Program 6

Commands to push an file from local repo to remote repo using git and github

- **Pwd**………present working directory -to check where we are now
- **cd desktop**… change directory to desktop(we need to create an git folder in desktop)
- **mk dir gitproject**……..creating an directory for working with git you can give any name of your choice
- **cd gitproject**…… coming inside that git folder
- **git init** ....initializing git software{we need to execute git commands inside this folder only}
  <u>Now we are using vi editor to write source code.</u>
- **vi file1.sh**.....press i to insert after writing press esc and :wq (to save and quit)
- **ls….**to check the list of files
- **git add ***........adding to staging area(*..will add all the files to staging area)
- **git commit -m “commit message”**..............adding to local repo
  <u>To do congiguration we need to give our user name and email id of our git hub account</u>
- **git config --global user.name”username”**
- **git config --global user.email”emailid”**
- **git config --global –list**….. to check what configuration we have added..
- **git remote add alias name url**……….to add repository url
- **git remote -v**………. to check which repo is mapped
  <u>To push the file to remote repo.. master is the default branch which will be created when we create an repository</u>

- **git push alias name master**……alias name is the name which is short form or nick name which we had given while adding remote repo

Once we execute git push cmd **, when we are executing for first time we will get and small window poped up asking pat(personal access tocken) or you can give your email and password**

