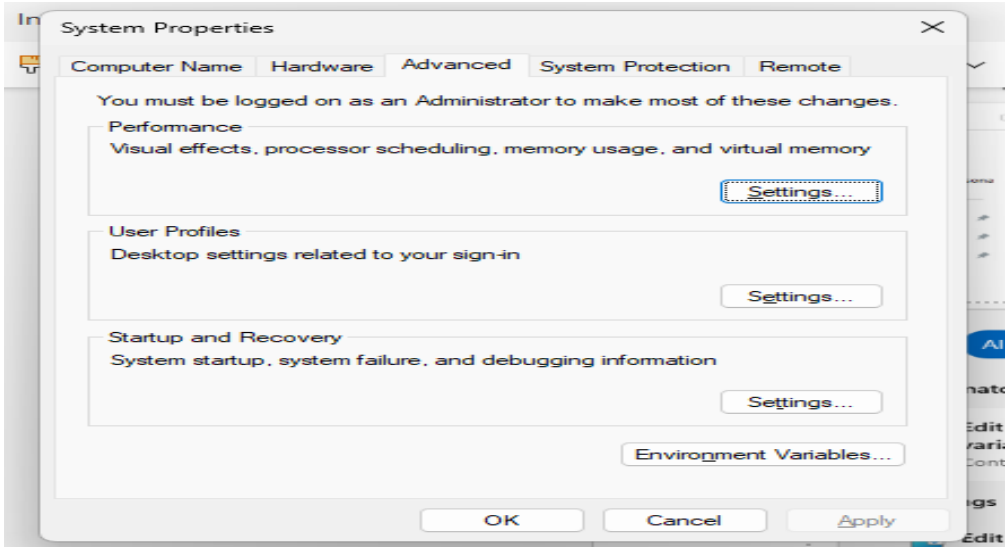
Step 5: copy the path of Jdk file



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



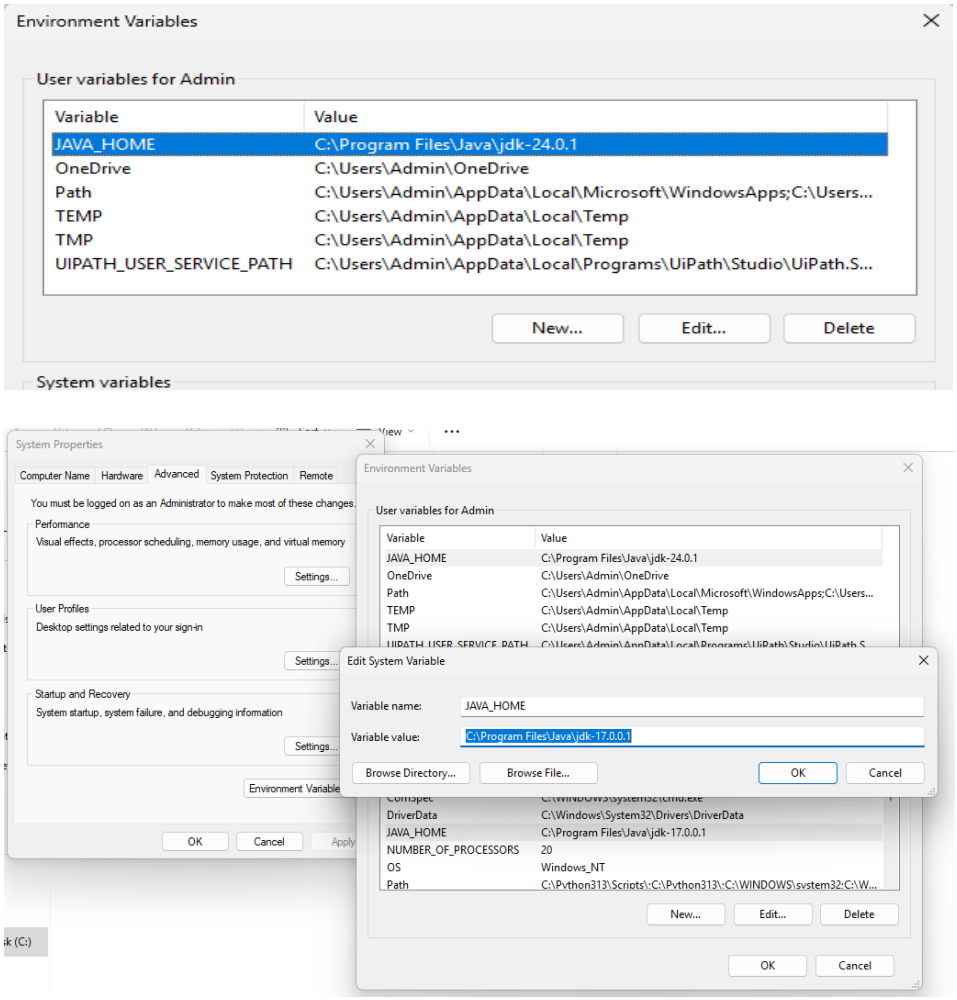
Step 7: In the System Properties window, click on Environment Variables.



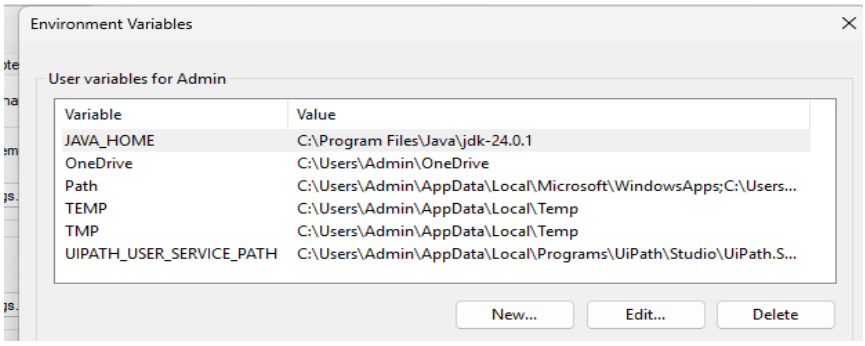
Step 8: Set JAVA_HOME->Under System variables, click New.

Variable Name**: JAVA_HOME**

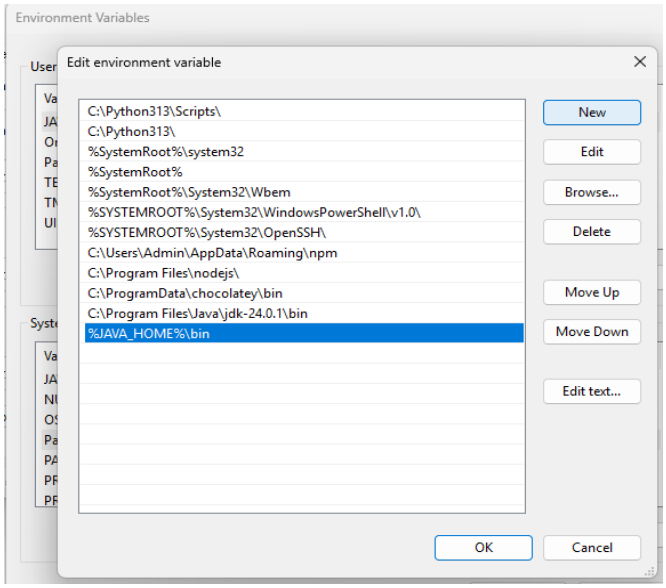
Variable Value: path to your JDK folder (e.g., C:\Program Files\Java\jdk-17)->Click OK.



