SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY (AUTONOMOUS)

Department of Computer Science and Engineering (IOT)

LAB MANUAL

Course Code: 20ACS50

Course Title: Source Code Management Using Git and GitHub (Skill Course)

Class: IV B.Tech I Semester – CSE & CSO

Name	:	
Roll No	:	
Branch	:	
Year & Sem	:	

LIST OF EXPERIMENTS

1. Basic Installation of GIT and GITHUB
2. Basic Commands on GIT (GIT cheat sheet)
3. Basic Commands on GITHUB (GITHUB Cheat sheet)
4. Create a "repository" (project) with a git hosting tool (like Bitbucket)
5. Copy (or clone) the repository to your local machine
6. Add a file to your local repo and "commit" (save) the changes
7. "Push" your changes to your main branch
8. Make a change to your file with a git hosting tool and commit
9. "Pull" the changes to your local machine
10. Create a "branch" (version), make a change, commit the change
11. Open a "pull request" (propose changes to the main branch)
12. "Merge" your branch to the main branch

Date: ____________

Experiment 1: Installation of Git and GitHub

Aim: To install Git on the local system and create a GitHub account to enable source code management.

Software Requirements:

- Git Installer (https://git-scm.com/)
- Web Browser (Chrome, Firefox, Edge, etc.)

Theory: Git is a distributed version control system that tracks changes in source code during software development. GitHub is a web-based platform that uses Git for version control and allows developers to collaborate remotely.

Procedure:

1. Visit https://git-scm.com/ and download the appropriate Git installer for your operating system.
2. Run the downloaded installer and complete installation with default settings.
3. Open the Command Prompt (Windows) or Terminal (Linux/Mac).
4. Verify the installation by typing:

```
git --version
```

5. Open a browser and navigate to https://github.com.
6. Click Sign Up, enter your details, and create a GitHub account.

Expected Output:

- Git version displayed in terminal (e.g., git version 2.34.1)
- GitHub account successfully created

Result: *Git installed and GitHub account created.*

Signature: ____________

Date: ______________

Experiment 2: Basic Git Commands (Git Cheat Sheet)

Aim: To familiarize with basic Git commands and understand how version control works locally.

Software Requirements:

- Git installed on local machine,Terminal/Command Prompt

Theory: Git helps manage changes in project files. It uses a snapshot model, storing the state of the entire project directory with each commit. The basic Git workflow includes staging files, committing changes, and managing versions.

Procedure:

1. Open a terminal / command prompt.
2. Configure Git with your name and email:

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

3. Create a new folder and initialize it as a Git repository:

```
mkdir git-demo
cd git-demo
git init
```

4. Create a sample file and check status:

```
echo "Hello Git" > demo.txt
git status
```

5. Add the file and commit it:

```
git add demo.txt
git commit -m "Initial commit with demo.txt" git status
```

Expected Output:

- Repository initialized
- File tracked and committed

Result: *Basic Git operations performed successfully.*

Signature: ______________

Date: ______________

Experiment 3: GitHub Basics (GitHub UI Overview)

Aim: To explore the GitHub interface and understand key features like repositories, pull requests, and issues.

Software Requirements:

- GitHub account
- Web browser

Theory: GitHub provides a graphical interface over Git. Developers can host, manage, and collaborate on code with features like issues (bug tracking), pull requests (code review), and GitHub Actions (automation).

Procedure:

1. Log in to your GitHub account.
2. Click the New Repository button.
3. Name your repository (e.g., git-practice), choose visibility (Public/Private), and click Create.
4. Explore features:
   - Code: View and edit repository files
   - Issues: Report and manage bugs
   - Pull Requests: Submit code changes for review
   - Actions: Automate workflows
   - Insights: View analytics

Expected Output:

- New repository created
- Access to GitHub features

Result: *Successfully created and navigated a GitHub repository.*

Signature: _______________

Date: ____________

Experiment 4: Creating Remote Repository on GitHub

Aim: To create a remote repository on GitHub and prepare it for code upload.

Software Requirements:

- GitHub account
- Web browser

Theory: A remote repository is hosted on a cloud platform like GitHub. Developers push code from their local machine to the remote repo to share and collaborate.

Procedure:

1. Log in to GitHub.
2. Click New Repository.
3. Fill in details:
   - Repository name: myproject
   - Description (optional)
   - Check "Initialize with README"
   - Choose license if needed
4. Click Create Repository.
5. Copy the repository URL for further use.

Expected Output:

- GitHub remote repository created

Result: *Remote repository successfully created.*

Signature: ______________

Date: ____________

Experiment 5: Cloning a Repository

Aim: To clone an existing remote GitHub repository to the local machine.

Software Requirements:

- Git installed on local machine
- Terminal/Command Prompt

Theory: Cloning allows developers to make a full local copy of a repository, preserving the entire history, branches, and commits.

Procedure:

1. Copy the repository link from GitHub (HTTPS or SSH).
2. Open terminal and run:

   ```
   git clone https://github.com/username/myproject.git
   ```

3. Navigate into the cloned project:

   ```
   cd myproject
   ```

Expected Output:

- Repository cloned into local machine
- Files visible in local directory

Result: *Repository cloned successfully.*

Signature: ______________

Date: ____________

Experiment 6: Add and Commit Changes

Aim: To stage modified or new files and commit them to the local Git repository with a message.