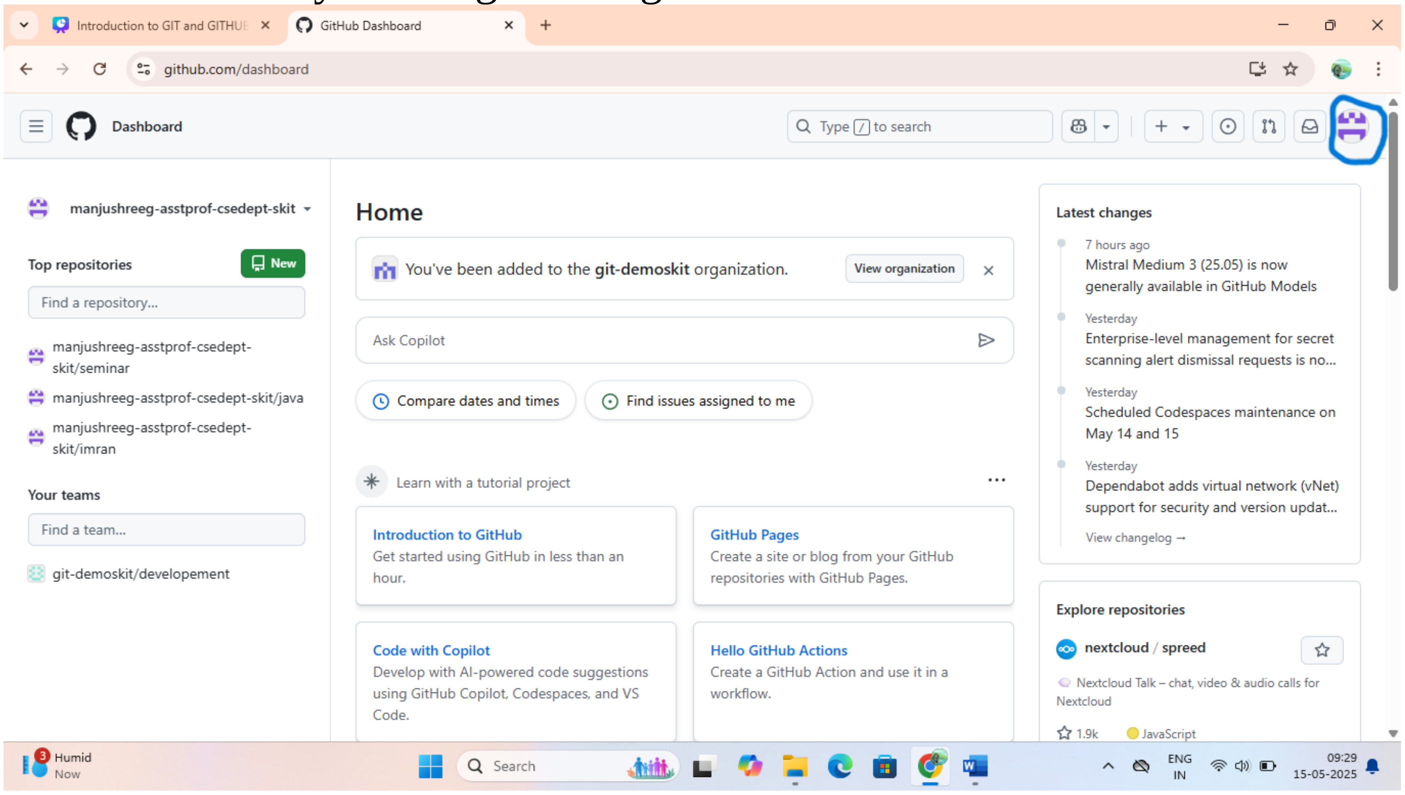
- **pat**

<u>Steps to generate PAT</u>

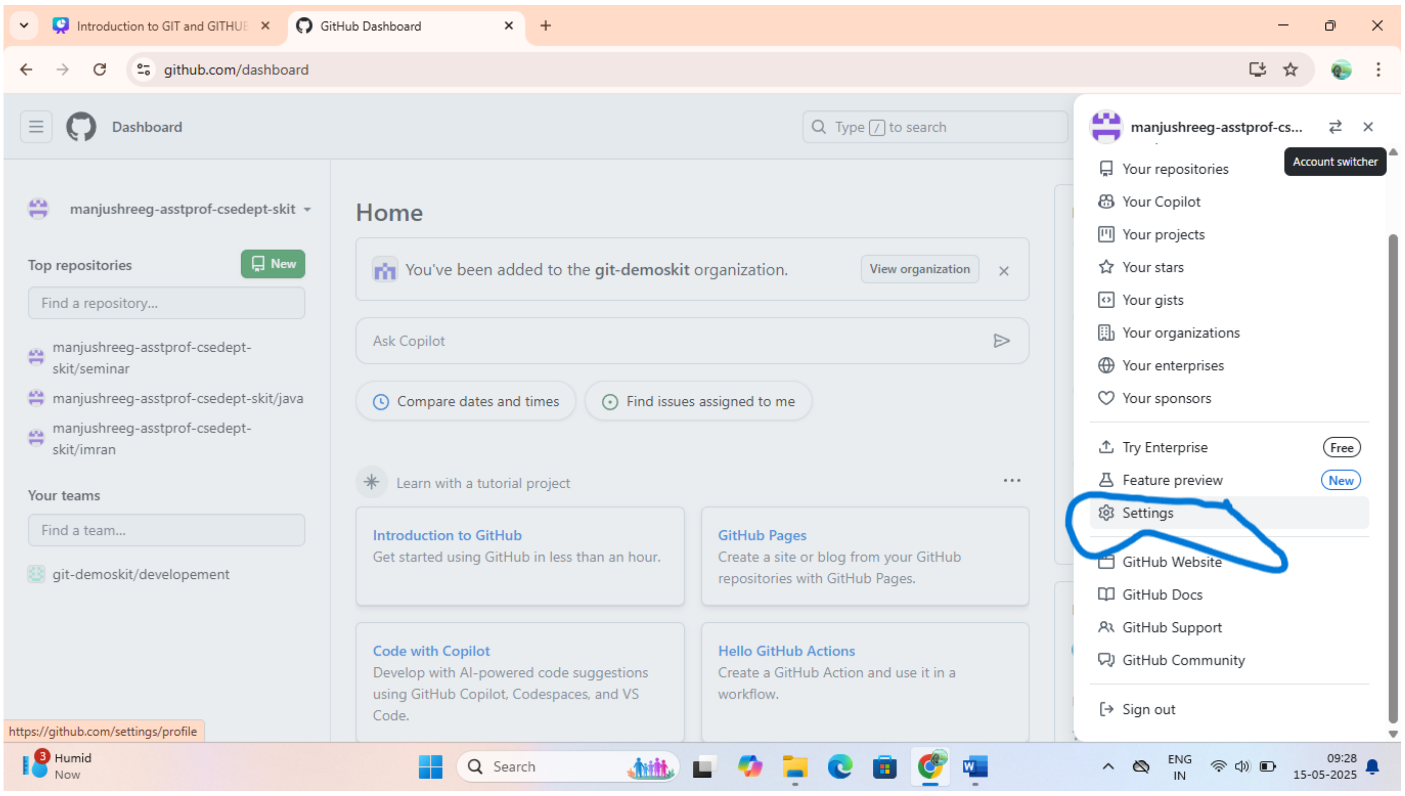
In your git hub account

Go to settings….developer settings….personal access token.. classic.. generate… give the duration of time which it will be active