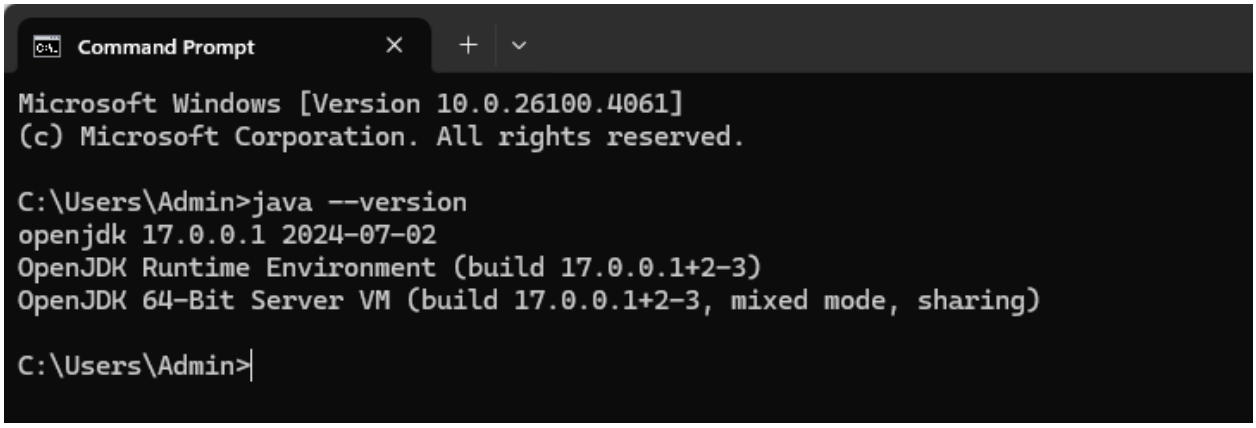
Step 9: Path variable->click on edit



Step 10: click on new-> write %JAVA_HOME%\bin->click ok



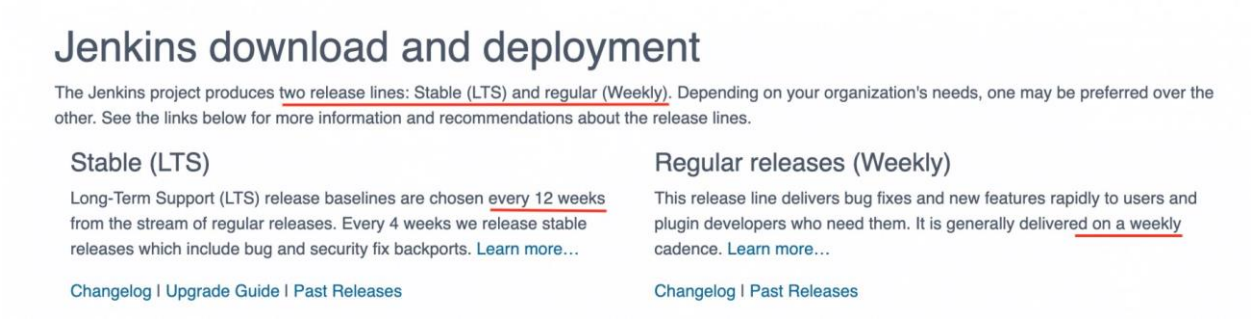
Step 11: Go to to command prompt and check for java --version



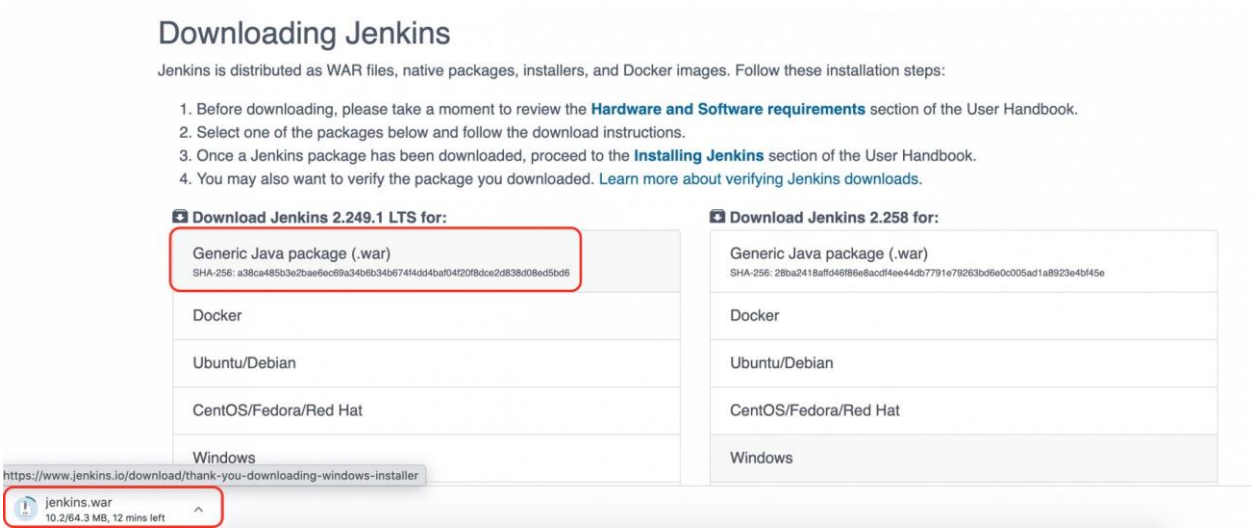
Download Jenkins Installer for Windows

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:

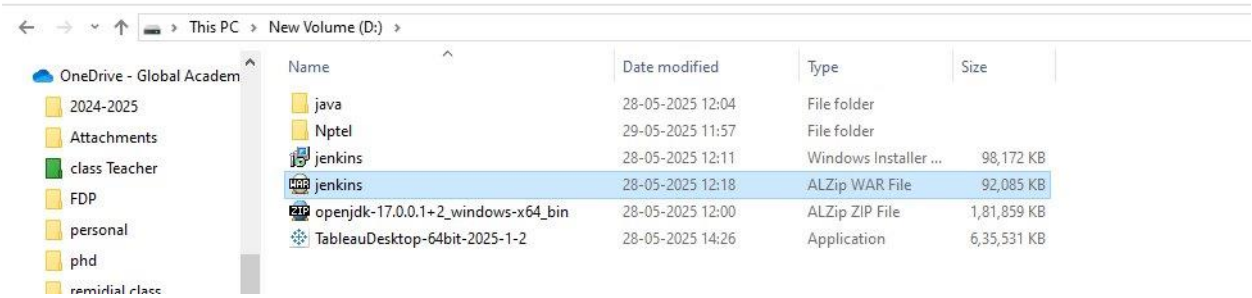


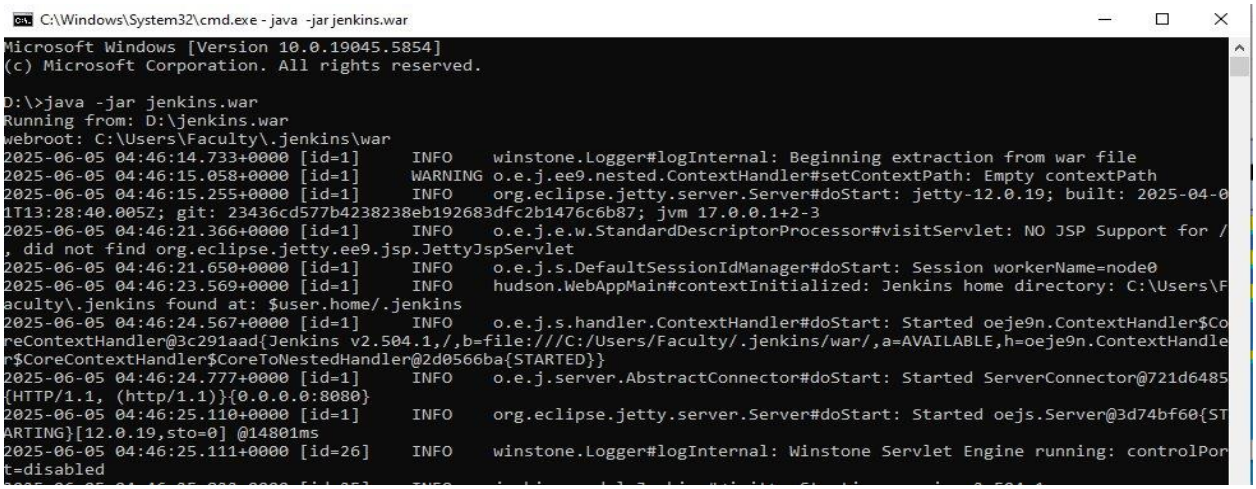
Step 3: Thirdly, we will see the list of operating systems for which Jenkins is available as an installer. Based on the operating system, we can select the corresponding option. Here, we are going with **Generic Java Package (.war)** file, a generic file that can set up Jenkins on all the operating systems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



