|  |
| --- |
| Subject Name: **Source Code Management**  Subject Code: **24CSE0106**  Cluster: **Alpha**  Department: **DCSE**    **Submitted By: Submitted To:**  **Kiratbir Singh Dr. Chetna Sharma**  **2410990383**  **G5**  Department of Computer Science & Engineering  Chitkara University Institute of   Engineering and Technology, Rajpura,  Patiala, Punjab |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Index Task 1.1#**   |  |  |  | | --- | --- | --- | | **S. No** | **Program Title** | **Page No.** | | **1.** | **To install and configure Git Client on your local system** | 01 | | **2.** | **Setting up GitHub Account and Adding Collaborators on GitHub Repository** | 03 | | **3.** | **To merge two branches within a Git repository.** | 06 | | **4.** | **To demonstrate push and pull operations in Git.** | 09 | | **5.** | **To demonstrate the concepts of forking, cloning, and creating pull requests on GitHub.** | 11 | | **6.** | **Using .gitignore and README Files** | 17 | | **7.** | **Scenario-Based Merge Conflict Resolution** | 21 | |

|  |
| --- |
| **Practical No. : 1**  **Setting Up Git Client** |
| **Aim:** To install and configure Git Client on your local system.  **Theory:** Git is a distributed version control system used to track changes in source code. Created by Linus Torvalds in 2005, it allows developers to work offline, create branches for new features, and merge them into the main codebase, facilitating collaborative work. Each developer has a local repository with a complete project history.  Git also supports collaboration through remote repositories, enabling multiple developers to work on the same project simultaneously. Its powerful features make Git an indispensable tool for modern software development, ensuring efficient and reliable version control.  **Procedure:**   1. Download Git from git-scm.com. 2. Install Git by following the setup wizard. 3. Open Git Bash and verify installation using the command: git --version. 4. Configure user details using the commands:. 5. git config --global user.name "Your Name". 6. git config --global user.email "Your Email".     **Screenshots:**     * This image guides you through the steps to install Git Bash, an essential tool for efficient version control and seamless collaboration. * Follow these instructions to get Git Bash up and running, making sure you're ready to streamline your development workflow. * With Git Bash installed, you're all set to manage your projects effectively and collaborate with others.     **Fig: Git installation wizard**   * The GNU General Public License (GPL) for Git Bash ensures that the software is free to use, modify, and distribute, promoting software freedom and collaboration among developers. * It requires that any modified versions of the software also be distributed under the same GPL license, ensuring the source code remains available to everyone.     **Fig: git --version in Git Bash**   * Set your Git username with the command git config --global user.name "Your Name". * Configure your Git email address using git config --global user.email "youremail@example.com". * To check the installed Git version, use the command git –version. |

|  |
| --- |
| **Practical No. : 2**  **Setting up GitHub Account and Adding Collaborators on GitHub Repository**  **Aim:** To add collaborators to a GitHub repository for collaborative work.  **Theory:** Collaborators are individuals with write access to a repository. They can contribute to the project by pushing changes and merging pull requests. They are essential for teamwork and project development, enabling multiple people to work on the same codebase simultaneously. Effective collaboration ensures that projects progress smoothly and efficiently.  **Procedure:**   1. Log in to your GitHub account and create a new repository. 2. Navigate to Settings > Manage Access in the repository. 3. Add collaborators by their GitHub usernames. 4. Collaborators will receive an invitation email, which they must accept.   **Screenshots:**     * Navigate to the main page of your repository on GitHub and click on the "Settings" tab.      * In the "Access" section of the sidebar, click "Collaborators" and then "Add people."      * Before that add password of your GitHub account.     **Fig: Collaborator Invitation**   * Enter the username or email of the person you want to invite, and click "Add NAME to REPOSITORY." * The user will receive an email invitation to collaborate on your repository. Once they accept, they will have collaborator access. |