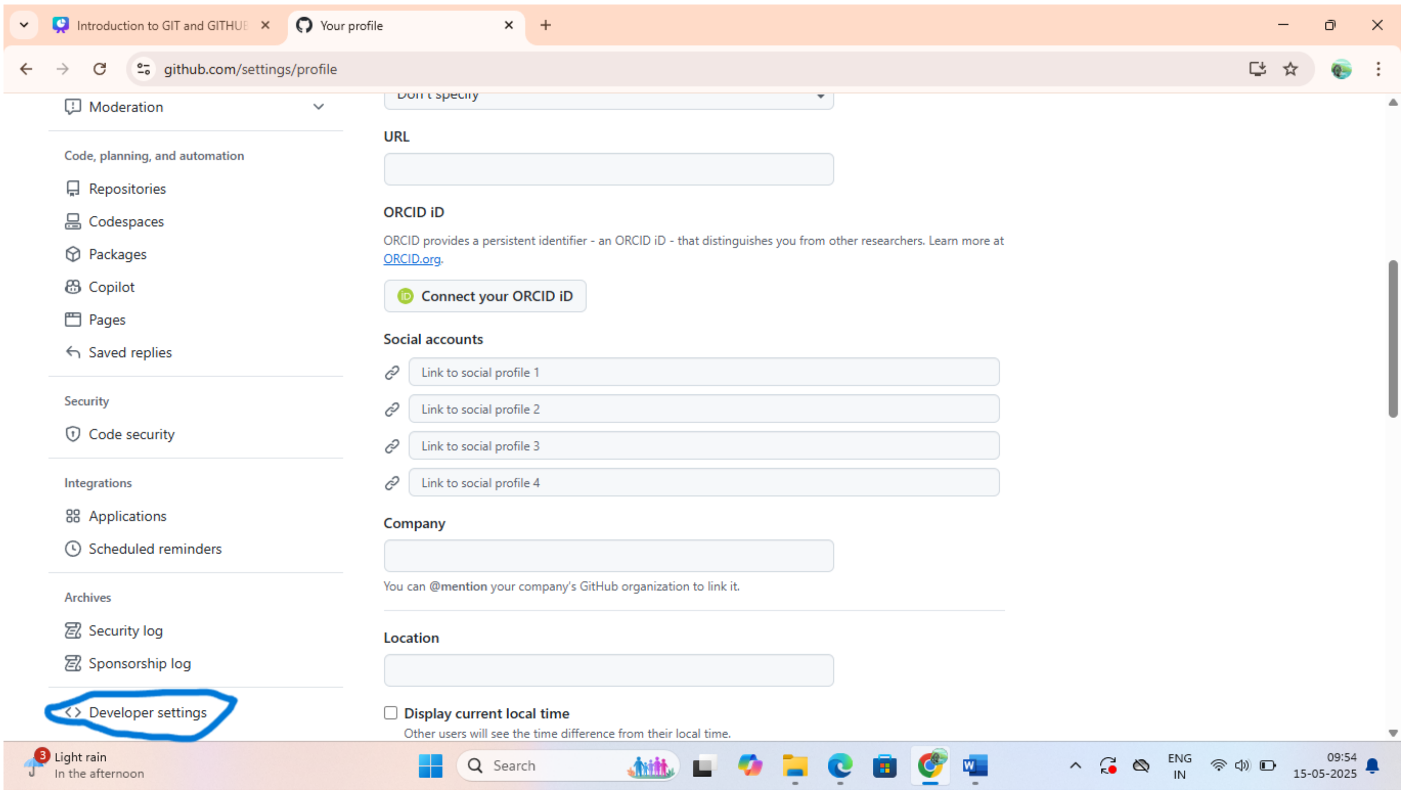
1.Click this icon you will get settings


Introduction to GIT and GITHUB
GitHub Dashboard
github.com/dashboard
Dashboard
Type / to search
manjushreeg-asstprof-csedept-skit
Top repositories
New
Find a repository...
manjushreeg-asstprof-csedept-skit/seminar
manjushreeg-asstprof-csedept-skit/java
manjushreeg-asstprof-csedept-skit/imran
Your teams
Find a team...
git-demoskit/developement
Home
You've been added to the **git-demoskit** organization.
View organization
Ask Copilot
Compare dates and times
Find issues assigned to me
Learn with a tutorial project
Introduction to GitHub
Get started using GitHub in less than an hour.
GitHub Pages
Create a site or blog from your GitHub repositories with GitHub Pages.
Code with Copilot
Develop with AI-powered code suggestions using GitHub Copilot, Codespaces, and VS Code.
Hello GitHub Actions
Create a GitHub Action and use it in a workflow.
Latest changes
7 hours ago
Mistral Medium 3 (25.05) is now generally available in GitHub Models
Yesterday
Enterprise-level management for secret scanning alert dismissal requests is no...
Yesterday
Scheduled Codespaces maintenance on May 14 and 15
Yesterday
Dependabot adds virtual network (vNet) support for security and version updat...
View changelog →
Explore repositories
nextcloud / spreed
Nextcloud Talk – chat, video & audio calls for Nextcloud
1.9k JavaScript
Humid Now
Search
ENG IN
09:29
15-05-2025