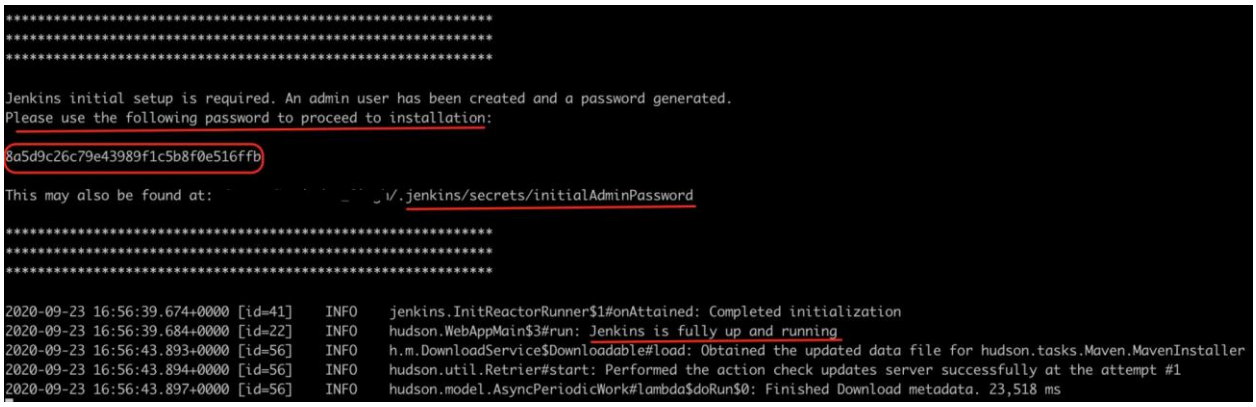
Step 4: Fourthly, place this jenkins.war file in any directory in your machine. Now open the command prompt and go to that directory and type the below command:

Java –jar jenkins.war –y

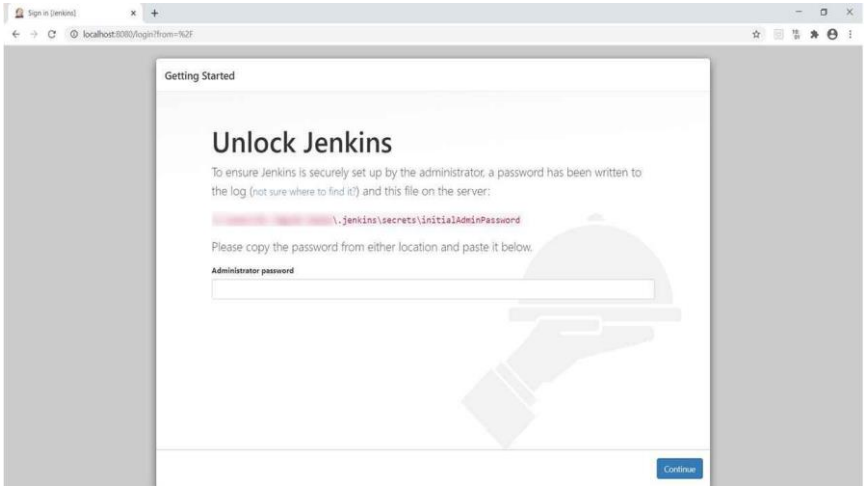




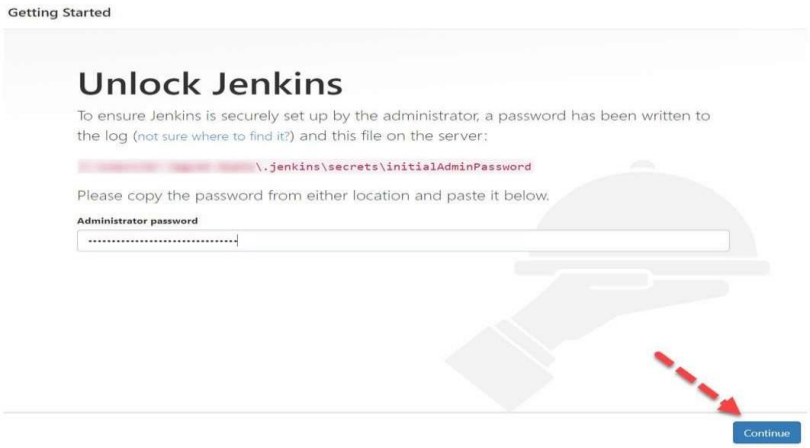
When Jenkins is fully up and running One time Password will be generated, copy the password



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

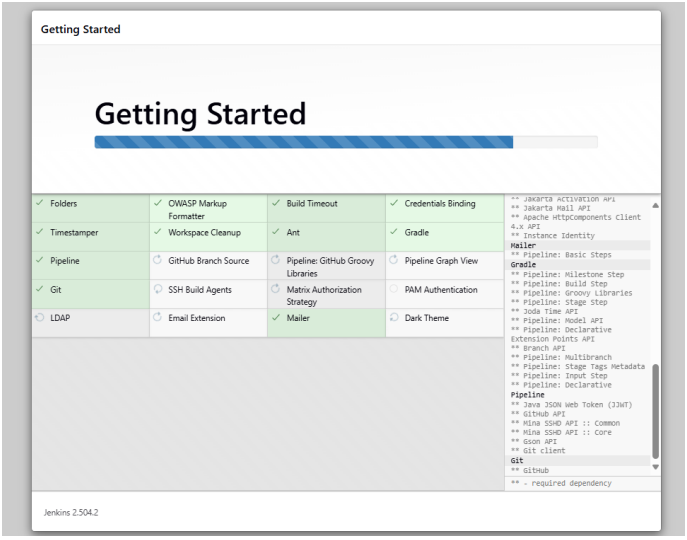


Step 6*:* Here, we need to put the password that we copied from the previous step and click the continue button.



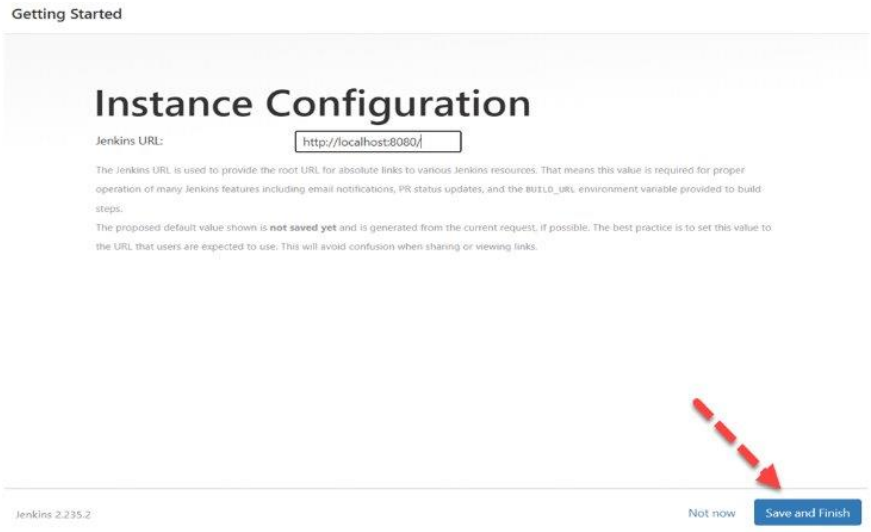
Step 7: will be redirected to the page to suggest you install suggested plugins. After that, click on “Install suggested plugins”. Kindly note that if users want to install only selected plugins required, it is recommended to select the option " Select plugins to install".

