Installation and Navigation of Visual Studio Code (VS Code) Instructions: Answer the following questions based on your understanding of the installation and navigation of Visual Studio Code (VS Code). Provide detailed explanations and examples where appropriate.

Questions:

1. Installation of VS Code:
    - Describe the steps to download and install Visual Studio Code on Windows 11 operating system. Include any prerequisites that might be needed.

    Open your web browser (e.g., Microsoft Edge, Google Chrome). Go to the official Visual Studio Code website: https://code.visualstudio.com/. Click on the "Download for Windows" button. This should automatically detect your operating system. The download should start automatically. If not, click on the download link provided. Once the download completes, locate the downloaded installer file (typically in your Downloads folder). Double-click on the installer file (VSCodeSetup.exe) to start the installation process. The installer will prompt you for installation options. You can typically accept the default settings unless you have specific preferences. Click "Next" through any initial screens (such as license agreement, destination folder). Optionally, you can choose additional tasks like creating a desktop icon or adding VS Code to the PATH for command-line use. Click "Install" to begin the installation process. Wait for the installer to complete the installation process. This usually takes a few moments. Once the installation finishes, you may see an option to launch Visual Studio Code immediately. Check or uncheck this option based on your preference. Click "Finish" to exit the installer.

Upon first launch, Visual Studio Code may prompt you to install recommended extensions based on the file type you opened. You can choose to install these extensions or skip the step. Verify that Visual Studio Code is running correctly by creating or opening a project, writing some code, and testing basic functionalities like syntax highlighting, code completion, and debugging (if applicable).

2. First-time Setup:
    - After installing VS Code, what initial configurations and settings should be adjusted for an optimal coding environment? Mention any important settings or extensions.
3. User Interface Customization: Theme: Choose a theme that suits your preference (e.g., Dark+, Light+, Material Theme). Go to File > Preferences > Color Theme to select a theme.

Icons: Install an icon theme for better visual differentiation of file types. Go to File > Preferences > File Icon Theme to choose an icon theme (e.g., Material Icon Theme, VS Code Icons).

2. Editor Settings: Font Size and Family: Adjust font size and family according to your readability preference. Go to File > Preferences > Settings, search for editor.fontFamily and editor.fontSize.

Line Numbers: Turn on line numbers for better code navigation. Set editor.lineNumbers to "on".

Word Wrap: Enable word wrap for a cleaner view. Set editor.wordWrap to "on".

Indentation: Set your preferred indentation (e.g., spaces or tabs). Configure editor.tabSize and editor.insertSpaces.

Format on Save: Automatically format code when you save. Set editor.formatOnSave to true.

3. Extensions: Programming Language Support: Install extensions for the programming languages you work with (e.g., Python, JavaScript, Java, etc.). These extensions provide syntax highlighting, IntelliSense, and debugging capabilities specific to each language.

Git Integration: Install Git extensions like GitLens for enhanced Git functionality within VS Code.

Code Formatting: Extensions like Prettier or ESLint for JavaScript/TypeScript, Black for Python, etc., can help maintain consistent code formatting.

Debugging: Install extensions like Debugger for Chrome, Python Extension Pack, or Java Extension Pack for debugging support in respective languages.

4. Workspace Settings (Optional): Folder Specific Settings: If you have specific settings for different projects, you can create .vscode/settings.json files in your project folders.
5. User Interface Overview:
    - Explain the main components of the VS Code user interface. Identify and describe the purpose of the Activity Bar, Side Bar, Editor Group, and Status Bar.
6. Activity Bar: Purpose: The Activity Bar is located on the far left side of the VS Code window. It provides quick access to different views and panels within VS Code.

Components:

Explorer: Allows navigation through files and folders in your workspace. Search: Provides tools for searching across files and folders. Source Control: Integrates with version control systems like Git. Run and Debug: Facilitates running and debugging applications. Extensions: Manages VS Code extensions installed on your system. 2. Side Bar: Purpose: The Side Bar is located next to the Activity Bar and contains additional navigation options and information related to your current project or workspace.

Components:

File Explorer: Displays the file structure of your project for easy navigation. Search: Allows searching within files or across the project. Source Control: Shows Git status and provides version control actions. Extensions: Lists installed extensions and allows management (similar to the Activity Bar). 3. Editor Group: Purpose: The Editor Group occupies the central area of the VS Code window and is where you write and edit code.

