
CO1 – Git Basics

1) Study of Basic Git commands

Q1. What is Git?

- Git is a version control system.
- It helps track changes in code files.
- It supports teamwork on software development.

Q2. List any 5 basic Git commands and their use.

- git init – Starts a new Git repository.
- git add – Adds files to the staging area.
- git commit – Saves changes with a message.
- git status – Shows current status of files.
- git log – Shows commit history.

Q3. What is the difference between git init and git clone?

- git init: Starts a new repository locally.
- git clone: Copies an existing repository from GitHub.

Q4. What does git status show?

- Shows tracked/untracked files.
- Displays staged and unstaged changes.

Q5. How does git commit work?

- Saves changes from staging area to the repository.
- Needs a commit message.

Q6. What is the use of git log?

- Shows all previous commits.
- Displays author name, date, and message.

CO2 – GitHub, GitLab & Collaboration

2a) Create and fork repositories in GitHub

Q1. What is a repository in GitHub?

- A storage space for your project (code, files, etc.).
- Can be public or private.

Q2. What is the difference between clone and fork?

- **Fork**: Copies a repository under your GitHub account.
- **Clone**: Downloads the repo to your computer.

Q3. How do you create a repository in GitHub?

- Click **New repository** → Add name → Choose public/private → Click **Create**.

Q4. What is the purpose of forking a repository?

- To make your own changes without affecting the original project.

2b) Apply branch, merge, rebase concepts

Q1. What is branching in Git?

- Creating a separate line of development.
- Helps test features safely.

Q2. How does git merge work?

- Combines changes from one branch into another.

Q3. What is the difference between merge and rebase?

- **Merge**: Adds a new commit with changes from another branch.
- **Rebase**: Moves or replays commits onto a new base.

Q4. What problems can occur while merging?

- Merge conflicts (when same file line is edited in both branches).

---

3) Implementation of Collaboration using Git

Q1. How do developers collaborate using Git?

- By using branches, push, pull, and merge requests.
- Everyone works on their copy and syncs changes.

Q2. What is a pull request?

- A request to merge your changes into the main branch (in GitHub).

Q3. What is the difference between push and pull?

- git push: Sends your changes to GitHub.
- git pull: Brings others' changes to your computer.

Q4. How do you resolve conflicts in Git?

- Open the file, check conflict markers, edit manually, and commit.

---

4) Collaborating and Cloning using GitHub

Q1. How do you clone a repository from GitHub?

- Use git clone <URL> command.

Q2. What happens when you clone a repository?

- Copies all files, branches, and history.

Q3. What are the steps for collaborating on GitHub?

- Clone → Create branch → Make changes → Push → Pull request.

Q4. What is a collaborator?

- A person given access to contribute to a repository.

---

5) Web IDE using GitLab

Q1. What is GitLab Web IDE?

- An online code editor in GitLab.
- Allows editing files in browser.

Q2. What are the benefits of using GitLab Web IDE?

- No need to install software.
- Edit, commit, and push from browser.

Q3. How do you open a project in Web IDE?

- Click **Web IDE** button in the GitLab project.

Q4. Can you commit changes from Web IDE?

- Yes, you can edit and commit directly.

---

6) Merge request using GitLab

Q1. What is a Merge Request in GitLab?

- A request to merge your branch into another.

Q2. How is it different from Pull Request in GitHub?

- Function is same, only name is different.

Q3. Steps to create Merge Request:

- Push branch → Click **Merge Request** → Select branches → Add title → Submit.

Q4. What is the role of a reviewer?

- Checks code before merging.

---

7) Workflow Management in GitLab

Q1. What is workflow in GitLab?

- Steps that automate development (like test, build, deploy).

Q2. What are GitLab pipelines?

- Pipelines run stages (build, test, deploy) automatically.

Q3. What is .gitlab-ci.yml?

- A config file that defines pipeline steps.

Q4. What are stages in GitLab workflow?

- Stages include build, test, deploy.

CO3 – Jenkins & Docker

8) Continuous Integration & Development using Jenkins

Q1. What is Jenkins?

- A tool for Continuous Integration/Development.
- Automates build and testing.

Q2. What is Continuous Integration?

- Merging code into main branch frequently and testing automatically.

Q3. How do you create a Jenkins job?

- Open Jenkins → New Item → Freestyle project → Add steps → Save.

Q4. What are build triggers?

- Conditions to start a build (like on code push or time).

---

9) Docker commands for content management

Q1. What is Docker?