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

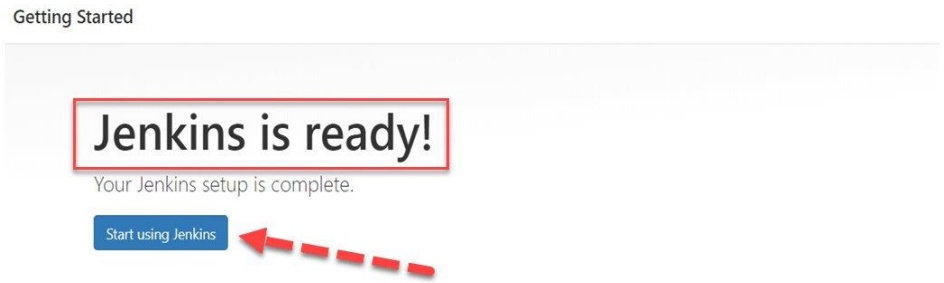


Step 9: Finally, after installation of all suggested plugins, you will be redirected to the User Account Page like below:

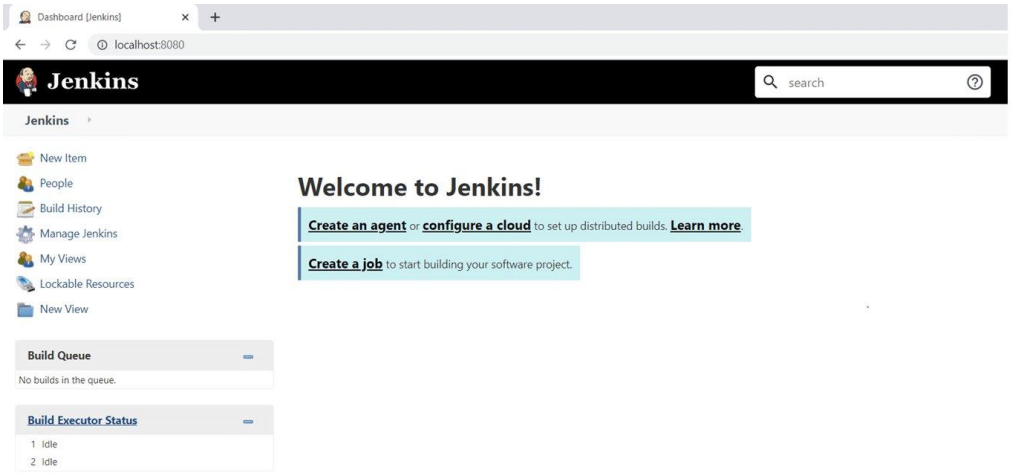
Step 10: After clicking on the ***"Save and Continue Button",*** you will be redirected to the instance configuration screen. Here click on the ***"Save and Finish Button".***



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.

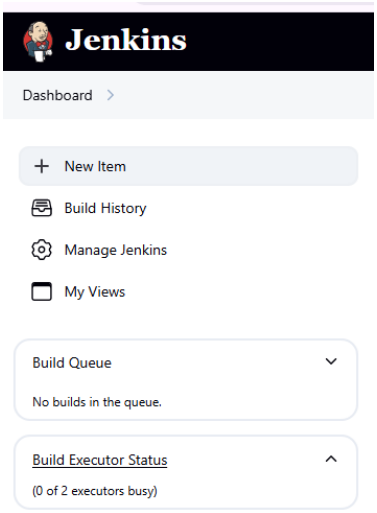


Step 12: After clicking on the ***"Start using Jenkins"*** button, you will be redirected to the *Jenkins* Dashboard.

