**4. Introduction to Git and GitHub**

1. **Setup**
   * Install Git on local machine.
   * Configure Git (user name, email).
   * Create GitHub account and generate a personal access token.

**Key Steps to Install Git on Windows**

**Download**: Visit https://git-scm.com/download/win and download the Git installer.

**Run Installer**: Double-click the downloaded file and start the setup wizard.

**Default Settings**: Click Next through the wizard, accept license, and leave settings as default unless instructed otherwise.

**Install**: Confirm the location and components, then click Install.

**Finish**: Complete the wizard and click Finish to exit.

**Verify**: Open Git Bash or Command Prompt, type **git --version** to confirm installation

**Configuration Commands**

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

**Steps**

Open Git Bash or Command Prompt.

Enter both commands (replacing with your actual name and email).

To check, use:

git config–list

**Creating a GitHub Account**

**Open GitHub sign-up page**:

Navigate to https://github.com/join or https://github.com/ in a web browser.

**Fill in account details**:

Enter a unique username (visible publicly).

Provide a valid email address.

Create a secure password (minimum 15 characters or at least 8 characters with a number and lowercase letter).

Click "Create account" or "Continue" after filling the details

**Verify you are human:**

Complete the CAPTCHA test.

**Verify your email**:

GitHub will send a verification email.

Open the email and click the verification link or enter the code on GitHub.

**Complete setup**:

Optionally, provide some profile information.

You can skip or complete user personalization.

**Generating a GitHub Personal Access Token (PAT)**

Log in to your GitHub account.

**Open Settings**:

Click your profile photo at the top-right.

Select Settings from the dropdown menu.

**Access Developer Settings**:

In the sidebar, scroll down and click Developer settings.

**Select Personal Access Tokens**:

Click on Personal access tokens.

Then click Tokens (classic) (or the available token option).

**Generate a new token**:

Click Generate new token button.

Re-enter your GitHub password if prompted.

**Configure the token**:

Add a descriptive Note for the token (e.g., "Exam token").

Set Expiration (e.g., No expiration or a set duration).

Select the necessary scopes/permissions (e.g., repo, workflow) depending on your use case.

Click Generate token to create the token.

**Copy the token immediately**:

This token is shown only once. Save it securely.

Use this token as your password for Git authentication instead of your GitHub password.

1. **Basic Git Workflow**

**Step 1**: Initialize Local Repository

git init

This creates a new empty Git repository in the current folder.

**Step 2**: Create a File using nano

nano file1.txt

Type some content (example: Hello Git using nano editor)

Press CTRL + O → Enter (to save)

Press CTRL + X (to exit nano)

**Repeat for more files:**  
nano file2.txt

**Step 3**: Add Files to Staging Area

git add .

This adds all created files (file1.txt, file2.txt, etc.) to the staging area.

**Step 4**: Check Status

git status

Confirms that files are staged and ready for commit.

Commit files

**Command**:

git commit -m "Initial commit"

his saves your changes into the local repository with a commit message ("Initial commit").

Connect local repo to GitHub (Remote Repository)

**Command**:

git remote add origin <repo\_url>

<repo\_url> is the HTTPS/SSH link of your GitHub repository.

**Example**:

git remote add origin https://github.com/username/repository.git

Push files to GitHub

**Command**:

git push -u origin master

1. **Branching and Collaboration**

The command git checkout -b feature1 is used in Git to create a new branch named "feature1" and switch (checkout) to this newly created branch immediately.

Command Breakdown

**git**: The command-line tool for Git version control.

**checkout**: The Git command used to switch from one branch to another.

**-b**: An option that tells Git to create a new branch before switching to it.

**feature1**: The name of the new branch you want to create.

**1. Create a Branch**

**Use the command**:

git checkout -b feature1

**Explanation**:

Creates a new branch named feature1.

Switches to the newly created branch immediately.

This isolates new feature work from the main branch.

**2. Work on the Feature Branch**

Make changes and commit them:

git add .

git commit -m "Implemented feature1 changes"

This records your changes in the feature1 branch.

**3. Switch Back to Main Branch**

Before merging, switch back to main:

git checkout main

Ensures main is the active branch for merging.

**4. Merge the Feature Branch into Main**

Merge feature1 into main:

git merge feature1

This incorporates the changes from feature1 into main.

If no conflicting changes exist, the merge completes automatically.

**5. Resolve Merge Conflicts (If Any)**

When Git cannot automatically merge files due to conflicting changes, it marks the conflict in the affected files. Follow these steps:

**a. Identify Conflicts**

Git will show the conflicting files after the merge.

Use:

git status

to see conflicted files listed as "both modified".

**b. Open Conflicted Files**

Conflicts are marked with special markers:

<<<<<<< HEAD

Your changes on the main branch

=======

Changes from feature1 branch

>>>>>>> feature1

Edit the file to choose which code to keep or combine parts as needed.

**c. Stage Resolved Files**

After editing and saving, mark conflicts as resolved by staging:

git add <filename>

**d. Complete the Merge Commit**

Commit the merge to finalize:

git commit

Git may auto-populate the commit message for the merge; save and close the editor.

Summary Workflow Commands

**# Create and switch to feature branch**

git checkout -b feature1

**# Work on feature, add and commit changes**

git add .

git commit -m "Add new feature1"

**# Switch back to main branch**

git checkout main

**# Merge feature branch into main**

git merge feature1

**# If merge conflict occurs:**

# 1. Check status and edit conflicted files

git status

# 2. Edit files to resolve conflicts

# 3. Stage resolved files

git add <filename>

# 4. Complete merge commit

git commit