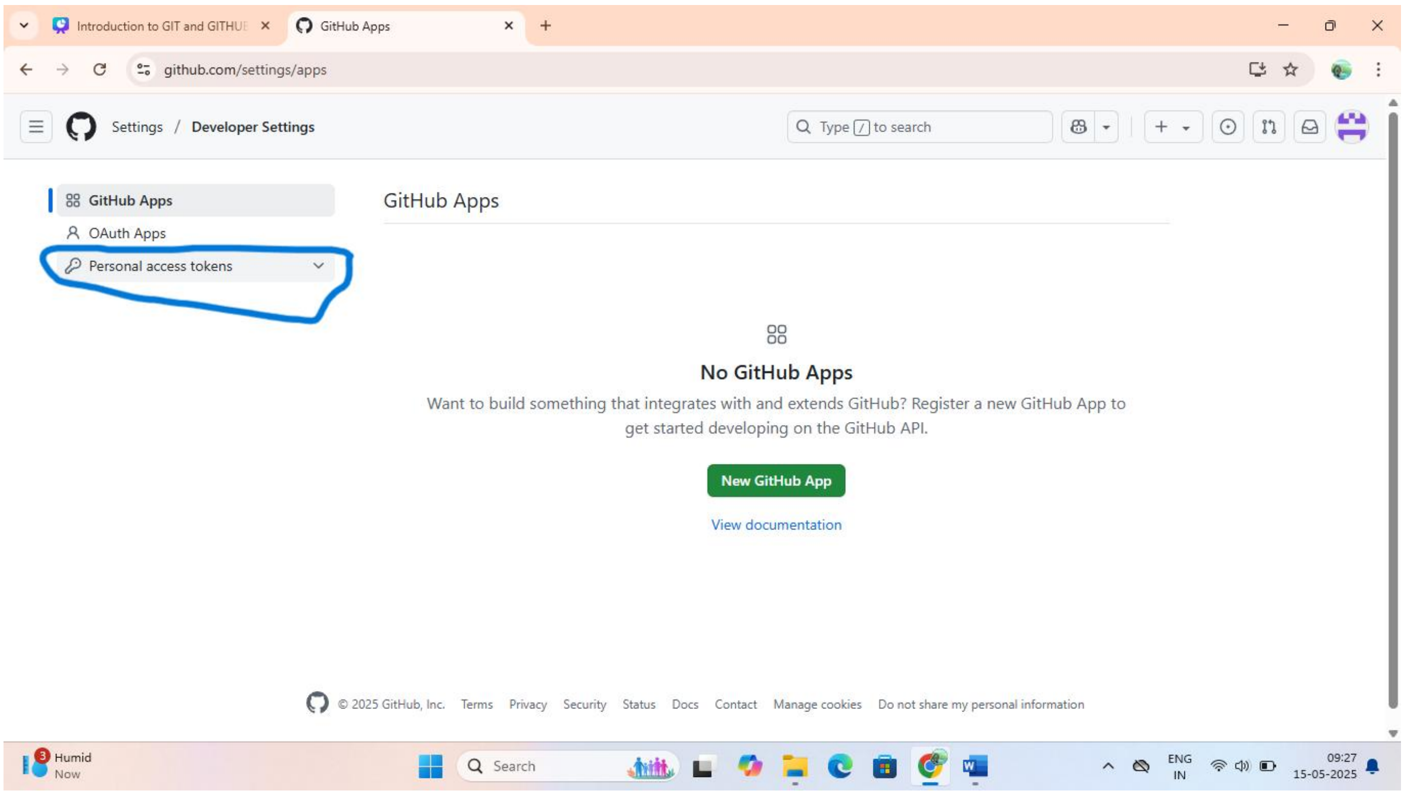
Go to settings



Select developer settings



Select **personal access token**…. Then **classic**…then **generate token**


github.com/settings/apps

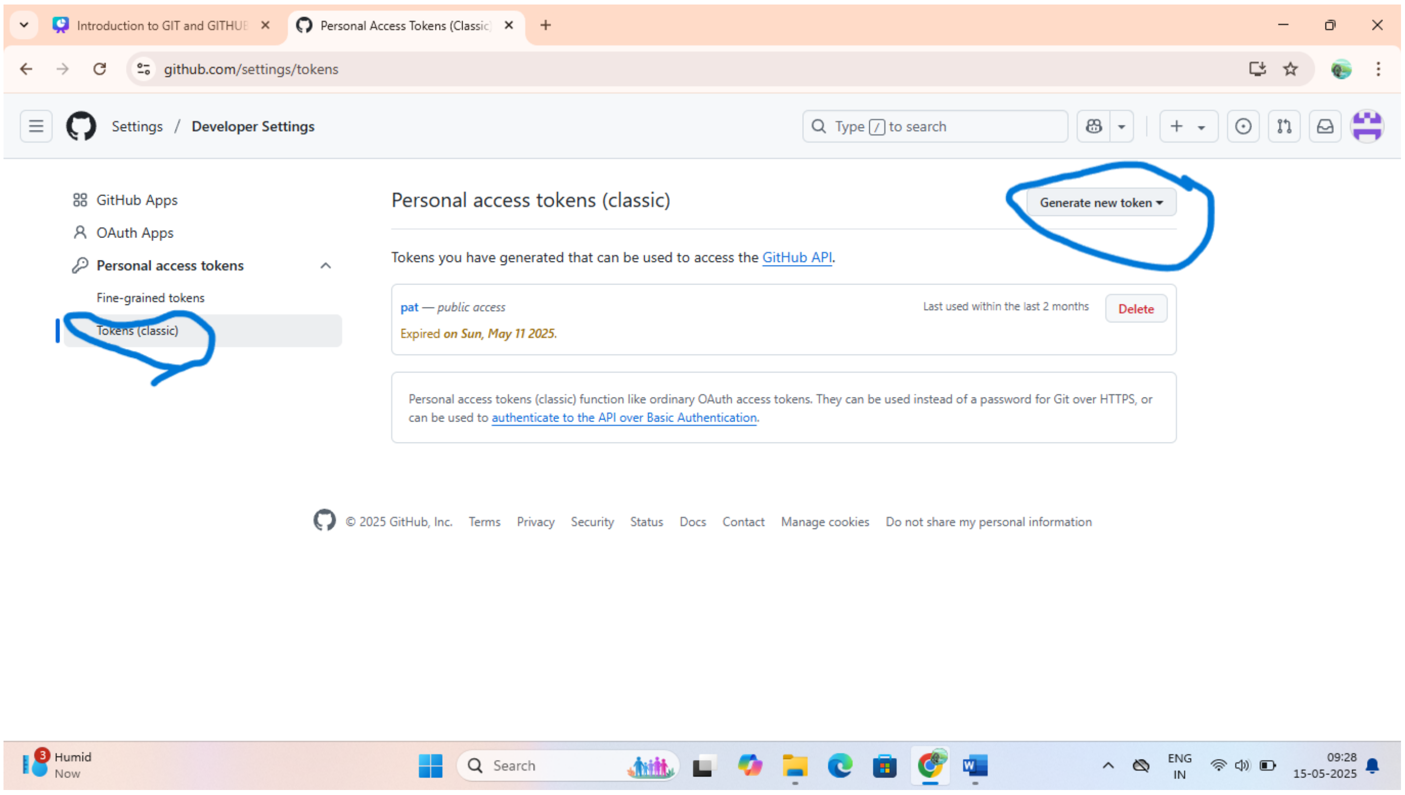
Settings / Developer Settings

GitHub Apps

No GitHub Apps

Want to build something that integrates with and extends GitHub? Register a new GitHub App to get started developing on the GitHub API.