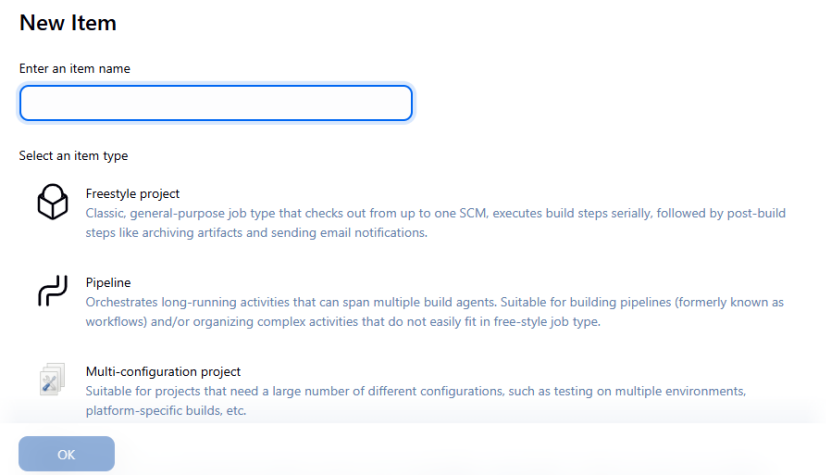


Create a New Job

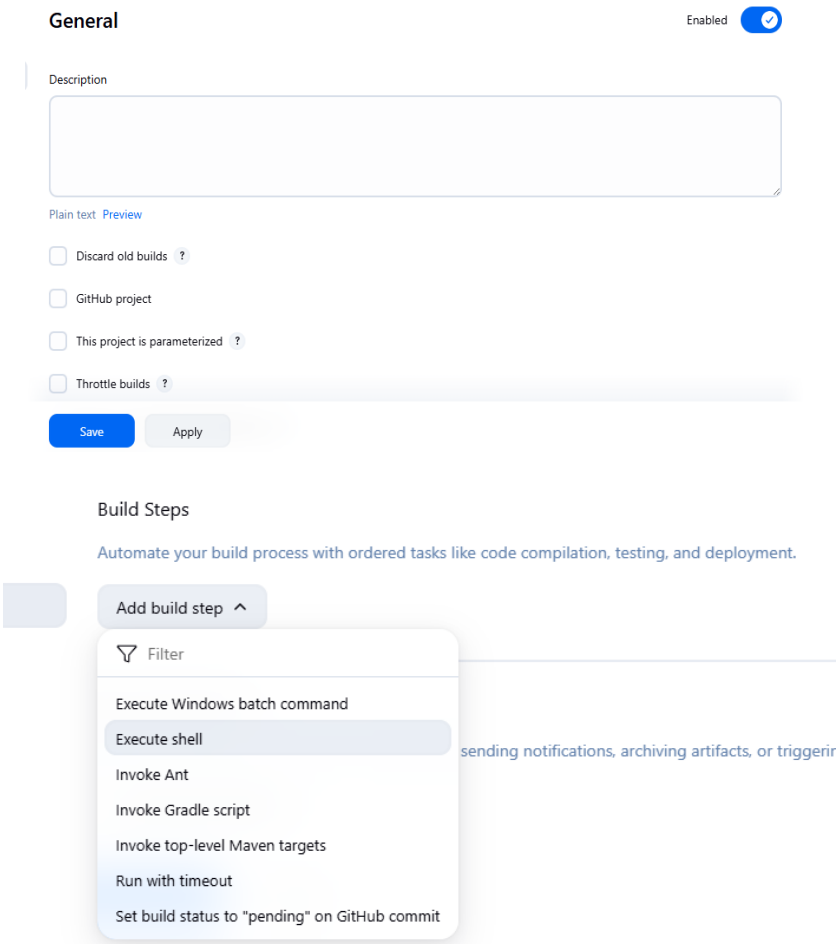
- Click **"New Item"**



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

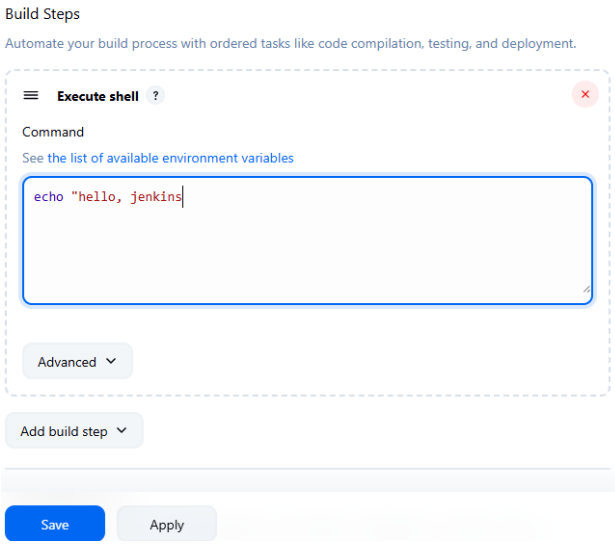


Step 2: Configure Job

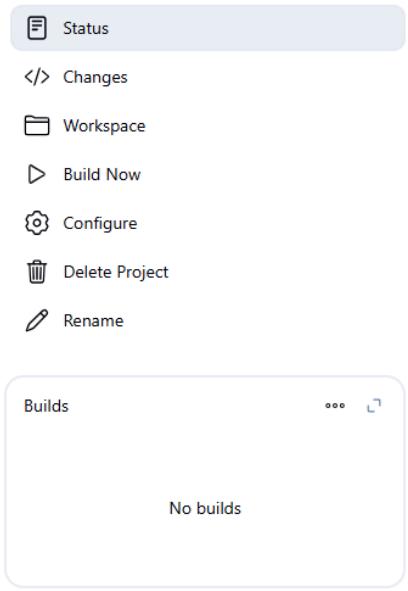


Step 2: Save and Build

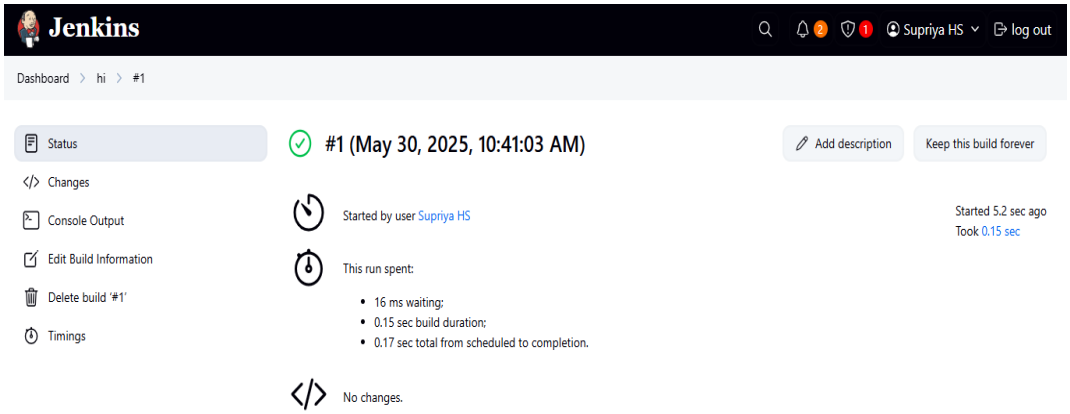
- Click **Save**



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



Status of job



Click on console output:

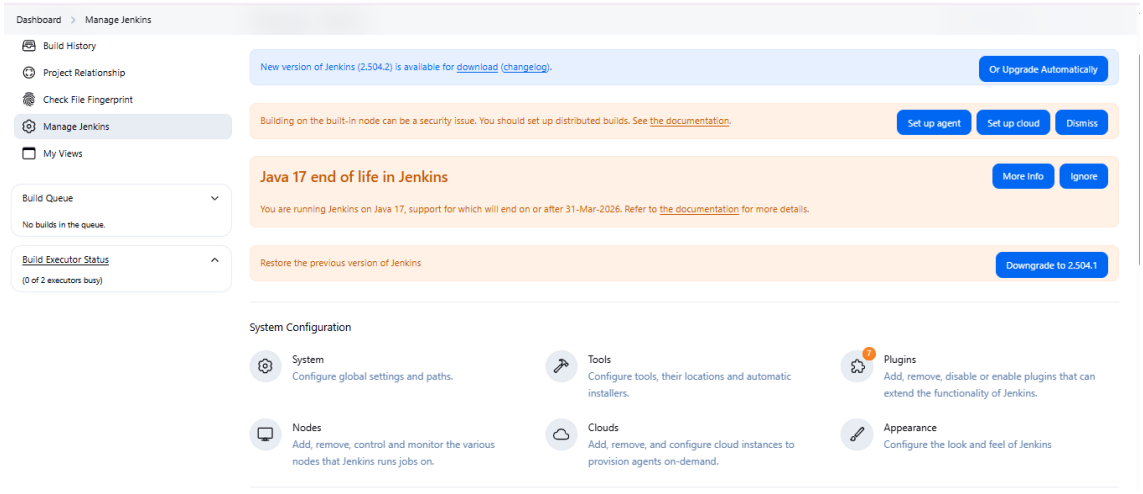
10. Configure a Jenkins Freestyle project to integrate with a GitHub repository using Source Code Management (SCM), and build based on changes in the Git repository.

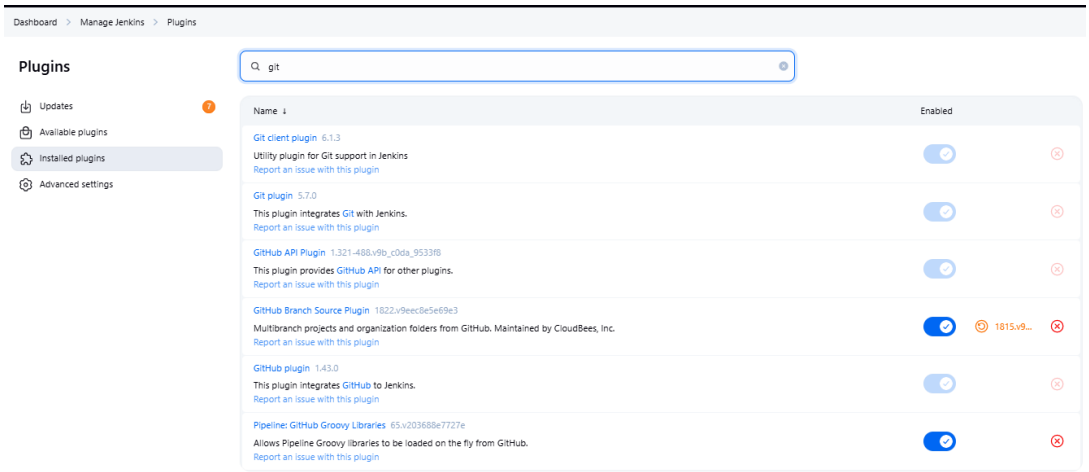
Solution:

Configure GitHub with Jenkins:

Step 1: Install Required Jenkins Plugins

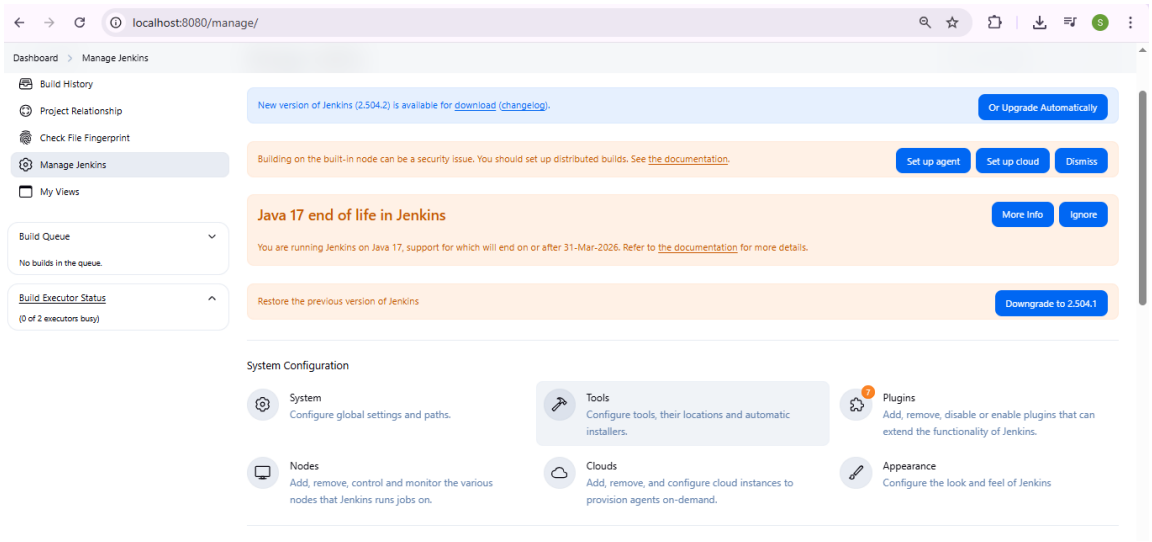
1. Go to Manage Jenkins → Manage Plugins
2. Under **Available** or **Installed** tabs, ensure the following are installed:
    - **Git plugin**
    - **GitHub plugin**
    - **GitHub Integration Plugin**
    - **GitHub Branch Source**
    - **Pipeline (if using scripted pipeline)**

