|  |  |
| --- | --- |
| Subject Name: **Source Code Management**  Subject Code: **24CSE0106**  Cluster: **Alpha**  Department: **DCSE** | |
| **Submitted By:**  Aryan Singh  (2410990059) | |  | | --- | | **Submitted To:**  Dr. Srikant Mohapatra  Department of Computer Science & Engineering  Chitkara University Institute of Engineering and Technology Rajpura, Punjab | |
|  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index Task 1.1**   |  |  |  | | --- | --- | --- | | **S. No** | **Program Title** | **Page No.** | | **1.** | **To install and configure Git Client on your local system** | **1 - 3** | | **2.** | **Setting up GitHub Account and Adding Collaborators on GitHub Repository** | **4 - 7** | | **3.** | **To demonstrate push and pull operations in Git.** | **8 - 10** | |

|  |
| --- |
| **Practical No. : 1**  **Aim:** To install and configure Git Client on your local system  **Theory:**  **Git Client Overview**  Git is a distributed version control system that helps developers track and manage code changes efficiently. It enables collaboration, maintains version history, and simplifies project management for open-source and enterprise development.  **Importance of Configuration**  Proper Git setup ensures accurate commit tracking, user identification, and streamlined workflows. Key configurations include setting up user credentials, default editors, and repository preferences for consistency and better collaboration.  **Procedure:**   * + Download Git from <https://git-scm.com/downloads> by clicking   + ‘Download for Windows’ option in the computer screen graphics.   + Git installation wizard steps:   A screenshot of a computer  Description automatically generated   * + Click on Next followed by many other next(s)        * + Click on install        * + Click on Finish * Verify Git Bash Installation using the command: git—version      * + Configure User Details and User Commands      * + Now the git is installed and configured with your name and gmail |

|  |
| --- |
| **Git Lifecycle Description**   * **Git Lifecycle Stages:**   Files in a Git project go through various phases like creation, modification, refactoring, and deletion. When a project is under Git version control, these phases correspond to specific Git states that help track changes efficiently.  Below are the key stages in the Git lifecycle:   * + **Untracked (Working Directory):**   Newly created files that Git isn’t tracking yet.  Use git status to check for untracked files.   * + **Unstaged (Modified in Working Directory):**   Files that have been modified but not yet added to the staging area.  Use git add <file> to move them to the staging area.   * + **Staged (Staging Area):**   Files that are prepared for the next commit.  Use git commit -m "message" to save them in the local repository.   * + **Committed (Git Directory - Local Repository):**   Changes are recorded in the local repository along with metadata like author, timestamp, and commit message.  Use git log to view the commit history.   * + **Pushed (Remote Repository - GitHub/GitLab):**   Commits are uploaded from the local repository to a remote repository for collaboration.  Use git push origin <branch> to sync local changes with the remote repository.   * + **Pulled (Updating Local Repository from Remote):**   Fetches and merges changes from the remote repository into the local repository.  Use git pull origin <branch> to stay updated with the latest changes.  **Basic Git Commands**   * **git init** – Initializes a new Git repository in the current directory, enabling Git to track changes. * **git status** – Displays the state of the working directory and staging area, showing untracked, modified, or staged files. * **git add <file>** – Moves changes from the working directory to the staging area, preparing them for a commit. * **git commit -m "message"** – Captures a snapshot of the staged changes and saves them to the repository with a descriptive message. * **git push** – Uploads local commits to a remote repository, making them available to others. * **git pull** – Fetches the latest changes from a remote repository and merges them into the current branch to stay updated. * **git log** – Displays a history of commits, including details like commit hashes, authors, timestamps, and messages.   **Practical No. : 2**  **Aim**: Setting up GitHub Account and Adding Collaborators on GitHub  Repository  **Theory:**  GitHub is an online platform that provides a cloud-based service for version control using Git. It enables individuals and teams to store, manage, and collaborate on code efficiently. Developers can track modifications, contribute to projects, and share their work seamlessly.  **Procedure:**  **Creating a Git Hub Account:**  1. Open a web browser and search for GitHub or visit <https://github.com/signup>.  2. Enter a valid email address, choose a username, and create a secure password.  3. Follow the on-screen instructions to complete the sign-up process.     * After visiting the link this type of interface will appear, if you already have an account, you can sign in and if not, you can create.     GitHub Login        Git Hub Interface  **Adding Collaborators to a GitHub Repository:**  1.Sign in to GitHub – Ensure you are logged into your account.  2.Access Your Dashboard – Click the "+" icon in the upper-right corner and select "New repository."  3.Set Up the Repository – Provide the necessary details:   * Repository Name – Choose a unique name. * Description (Optional) – Briefly describe the project. * Visibility – Choose between a public or private repository. * Initialize with README (Optional) – Select this option if you want a README file created automatically.   4.Click "Create repository" to finalize the setup.    Creating a new repository  **Inviting Collaborators:**  5. Open the newly created repository.  6. Navigate to **Settings** in the top menu.  7. Scroll down and select **Collaborators** under the **Access** section.  8. Click **"Add collaborator."**  9. Enter the GitHub username of the person you want to invite.  10.Click **"Add"** to send the invitation.  11.Wait for the invitee to accept the request. Once accepted, they will  have collaborator access.    Go to settings in repository      Click add people to add collaborators in repository  **Managing Collaborators:**  9.In the Collaborators section, you can see a list of all collaborators.  10.To change permissions, click the dropdown next to the collaborator’s name and choose the appropriate role:   * Read – Can view the repository but cannot make changes. * Write – Can push changes but cannot manage settings.\ * Admin – Has full control over the repository.   11.To remove a collaborator, click the Remove button next to their name. This will revoke the access immediately.  12. Transferring Ownership (If Needed)  13.If you want to transfer the repository to another user:   * Go to Settings > General. * Scroll down to Danger Zone and click Transfer ownership. * Enter the new owner’s GitHub username and confirm the transfer**.**     Here you can see the list of collaborators      After clicking the manage, you can do the given changes(danger zone)  **Practical 3: Push/Pull Using Git**  **Aim:** To demonstrate push and pull operations in Git.  **Theory:**  The git push command is used to transfer local commits to a remote repository, such as GitHub, making the latest changes accessible to others. Before pushing, modifications need to be staged using git add and committed with git commit. The command git push origin branch\_name then uploads these updates to the remote repository, ensuring consistency between the local and remote versions.  On the other hand, git pull is used to retrieve and merge the latest changes from the remote repository into the local branch. By executing git pull origin branch\_name, developers can fetch new updates made by collaborators and incorporate them into their local copy. This ensures that all team members work with the most recent version of the project, reducing conflicts and promoting seamless collaboration.  Additionally, git fetch allows developers to retrieve remote changes without immediately merging them. This provides an opportunity to review updates before integrating them into the local branch within a Version Control System (VCS).  **Procedure:**   * **Push:**  1. Firstly we will initialize the git repository and check the status.        1. Now we will add any new file that we have created      1. Then we will use git commit -m to save any changes in git before pushing them to repository. We are saving all staged files in git history with a message “commit1”.          1. After that we will connect the local git repository to remote repository in git hub.      1. Use git pull origin main --rebase to update your branch by applying remote changes first, then replaying local commits for a clean, linear history.        1. Push all the files in the git hub repository.      1. Now we can see all the files in git hub repository.      * **Pull:**  1. First we will create a new file in our repository by clicking on add file.      1. Name the file, write content and click on “Commit changes”.      1. Now we can see the file in our repository.      1. Now we will use pull command to add the file to our local storage. |