Generate tocken and give life span for that token and save in notepad for future use


github.com/settings/tokens

Settings / Developer Settings

Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

pat — public access — Last used within the last 2 months — Delete
Expired on Sun, May 11 2025.

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Program 7

ansible-playbook playbooks/apache-install.yml

steps:

1.sudo apt update

2.sudo apt upgrade -y

3.sudo apt install ansible -y

4.ansible --version

Step 1: Create an Inventory File

1. Open your text editor to create a file called hosts.ini:

2. nano hosts.ini

3. Add the following content to define the local host:

```
[local]
localhost ansible_connection=local
```

Explanation:

▪ [local] is a group name.

▪ localhost is the target host.

▪ ansible_connection=local tells Ansible to execute commands on the local machine without SSH.

6. Save the file by pressing Ctrl+O then Enter, and exit with Ctrl+X.

Step 2: Create the Playbook File

1. Open your text editor to create a file called setup.yml:

2. nano setup.yml

```
- name: Basic Server Setup
  hosts: local
  become: true  # Optional: if your tasks need root access
  tasks:
    - name: Example task
      debug:
        msg: "Hello, this is a basic setup"
```

Step 3: Execute the Playbook

In your terminal, run the following command:

```
ansible-playbook -i hosts.ini setup.yml


ansible-playbook -i hosts.ini setup.yml --ask-become-pass
```

Program 8

Experiment 8: Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven

Project, Use Ansible to Deploy Artifacts Generated by Jenkins

Step 1:

1. first create directory for program8

.mkdir program8

2.change the directory

.cd program8

3.Preparing the Maven Project

.mvn archetype:generate -DgroupId=com.example -DartifactId=hellomaven-ci -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

.cd to hellomaven-ci(remember that what ur given in dartifact id)

4.Writing an Ansible Playbook for Deployment

A. Create an Inventory File

.in terminal type nano hosts.ini and enter below code in that (for saving ctrl+o and you have to press enter after that ctrl +x-to revert back to terminal)

```
[local]

localhost ansible_connection=local
```

B. Create the Deployment Playbook

after saving get back to terminal and type nano deploy.yml and enter below code inside it

```
- name: Deploy Maven Artifact
  hosts: localhost
  become: yes
  tasks:
    - name: Create deployment directory if it does not exist
      file:
        path: /opt/deployment
        state: directory
        mode: '0755'

    - name: Copy the artifact to the deployment directory
      ansible.builtin.copy:
        src: "/var/lib/jenkins/workspace/hellomaven-ci/target/hellomaven-ci-1.0-SNAPSHOT.jar"
        dest: "/opt/deployment/HelloMaven.jar"
```

c. Save and Exit the File

5.go to git hub create one repository by the name

.hello-maven

after that follows steps in terminal which is shown in git repository

```
  .git init
  .git add .
  .git commit -m "Initial commit: hellomaven-ci"
```

6.Then push it to your repository (GitHub, GitLab, etc.).

```
  .git push -u origin main
```

Step 2: Configuring Jenkins to Build the Maven Project

A. Create a New Jenkins Job (Freestyle Project)

1. Log into Jenkins:

Open your browser and navigate to your Jenkins URL (e.g.,

http://localhost:8080).

2. Create a New Job:

. Click “New Item” on the Jenkins dashboard.

. Enter an Item Name: e.g., HelloMaven-CI.

. Select “Freestyle project” and click “OK”.

. Configure Source Code Management (SCM)

3. Scroll to the “Source Code Management” Section:

. Select “Git”.

. Repository URL: Enter your repository URL (e.g.,

https://github.com/yourusername/HelloMaven.git).

. Branch Specifier: (e.g., */main or */master).

4. Add a Maven Build Step

1. Scroll Down to the “Build” Section:

. Click “Add build step” and select “Invoke top-level Maven targets”.

2. Configure the Maven Build:

. Goals: Type:

.clean package

This command cleans any previous builds, compiles the code, runs tests, and

packages the application into a JAR file

5. Step 3: Archiving the Artifact

After the Maven build completes, you need to archive the generated artifact so that it can be

used later by the deployment process.

1. Scroll Down to the “Post-build Actions” Section:

. Click “Add post-build action” and select “Archive the artifacts”.

2. Configure Artifact Archiving:

. Files to Archive: Type:

. target/*.jar

This pattern tells Jenkins to archive any JAR file found in the target directory

6. Step 4: Integrating Ansible Deployment in Jenkins

Now, integrate an Ansible deployment step into the Jenkins job. You can do this as a post-build action that executes a shell command.