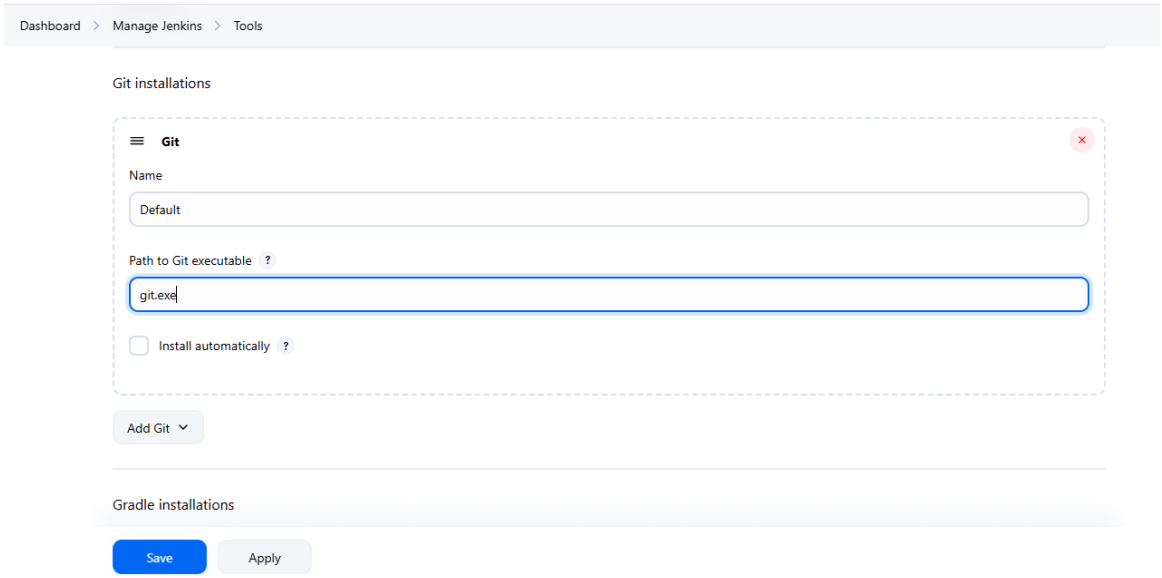




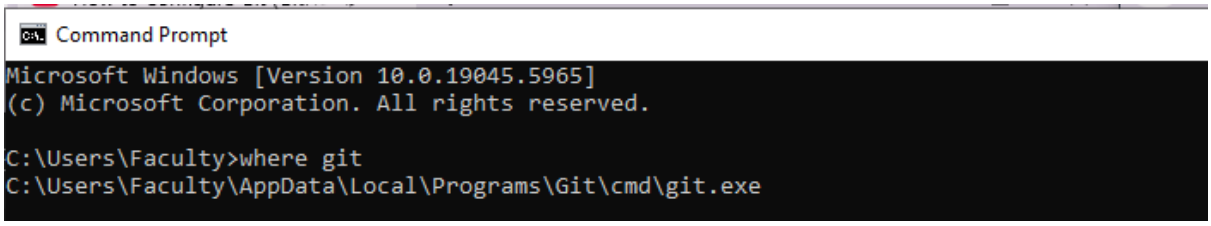
Step 2: Add local git.exe file Jenkins

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





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

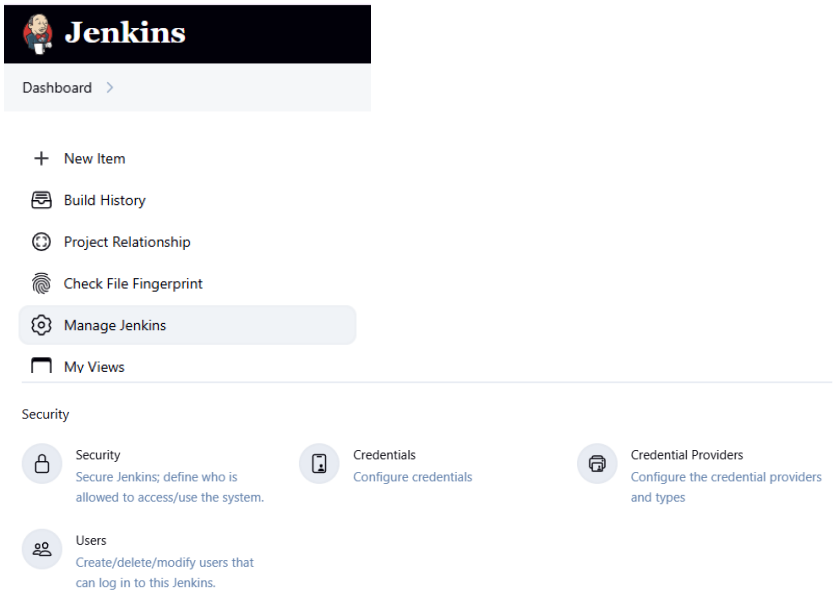


Paste the path click on Apply and save

Create Credentials in Jenkins

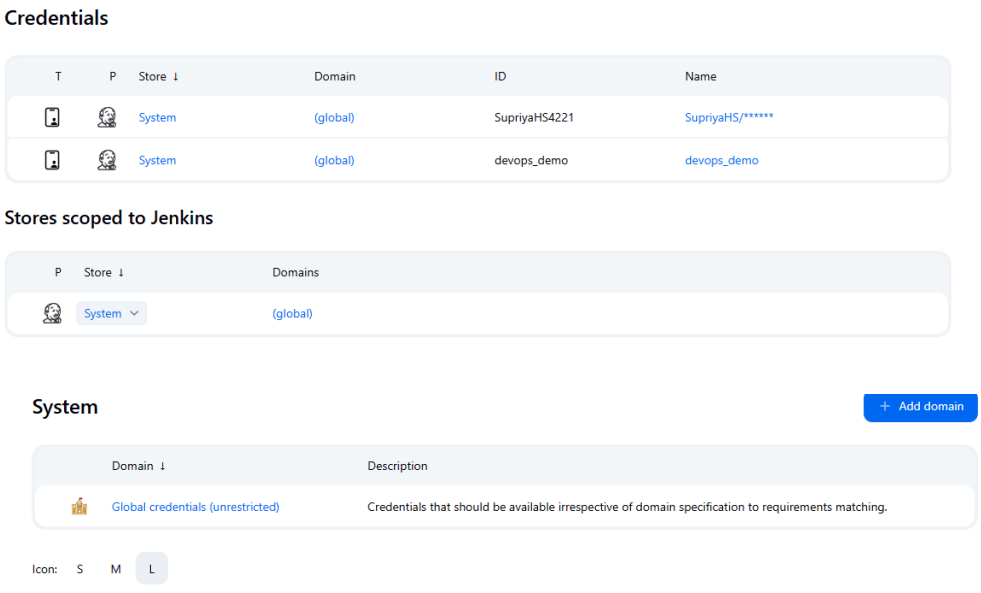
Step 1. Go to: http://localhost:8080 (or your Jenkins server URL) > Login with your credentials.