| **Practical No. : 3**  **Merging Two Branches**  **Aim:** To merge two branches within a Git repository.  **Theory:** Merging branches in Git allows you to combine changes from one branch into another. It is a fundamental process in collaborative workflows, ensuring all contributions are integrated into a single codebase.  This practice helps prevent conflicts and ensures that all team members are working with the most up-to-date code. Regular merging can also make it easier to track changes and maintain a clean project history.  **Procedure:**   * Create a new branch and switch to it: **git checkout -b new-branch** * Make changes to a file in the new branch and commit them: echo "New content" > file.txt **git add file.txt git commit -m "Add changes in new branch"** * Switch back to the main branch: **git checkout main** * Modify another file in the main branch and commit the changes: **echo "Main branch changes" > another-file.txt git add another-file.txt** git commit -m "Modify file in main branch" * Merge the new branch into the main branch: **git merge new-branch**   **Screenshots:**     * Initializes a new Git repository using “**git init”**. * Create an empty file named “file.txt” using“**touch file.txt**”. * Rename the master branch by “**git branch -M main**”. * Stages all changes for the next commit using “**git add .**”.      * Shows the working directory status. * Commits the staged changes with the message "First commit". * Displays the commit history. * Creates a new branch named new-branch.      * Lists all branches in the repository using “**git branch**”. * Switches to the “**new-branch**” branch. * Creates a new empty file named file1.txt. * Stages all changes in the working directory. * Commits the staged changes with the message "Second commit".     **Fig: Merging of Branches**   * Switches to the main branch. * Merges the new-branch into the main branch using “**git merge new-branch**”.   **Practical No. : 4**  **Push/Pull Using Git**  **Aim:** To demonstrate push and pull operations in Git.  **Theory:** Push transfers committed changes from the local repository to the remote repository, while pull retrieves updates from the remote repository.  This process ensures that the local and remote repositories are synchronized, allowing team members to access the latest code changes. Regularly using push and pull commands helps maintain consistency and collaboration within a project.  **Procedure:**   * Make changes in the local repository and commit them. * Push the changes to the remote repository using git push. * Make changes directly on the remote repository (e.g., via GitHub interface). * Pull the changes to the local repository using git pull.   **Screenshots:**     * After initializing the git, commit all the changes to the local repository by using “**git commit -m “First Commit**”.      * Adds a remote repository with the URL https://github.com/mr-singh96-ux/Story.git. * Pulls the latest changes from the remote repository's main branch and rebases your local changes on top of them (if the repository is already created). * Then, pushes your local commits to the remote repository's main branch.     **Fig: Push/Pull using git** |

|  |
| --- |
| **Practical No.: 5**  **Fork, Clone, and Create Pull Requests**  **Aim:** To demonstrate the concepts of forking, cloning, and creating pull requests on GitHub.  **Theory:** Forking a repository on GitHub creates a personal copy of someone else's project in your own GitHub account. This allows you to freely make changes, experiment, and contribute to the project without affecting the original repository. It's especially useful for contributing to open-source projects, as you can work on your version independently. After making changes, you can propose your improvements to the original project through a pull request, which allows the maintainers to review and merge your changes if they approve.  Cloning a repository, on the other hand, means downloading the project to your local machine. This lets you work on the project offline in your own development environment. Cloning doesn’t create a separate copy on GitHub, it just creates a local version on your computer. If you have write access, you can push changes back to the original repository; otherwise, you can still make changes and suggest them by submitting a pull request. Pull requests allow you to propose changes to the original project, letting maintainers decide whether or not to incorporate them.  **Procedure:**   * Fork an existing repository on GitHub. * Clone the forked repository using git clone https://github.com/Harshdeep30/self.git. * Make changes in the cloned repository and push them to your forked repository. * Create a pull request to propose changes to the original repository.   **Screenshots:**   * Navigate to the repository you want to contribute to on GitHub (e.g., <https://github.com/Harshdeep30/self>). * Click the **Fork** button at the top-right corner of the GitHub repository page.     **Fig: Git repository**   * Wait for GitHub to create a copy of the repository under your own GitHub account. * Once the fork is complete, you'll be redirected to your forked repository.     **Fig: Forking the repository**   * Copy the URL of the forked repository (e.g., <https://github.com/mr-singh96-ux/self>). * Open your terminal and navigate to the directory where you want to clone the repository.     **Fig: Forked Repository**    **Fig: Add README.md file.**   * Add a Readme.md file as a change.     **Fig: Click “Pull request”**   * Now click on pull requests option on menu to create a new request.     **Fig: Creating pull request**   * Now click “New pull request” which will redirect you to pull request creation page.     **Fig: Reviewing changes**   * Review the changes made by you and hit “create pull request”.     **Fig: Created the request**   * Add a note and comment and hit pull.     **Fig: Request sent to owner**   * The pull request is now created and sent to the owner.     **Fig: Clone the repo using terminal**   * Now, use “git clone <https://github.com/Harshdeep30/self.git>” to clone the site to your local machine.     **Fig: Repo is cloned**   * The repository is cloned.   **Practical No.: 6**  **Using .gitignore and README Files**  **Aim**: To manage ignored files and create a structured README file for a repository.  **Theory**: The .gitignore file is used to tell Git which files or directories it should ignore during version control. This ensures that unnecessary or sensitive files, like temporary IDE files, build outputs, or configuration files, aren’t tracked and committed to the repository. Common entries include node\_modules/, .env, or .DS\_Store, which don’t need to be shared or versioned across different environments.  The README.md file serves as the documentation for a project, providing an overview of what the project is, how to set it up, and how to use it. It typically includes sections like installation instructions, usage examples, and contributing guidelines. This file helps users and developers quickly understand the purpose of the project and how to get started with it.  **Procedure**:   * Create a .gitignore file in your repository. * Add patterns to the .gitignore file to specify files or directories to ignore, such as: ▪ \*.log (ignore log * files). ▪ node\_modules/ (ignore the Node.js modules directory). ▪ \*.env (ignore environment files). * Create a README.md file and add the following sections: * ▪ Project Title: Add the name of your project. * ▪ Description: Provide a brief overview of the project. * ▪ Installation Steps: Include detailed steps for setting up the project. * ▪ Usage: Explain how to use the project. * Commit the .gitignore and README.md files to the repository. * Push the changes to the remote repository on GitHub.   **Screenshots:**   * Created a **.gitignore** file locally to specify which files and directories Git should ignore. * Added common files like **node\_modules/**, **.env**, and **.DS\_Store** to the **.gitignore** to prevent unnecessary or sensitive files from being tracked. Here I have added my file namely file1.txt.     **Fig: Created .gitignore File**   * Initialized a Git repository locally using git init.     **Fig: Files added to staging area**   * Created a **README.md** file locally to provide an overview, installation instructions, and usage guidelines for the project. * Staged the changes using git add . to include both **.gitignore** and **README.md** files.     **Fig: Remote Repository**    **Fig: Created README.md File**   * Committed the changes locally with a meaningful commit message like git commit -m "Add .gitignore and README.md files". * Created a remote repository on a platform like **GitHub**.     **Fig: Updated Remote Repository**   * Linked the remote repository to the local project using git remote add origin [**https://github.com/mr-singh96-ux/Story.git**](https://github.com/mr-singh96-ux/Story.git). * Pushed the local changes to the remote repository using git push -u origin main. * Verified the changes on the remote repository to ensure the .gitignore and README.md files were successfully uploaded. |

|  |
| --- |
| **Practical No.:7**  **Scenario-Based Merge Conflict Resolution**  **Aim**: To simulate and resolve merge conflicts in Git.  **Theory**: Merge conflicts occur when multiple developers make conflicting changes to the same part of a codebase, and Git cannot automatically merge those changes. This often happens when two branches modify the same line of code, delete a file that another branch modifies, or when there are conflicting updates to the same block of code. Since Git is designed to automatically merge changes when possible, a conflict arises when Git cannot determine which version of the code should be kept. The presence of a conflict typically requires human intervention, where developers manually resolve the differences.  To identify and resolve merge conflicts, Git inserts special markers within the conflicting file. These markers include <<<<<<< HEAD, which indicates the start of the current branch's changes, and >>>>>>> branch-name, which denotes the changes from the branch being merged. The code between ======= separates these conflicting changes. Developers must decide which version of the code to retain, or merge both changes, and then remove the conflict markers. Once the conflict is resolved, the developer can commit the changes, allowing the merge process to complete successfully. Merge conflicts are common in collaborative environments, and tools like git mergetool can assist in resolving them efficiently.  **Procedure:**   * Create a conflict by modifying the same file in two different branches. * Merge the branches and encounter the conflict. * Use Git tools like git mergetool or manual resolution to resolve the conflict. * Commit the resolved file.   **Screenshots:**   * Create a new Git repository or navigate to an existing one using git init. * Create and checkout a new branch, e.g., git checkout -b branch1.     **Fig: Creating a file “file.txt”**   * Modify a file (e.g., file.txt) in branch1 and make changes to it. * Commit the changes in branch1 using git add file.txt and git commit -m "Changes in branch1".     **Fig: Creating branch1 after using “git init”**   * Checkout another branch, e.g., git checkout -b branch2. * Modify the same file (e.g., file.txt) in branch2, making different changes from branch1.     **Fig: Creating branch2**   * Commit the changes in branch2 using git add file.txt and git commit -m "Changes in branch2". * Switch back to branch1 using git checkout branch1.     **Fig: Add and Commit files to branch1.**   * Merge branch2 into branch1 using git merge branch2 to simulate a merge conflict. * Observe the merge conflict message in the terminal indicating that file.txt has a conflict.     **Fig: Add and Commit files to branch1.**   * Open the conflicted file (file.txt) in a text editor to see the conflict markers (e.g., <<<<<<<, =======, and >>>>>>>). * Resolve the conflict by using the “git mergetool” command and edit the file and choosing which changes to keep or combine.     **Fig: Use “git mergetool” for resolving**   * After resolving the conflict, remove the conflict markers and save the file. * Stage the resolved file using git add file.txt.     **Fig: Conflict merge arise**   * Commit the resolved conflict using git commit -m "Resolved merge conflict in file.txt". * Verify the commit history using git log to ensure the conflict resolution is recorded.     **Fig: Fix the merge**   * Optionally, run git diff to check if any unintended changes are still in the file. * Use git status to ensure the working directory is clean after the merge.     **Fig: Successful Merging**    **Fig: After Merging in “main” branch.**  **Practical No.:8**  **Automating Workflows with GitHub Actions**  **Aim:** To automate workflows using GitHub Actions.  **Theory:** GitHub Actions allow developers to create workflows for automating software development processes like testing and deployment. This practical demonstrates how to create a workflow that greets new contributors when they open their first pull request.  **Procedure:**   * Create a .github/workflows directory in your repository. * Add a YAML file (e.g., welcome.yml) to define the workflow steps. * Use the following YAML content for the workflow file. * Commit the YAML file and push it to the repository. * Monitor the workflow execution on GitHub under the "Actions" tab. * Capture the YAML file content and screenshots of the workflow execution.   **Screenshots:**   * A new YAML file desc.yaml was created locally using the touch command. * The file was opened using the start desc.yaml command.     **Fig: Created a desc.yaml file**   * Git repository initialized using **git init**. * Changes staged using **git add desc.yaml**.     **Fig: Add Content to file**   * Commit made using **git commit -m ".yaml file added"**. * Remote repository linked using **git remote add origin** [**https://github.com/mr-singh96-ux/gen5Ai-AiContentGenerator.git**](https://github.com/mr-singh96-ux/gen5Ai-AiContentGenerator.git). * Pulled latest changes from GitHub using **git pull origin main --rebase**.     **Fig: git initialized and generated pull request**   * Pushed the file to GitHub using git push origin main.     **Fig: File pushed to Github Repository**   * A screenshot confirms the **desc.yaml** file is successfully uploaded to the remote repository.     **Fig: File Added Successfully**  **Practical No.:9**  **Final Team Project Demonstration**  **Aim:** To integrate all Git concepts into a comprehensive team project.  **Theory:** This practical combines branching, merging, conflict resolution, and automation workflows in a collaborative project. It allows team members to experience real-world version control collaboration using Git and GitHub. By simulating a development environment, students learn how to manage parallel workstreams efficiently. Automated workflows help maintain code quality and reduce manual effort. This exercise also highlights the importance of pull requests, code reviews, and continuous integration in a professional software development cycle. Working in a team fosters communication and coordination, essential for successful project delivery.  **Procedure:**   * Form a team and assign roles for branching and merging. * Create a shared project repository and define project tasks. * Implement Git workflows for project management, including: * Creating and merging branches * Resolving conflicts * Automating tasks with GitHub Actions * Demonstrate the project progress and outcomes. * 5. Capture the repository link and screenshots of the demonstration.   **Screenshots:** |
|  |