1. Scroll up to the “Build” Section:

. Click “Add build step”and select “Execute shell”

2. Configure the Shell Command:

.In the command box, add a command to trigger your Ansible playbook.

```
  ansible-playbook -i hosts.ini deploy.yml --ask-become-pass
```

3. Save the Jenkins Job:

. Click “Save” at the bottom of the configuration page.

after that build now

Program 9

Experiment 9: Introduction to Azure DevOps: Overview of Azure DevOps
Services, Setting Up an Azure DevOps Account and Project

Overview of Azure DevOps

Azure DevOps is a comprehensive suite of cloud-based services designed to support the entire software development lifecycle. It provides tools for planning, developing, testing, delivering, and monitoring applications. Here are the primary services offered:

• Azure Repos:

A set of version control tools that allow you to host Git repositories or use Team Foundation Version Control (TFVC). It offers collaboration features such as pull requests, branch policies, and code reviews.

• Azure Pipelines:

A CI/CD service that helps automate builds, tests, and deployments. It supports multiple languages, platforms, and can run on Linux, Windows, or macOS agents.

• Azure Boards:

A work tracking system that helps teams manage work items, sprints, backlogs, and Kanban boards. It facilitates agile planning and reporting.

• Azure Test Plans:

Provides a solution for managing and executing tests, capturing data about defects, and tracking quality.

• Azure Artifacts:

Allows you to create, host, and share packages (such as Maven, npm, NuGet, and Python packages) with your team, integrating package management into your CI/CD pipelines.

These services integrate with each other and with popular third-party tools to create a cohesive DevOps ecosystem.

Step 1: Sign Up for an Azure DevOps Account

1. Open Your Web Browser:

.Navigate to the Azure DevOps website: https://dev.azure.com

2. Sign In or Create a Microsoft Account:

.If you already have a Microsoft account (such as Outlook, Hotmail, or Office 365), click "Sign in".

.If you do not have a Microsoft account, click "Create one!" and follow the instructions to create a new Microsoft account.

Step 2: Create an Azure DevOps Organization

1. click on azure devops organization or search it on search bar

.click on my azure devops organizations

.Click on create a New Organization and click on continue

. Enter a unique name for your organization (e.g., YourCompanyoryournameDevOps or MyPersonalOrg).

. Select a Region: india. (it is automatically shows india)

.Enter the characters you see

.Click "Continue" or "Create

step 3. Creating an Azure DevOps Project

1.Configure Your Project:

.Project Name: Enter a descriptive name for your project (e.g., HelloDevOps).

.Description: Optionally, provide a brief description (e.g., "A sample project to demonstrate Azure DevOps services”).

.Visibility:

▪ Choose “Private” if you want to restrict access to your project.

▪ Choose “Public” if you are okay with the project being accessible to anyone.

.Advanced Options (Optional): You can choose a version control system (Git is the default) and a work item process (Agile, Scrum, or Basic). For most beginners, the defaults are recommended.

.Click “Create

Step 4: Explore Your Project Dashboard

1. Project Overview:

.Once your project is created, you will be directed to the project dashboard. Here you will see navigation options for:

▪ Repos: Where your code is stored.

▪ Pipelines: For build and release automation.

▪ Boards: For work tracking and agile planning.

▪ Test Plans: For managing and running tests.

▪ Artifacts: For hosting packages.

2. Familiarize Yourself with the Interface:

.Click through each section (e.g., Repos, Pipelines, Boards) to get a sense of the available features.

Program 10

Step 1: Installing IntelliJ IDEA

1. Download IntelliJ Idea from https://www.jetbrains.com/idea/download/?section=linux and activate a free trial for 30 days or through the following command.

```
.sudo snap install intellij-idea-community --classic
```

2. Open the IDE using the below command in the terminal.

```
.intellij-idea-community
```

to check intalled or not

.snap info intellij-idea-community

3. Java is a pre-requisite for using this IDE

Step 2: Create a new Maven project using the IDE.

Create a project by clicking on the New Project option, choose an appropriate name and

select the following features as mentioned in the image (Build system: Maven

andLanguage: Java and check in create git repository and check advanced settings for groupid and artifact id)

and click create.

Step 3: Install GIT and configure it --go to terminal which is in intellij idea application follow below instructions

1. Check whether git is installed using the following command. $ git -v

2. Install git using $ sudo apt install git

3.Once git has been installed execute the following commands one by one.