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

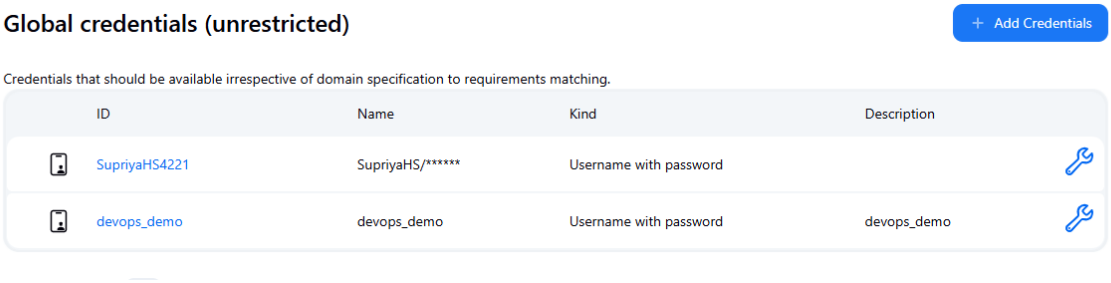


Step 3. Choose Scope

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



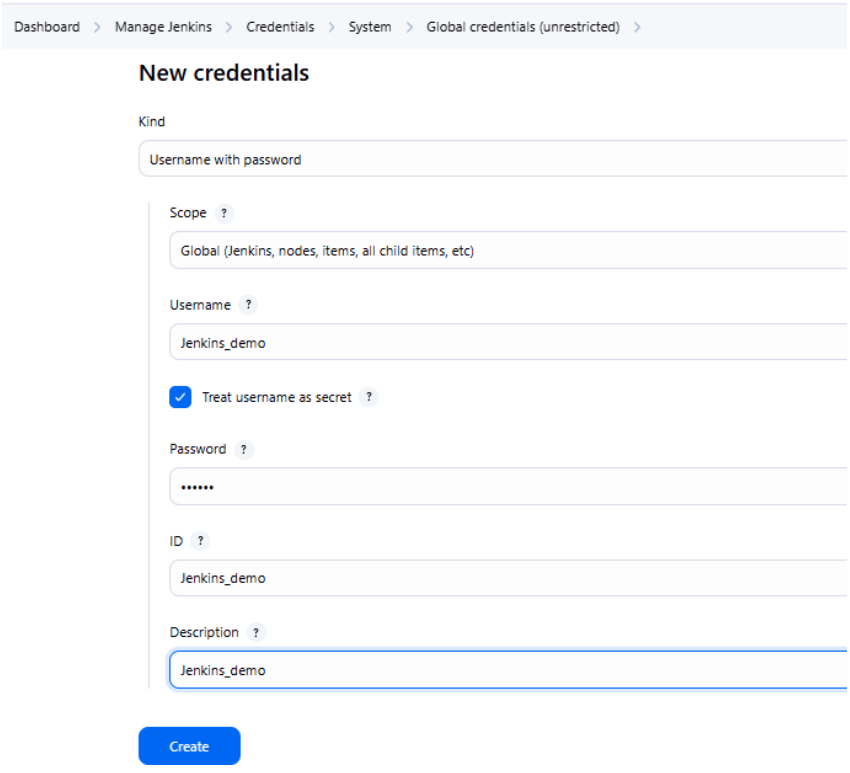
Step 4. On the left side, click **Add Credentials**.



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**.



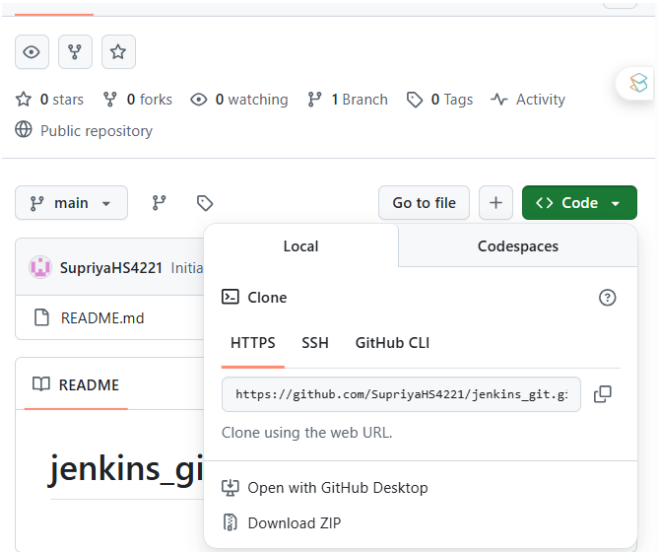
Create the new job in Jenkins

Step 1: **Create a new Freestyle project** in Jenkins named jenkins_git_project.

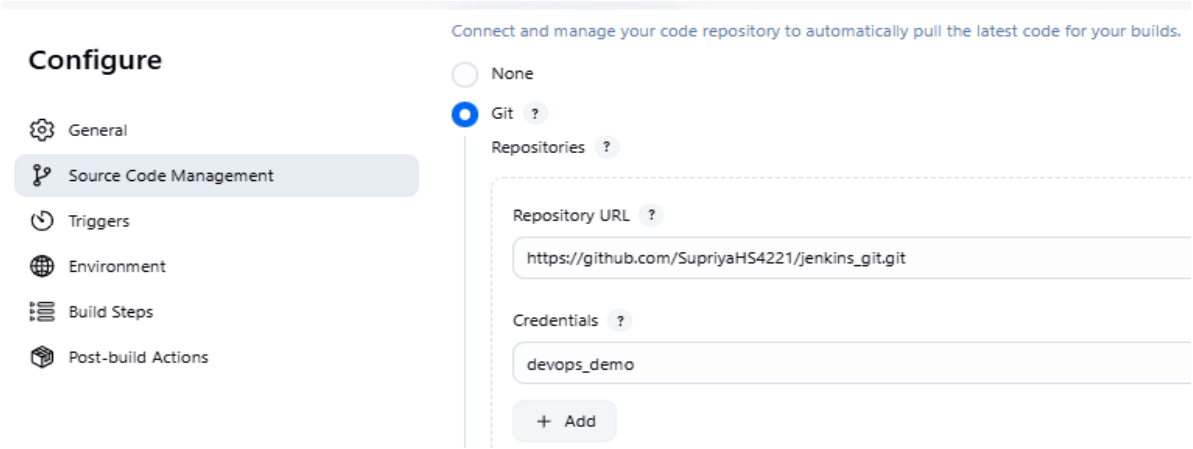
Step 2: In the **Source Code Management (SCM)** section:

- Select **Git**.

- Enter the GitHub repository URL:
  Example: https://github.com/YourUsername/jenkins_git.git



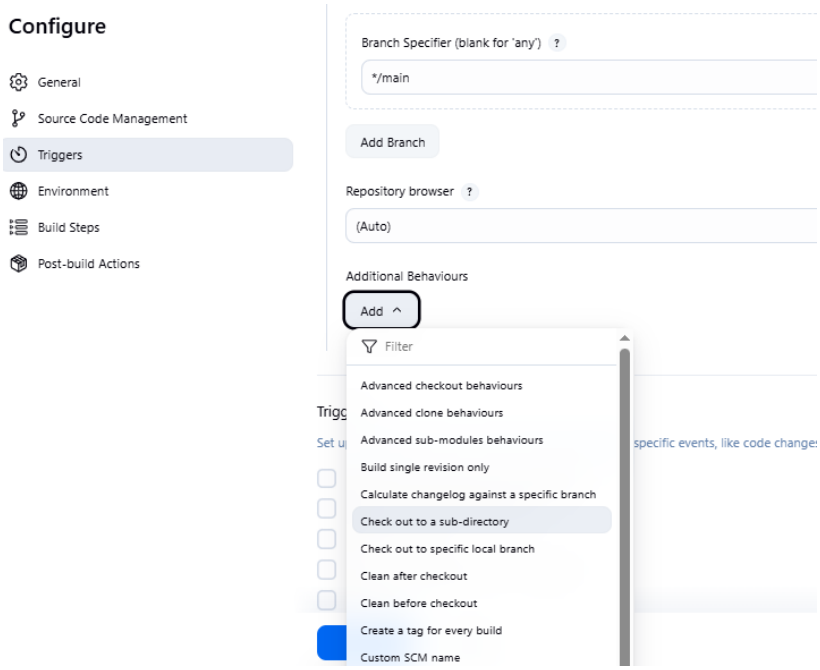
- Add Git credentials

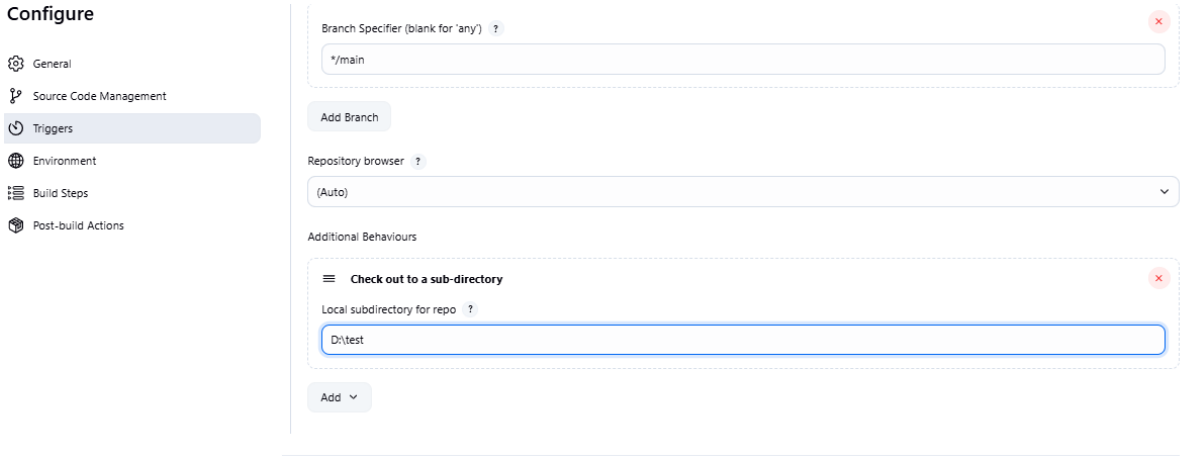


- Specify the **branch to build** (e.g., `*/main`).