Software Requirements:

- Git installed on local machine
- Terminal/Command Prompt

Theory: In Git, files go through a lifecycle: untracked → staged → committed. `git add` stages the files for commit. `git commit` saves the changes to the repository with a log message.

Procedure:

1. Open your terminal and navigate to your project folder.
2. Create or modify a file:

```
touch index.html
echo "
```

Hello Git

```
" > index.html
```

3. Add the file to the staging area:

```
git add index.html
```

4. Commit the file with a meaningful message:

```
git commit -m "Added index.html with basic content"
```

5. View commit history using:

```
git log
```

Expected Output:

- File added to staging
- Commit created successfully
- Commit visible in the log

Result: *Changes added and committed locally.*

Signature: ____________

Date: ____________

Experiment 7: Push Changes

Aim: To upload local commits to a remote GitHub repository.

Software Requirements:

- Git installed on local machine
- Terminal/Command Prompt

Theory: After committing changes locally, `git push` is used to upload those changes to a remote repository like GitHub. This helps in collaboration and code backup.

Procedure:

1. Ensure the remote repository is linked (use

   ```
   git remote -v
   ```

).
2. Push your changes to GitHub:

   ```
   git push origin main
   ```

   (Use `main` or `master` depending on your default branch name)
3. Provide GitHub credentials or token if prompted.

Expected Output:

- Files pushed to remote repository
- Repository reflects latest commit

Result: *Local commits successfully pushed to GitHub.*

Signature: ______________

Date: _____________

Experiment 8: Commit Using GitHub Interface

Aim: To directly edit and commit files using GitHub's web interface.

Software Requirements:

- GitHub account
- Web browser

Theory: GitHub provides a simple interface to make changes to files directly in the browser. This is useful for quick edits and documentation updates.

Procedure:

1. Open your GitHub repository.
2. Click on the file (e.g., README.md) you want to edit.
3. Click the pencil icon (Edit button).
4. Make your changes.
5. Scroll down to the commit message section.
6. Enter a message like "Updated README with project details".
7. Click Commit changes.

Expected Output:

- File updated and committed in repository
- Commit appears in the history

Result: *Successfully committed file using GitHub UI.*

Signature: _______________

Date: ____________

Experiment 9: Branching in Git

Aim: To create and manage branches in a Git repository.

Software Requirements:

- Git installed on local machine
- Terminal/Command Prompt

Theory: Branching allows developers to diverge from the main line of development and continue to work independently. This is useful for developing features, fixing bugs, or experimenting without affecting the main codebase.

Procedure:

1. Open your terminal and navigate to your project folder.
2. Create a new branch:

```
git branch feature-branch
```

3. Switch to the new branch:

```
git checkout feature-branch
```

4. Make changes to files and commit them:

```
echo "New feature" > feature.txt
git add feature.txt
git commit -m "Added feature.txt"
```

5. Switch back to the main branch:

```
git checkout main
```

6. View branches:

```
git branch
```

Expected Output:

- New branch created and switched to
- Changes committed in the feature branch
- Branch list shows both branches

Result: *Successfully created and managed branches in Git.*

Signature: ______________

Date: ____________

Experiment 10: Merging Branches

Aim: To merge changes from one branch into another.

Software Requirements:

- Git installed on local machine
- Terminal/Command Prompt

Theory: Merging is the process of integrating changes from one branch into another. This is typically done to incorporate new features or fixes into the main branch.

Procedure:

1. Ensure you are on the main branch:

```
git checkout main
```

2. Merge the feature branch into the main branch:

```
git merge feature-branch
```

3. Resolve any merge conflicts if they arise.
4. Commit the merge if necessary:

```
git commit -m "Merged feature-branch into main"
```

5. Delete the feature branch if no longer needed:

```
git branch -d feature-branch
```

Expected Output:

- Feature branch merged into main
- Changes reflected in the main branch
- Feature branch deleted (if applicable)

Result: *Successfully merged branches in Git.*

Signature: ______________

Date: ____________

Experiment 11: Pull Requests on GitHub

Aim: To create a pull request on GitHub to propose changes from one branch to another.

Software Requirements:

- GitHub account
- Web browser

Theory: A pull request is a way to propose changes to a repository. It allows team members to review the changes before merging them into the main branch.

Procedure:

1. Push your feature branch to GitHub:

```
git push origin feature-branch
```

2. Go to your GitHub repository in a web browser.
3. Click on the "Pull Requests" tab.
4. Click "New Pull Request".
5. Select the base branch (main) and compare branch (feature-branch).
6. Click "Create Pull Request".
7. Enter a title and description for your pull request.
8. Click "Create Pull Request".

Expected Output:

- Pull request created successfully
- Team members can review the changes

Result: *Successfully created a pull request on GitHub.*

Signature: ______________

Date: ______________

Experiment 12: Merge Pull Requests

Aim: To merge a pull request into the main branch after review.

Software Requirements:

- GitHub account
- Web browser

Theory: Merging a pull request integrates the proposed changes into the main branch. This is typically done after code review and approval.

Procedure:

1. Open the pull request on GitHub.
2. Review the changes and comments.
3. Click "Merge Pull Request".
4. Confirm the merge.
5. Optionally delete the feature branch after merging.

Expected Output:

- Pull request merged into the main branch
- Changes reflected in the main branch
- Feature branch deleted (if applicable)

Result: *Successfully merged a pull request on GitHub.*

Signature: ______________