Configure git using the below commands

.git config --global user.name “Your_Name”

.git config --global user.email “Your_Email_ID”

.git init

.git add .

.git commit -m "First Commit"

.git branch -M main

Go to https://github.com sign in with your account and create a new private repository

with the same name as your maven project.

1. Now go back to IDE’s terminal and execute the following commands

.ssh-keygen -t rsa -b 4096 -C youremail@gmail.com

(press ENTER for all questions)

.cat ~/.ssh/id_rsa.pub (copy the printed SSH key)

2. Now go to the SSH and GPG section in the GitHub settings option.

3. Click on the New SSH key button

4. Choose a suitable name and paste the SSH key into the provided space.

5.Now get back to the IDE’s terminal and type the following command

.git remote add origin git@github.com:heln123-maker/sample.git(ur adding origin by using ur name and ur project)

.git remote set-url origin git@github.com:heln123-maker/sample.git(ur name and ur project name)

Now finally run the push commands to push the code to the remote repository from the

local repository.

.git push --set-upstream origin main (Type YES for prompted question)

.git push

The code has been successfully pushed to the remote repository.

you can check it by refreshing ur repository.

STEP 4: Create a azure project same as program9

step 5:

1.Log into Azure DevOps: Navigate to your Azure DevOps organization.

2.Access Personal Access Tokens: Go to User Settings (usually at the top right of the page) and select "Personal access tokens".

3.Create a New Token: Click "New Token" to initiate the token creation process.

4.Configure the Token:

.Name: Give your token a descriptive name (e.g., "Agent Registration Token").

.Organization: Select the organization where you want to use the token.

.Expiration: Set the token's expiration date (e.g., one year).

.Scopes: Choose "Full Access" (recommended) or customize the scopes to grant the necessary

If you don’t already have a PAT:

Go to Azure DevOps > User settings (top right) > Personal access tokens

Click New Token

Scope: Agent Pools (Read & Manage) at a minimum

Expiration: Up to you

Use this PAT during agent setup when prompted.

STEP 6: Create a pipeline for the created maven repository.

Click on Pipelines on the left pane of the Azure DevOps website.

1. Click on the Create Pipeline button.
2. Choose the GitHub YAML option.
3. Select the created repository “sample” from the available repositories.
4. Give permissions if prompted to do so and sign in to your GitHub account if needed.
5. Choose Maven Pipeline from the given options.

6.A YAML file is automatically created based on the configuration details found in the pom.xml file in the repository.

1. Press the Save and Run button.

ISSUES:-have to create agent

1 mkdir myagent && cd myagent

2 tar zxvf vsts-agent-linux-x64-4.254.0.tar.gz

3 ./config.sh

Prompt Example

Server URL	https://dev.azure.com/my-org
PAT (Personal Access Token)	Generate one in Azure DevOps
Agent pool	Default or the one you created
Agent name	Anything (e.g., my-linux-agent)
Work folder	Press Enter for default (_work)

4 ./run.sh

permissions for agent registration.

MY TOKEN:-
iv1PDPvINws7E58kQu9GVcXwd0YTvTIdWO018bk4eiimtfkatc7YJQQJ99BDACAAAAAAAAAAAAAGAZD
O4AxC

program

```
trigger:
- main  # Trigger the pipeline when there is a push to the main branch

pool:

  name: Default   # Or your custom pool name

  demands:

    - agent.name -equals MyLinuxAgent
steps:
# Step 1: Maven Build and Test Execution
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'  # Ensure your pom.xml is at the repository root
    goals: 'clean package'  # This will clean, compile, and package your application
    options: ''  # You can add any additional Maven options if needed, e.g., '-DskipTests'

# Step 2: Publish Unit Test Results
```

```
- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/target/surefire-reports/TEST-*.xml'  # Path to test result files generated by Maven
    mergeTestResults: true  # Merge test results if there are multiple result files
    testRunTitle: 'Maven Unit Test Results'  # Title for the test run
```

for removing

```
.sudo snap remove intellij-idea-community
.rm -rf ~/.IdeaIC*              # For Community Edition
.rm -rf ~/.cache/JetBrains/Idea*
.rm -rf ~/.config/JetBrains/Idea*
.rm -rf ~/.local/share/JetBrains/Idea*
.rm -rf ~/.local/share/JetBrains/Toolbox
```