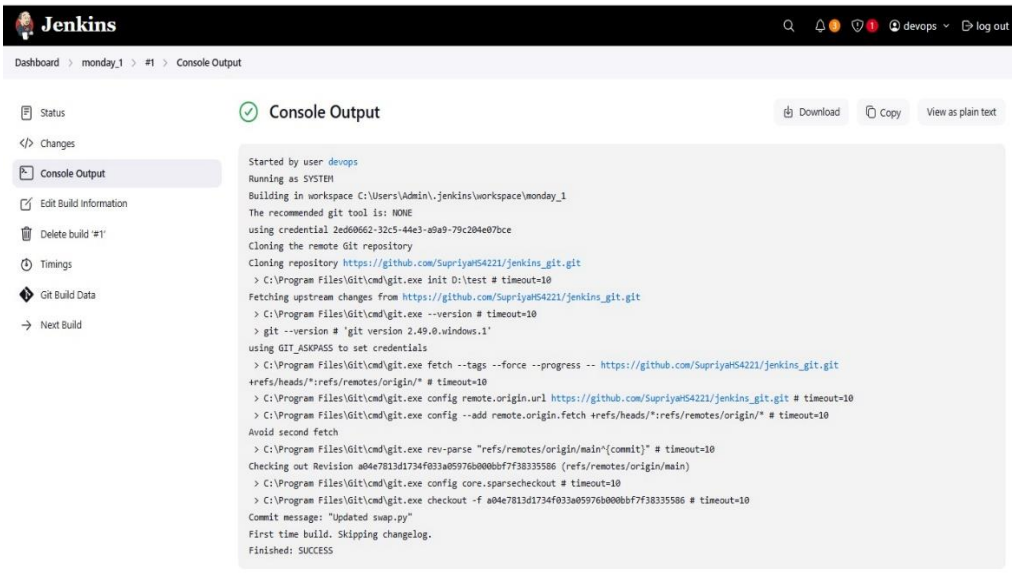
- Click on **"Add"** under **Additional Behaviours** and configure the following:

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



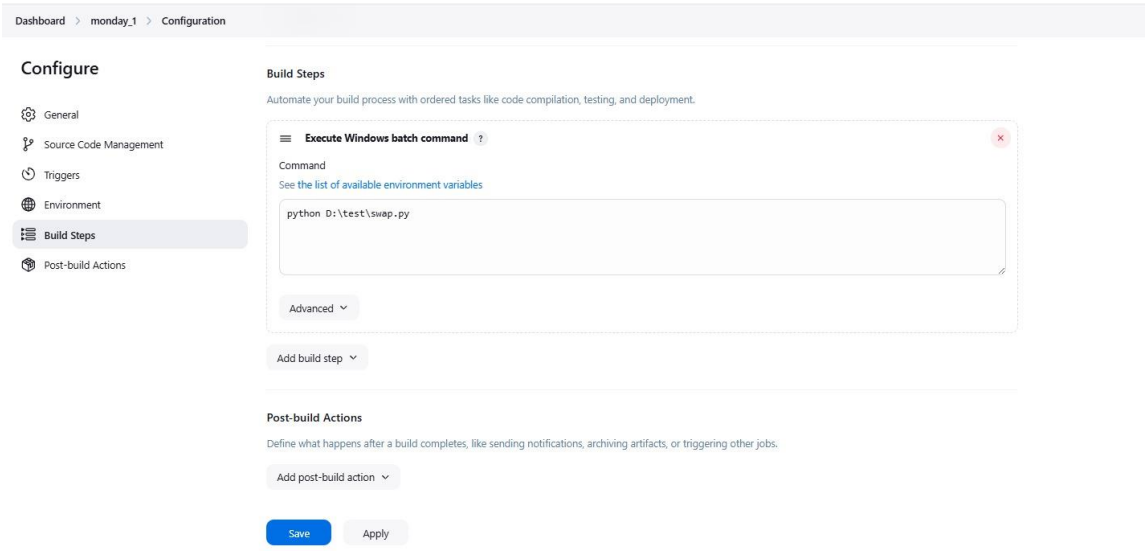


- Save the job and **run the build** manually to test.

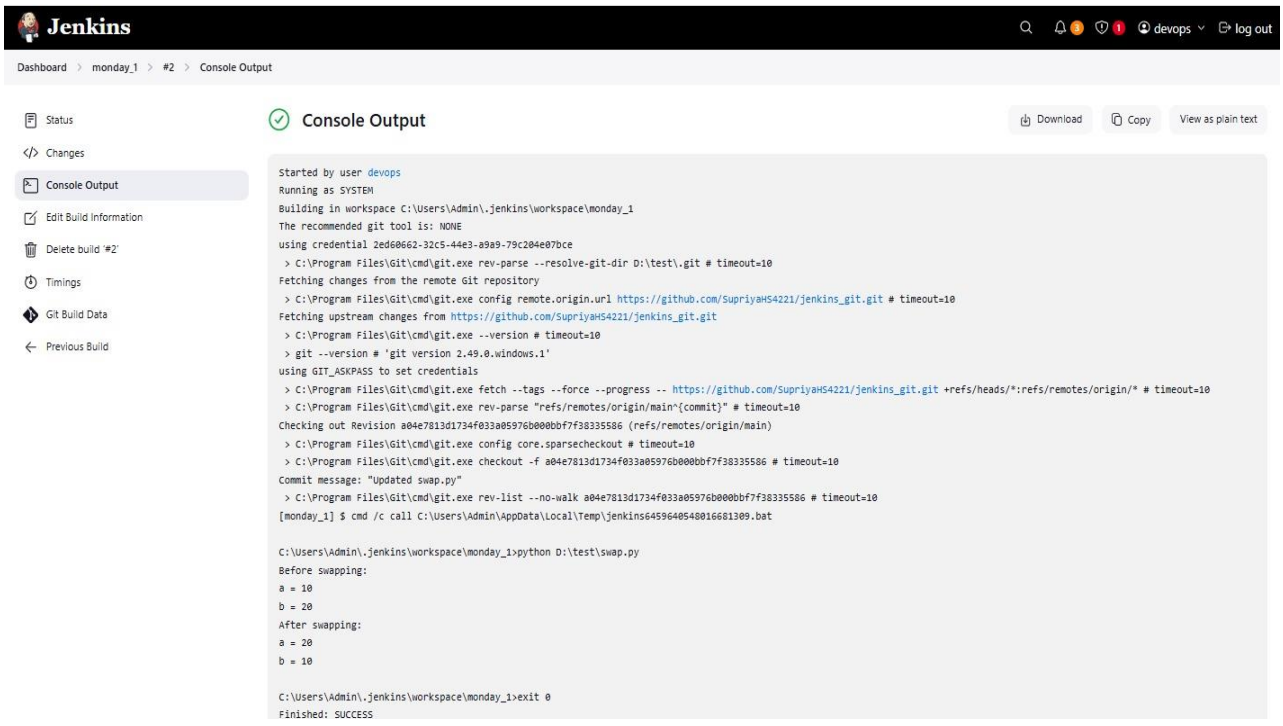


- Now go back to configure job, In the **Build section**, add a simple window batch command such as:

```
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.

2. **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.

3. **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.

4. **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.

5. **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.

6. **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.

7. **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).

8. **How do you create and switch to a new branch?**

$git checkout -b branch_name

9. . **How do you connect a local repo to GitHub?**

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

$git push -u origin master

10. **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.

11. What does git reset --hard HEAD~1 do?

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

12. How can you undo a git push?

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

13. What's the difference between git merge and git rebase?

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

14. How do you resolve a merge conflict?

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

15. 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.

16. Can you commit without staging? How?

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

17. How do you squash multiple commits into one?

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

18. 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.

19. How can you make a branch from a previous commit?

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

20. What is a detached HEAD state?

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

21. What triggers a Jenkins job automatically?

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

22. What if Jenkins fails due to a plugin error?

Remove or disable the plugin manually from the `JENKINS_HOME/plugins` directory.

23. What is the role of `workspace` in Jenkins?

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

24. What's the difference between `build now` and scheduled builds?

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