- A tool to run apps in isolated containers.
- Containers are lightweight virtual machines.

Q2. What is the difference between Docker image and container?

- Image: Blueprint of application.
- Container: Running instance of image.

Q3. List common Docker commands:

- docker build – Builds image.
- docker run – Runs container.
- docker ps – Lists running containers.

---

10) Simple containerized app using Docker

Q1. What is a Dockerfile?

- A script with instructions to build an image.

Q2. How do you build an image from Dockerfile?

- docker build -t appname .

Q3. How do you run the app in a container?

- docker run -p 8080:80 appname

Q4. What is the use of EXPOSE?

- Tells which port the app will use.

CO4 – Ansible

11) Ad-hoc Ansible commands

Q1. What is Ansible?

- A tool for automation (like configuration, deployment).

Q2. What are ad-hoc commands?

- One-time commands to perform quick tasks.

Q3. Examples of ad-hoc commands:

- ansible all -m ping – Checks connectivity.
- ansible all -m shell -a "df -h" – Checks disk usage.

Q4. What is an inventory file?

- A file that lists the managed nodes/hosts.

---

12) Ansible Playbooks

Q1. What is a playbook in Ansible?

- A YAML file with tasks to run automatically.

Q2. Difference between ad-hoc and playbook?

- Ad-hoc: One-time task.
- Playbook: Set of tasks for automation.

Q3. What is YAML in Ansible?

- A file format used to write playbooks.
- Easy to read and write.

Q4. What are tasks and handlers?

- **Tasks**: Main operations (like install package).
- **Handlers**: Triggered only when notified (like restart service).

Dear Students,

You all can refer the following basic Git commands:

1. git init – Initializes a new Git repository in the current directory.
2. git clone <repository_url> – Clones an existing repository from a remote source (e.g., GitHub).
3. git status – Displays the current state of the working directory and the staging area.
4. git add <file> – Stages a specific file for commit.
5. git add . – Stages all modified and new files for commit.
6. git commit -m "message" – Saves the staged changes with a descriptive message.
7. git commit -am "message" – Commits tracked files with changes, skipping git add.
8. git log – Shows commit history.
9. git log --oneline – Displays commit history in a concise format.
10. git diff – Shows changes between the working directory and the last commit.
11. git diff --staged – Shows changes that have been staged for commit.

Branching and Merging

12. git branch – Lists all branches in the repository.
13. git branch <branch_name> – Creates a new branch.

14. git checkout <branch_name> – Switches to the specified branch.
15. git switch <branch_name> – Alternative to git checkout for switching branches.
16. git checkout -b <branch_name> – Creates and switches to a new branch.
17. git merge <branch_name> – Merges a branch into the current branch.
18. git branch -d <branch_name> – Deletes a local branch.
19. git branch -D <branch_name> – Forcibly deletes a branch.

Remote Repository Commands

20. git remote add origin <repository_url> – Adds a remote repository.
21. git remote -v – Lists the remote repositories linked to the project.
22. git push -u origin <branch_name> – Pushes a local branch to the remote repository.
23. git push – Pushes changes to the remote repository.
24. git pull – Fetches and integrates changes from the remote repository.
25. git fetch – Retrieves updates from a remote repository without merging.
26. git remote remove <name> – Removes a remote repository link.

Stashing and Cleaning

27. git stash – Saves uncommitted changes for later use.
28. git stash pop – Applies stashed changes back to the working directory.
29. git stash list – Shows all stashed changes.
30. git stash drop – Deletes a specific stash.
31. git clean -f – Removes untracked files from the working directory.

Undoing Changes

32. git reset <file> – Unstages a file but keeps changes.
33. git reset --hard – Resets everything (staging + working directory) to the last commit.

34. git reset --soft HEAD~1 – Moves the last commit back to the staging area.
35. git revert <commit_hash> – Creates a new commit that undoes a previous commit.
36. git checkout -- <file> – Discards changes to a specific file.

Tagging

37. git tag – Lists all tags in the repository.
38. git tag -a <tag_name> -m "message" – Creates an annotated tag.
39. git push origin <tag_name> – Pushes a specific tag to the remote repository.
40. git push --tags – Pushes all local tags to the remote repository.

Configuration & Help

41. git config --global user.name "Your Name" – Sets the global username for Git.
42. git config --global user.email "you@example.com" – Sets the global email for Git.
43. git config --list – Shows current Git configuration.
44. git help <command> – Shows help information for a specific Git command.