Components:

Editor Tabs: Each tab represents an open file. You can have multiple tabs open simultaneously within the same Editor Group. Split Editors: You can split the editor horizontally or vertically to view and work on multiple files simultaneously. 4. Status Bar: Purpose: The Status Bar is located at the bottom of the VS Code window and provides information about the current state of the editor and workspace.

Components:

Language Mode: Displays the programming language associated with the current file. Line and Column Numbers: Shows the current cursor position in the editor. Git Branch: Displays the active Git branch and status if the project is under version control. Errors and Warnings: Shows diagnostics such as errors and warnings in the current file. Notification Area: Provides notifications about tasks, extensions, and other relevant information. Additional Components: Toolbar: Above the Editor Group, the Toolbar contains buttons for common actions like saving files, undo/redo, and running/debugging.

Integrated Terminal: Accessible through View > Terminal or with the shortcut `Ctrl+`` (backtick), it allows you to execute shell commands and scripts within VS Code.

Customization: Layout: VS Code's user interface is highly customizable. You can drag and drop components to rearrange them according to your workflow preferences.

Extensions: Many additional functionalities can be added through extensions, which can modify or add new views and behaviors within VS Code. 4. Command Palette:

- What is the Command Palette in VS Code, and how can it be accessed? Provide examples of common tasks that can be performed using the Command Palette. The Command Palette in Visual Studio Code (VS Code) is a powerful feature that allows users to access various functionalities and commands through a text-based interface. It provides a quick and efficient way to execute commands, search for settings, install extensions, and perform many other tasks without navigating through menus or remembering keyboard shortcuts.

Accessing the Command Palette: To open the Command Palette in VS Code, you can use one of the following methods:

Keyboard Shortcut: Press Ctrl+Shift+P (Windows, Linux) or Cmd+Shift+P (Mac). Menu Option: Click on View in the top menu, then select Command Palette.... Once the Command Palette is open, you'll see a text input field where you can start typing to search for commands. As you type, the Command Palette filters and displays relevant commands and options.

Examples of Common Tasks Using the Command Palette: File Operations:

Open File: Type File: Open File to open a specific file by browsing the file system. New File: Type File: New File to create a new file in the current workspace. Save All: Type File: Save All to save all open files. Editing and Navigation:

Go to Line: Type Go to Line... to navigate directly to a specific line number in the current file. Toggle Line Comment: Type Toggle Line Comment to comment or uncomment the current line or selected lines. Git and Version Control:

Git: Pull: Type Git: Pull to pull the latest changes from a remote Git repository. Git: Commit: Type Git: Commit to commit changes to the local repository. Extensions and Marketplace:

Extensions: Install Extensions: Type Extensions: Install Extensions to browse and install extensions from the VS Code Marketplace. Extensions: Show Recommended Extensions: Type Extensions: Show Recommended Extensions to view a list of recommended extensions for your current workspace. Settings:

Preferences: Open Settings: Type Preferences: Open Settings to open the settings.json file where you can configure VS Code settings. Preferences: Configure Language Specific Settings...: Type Preferences: Configure Language Specific Settings... to set specific preferences for different programming languages. Debugging:

Debug: Start Debugging: Type Debug: Start Debugging to begin debugging your application based on the current configuration. Tasks and Runners:

Tasks: Run Task: Type Tasks: Run Task to execute a specific task defined in your workspace's tasks.json file. Terminal:

Terminal: Create New Integrated Terminal: Type Terminal: Create New Integrated Terminal to open a new integrated terminal within VS Code. Additional Tips: Command Palette History: VS Code remembers the commands you've recently used in the Command Palette, making it easier to quickly access frequently performed tasks.

Extensions: Many extensions add their own commands to the Command Palette, extending its functionality based on your specific needs (e.g., Python: Run Python File in Terminal). 5. Extensions in VS Code:

- Discuss the role of extensions in VS Code. How can users find, install, and manage extensions? Provide examples of essential extensions for web development. Extensions in Visual Studio Code (VS Code) are essential tools that extend its functionality beyond the core editor capabilities. They allow users to customize their development environment, add support for various programming languages, integrate with external tools, enhance productivity, and more. Here’s a detailed overview of the role of extensions and how users can find, install, and manage them:

Role of Extensions: Enhanced Functionality: Extensions add new features such as language support, debugging tools, version control integration, and task automation.

Customization: Users can personalize their coding experience by adding themes, customizing keybindings, and adjusting editor behavior through extensions.

Integration: Extensions facilitate integration with external services and APIs, enabling seamless workflow integration from within VS Code.

Finding and Installing Extensions: Finding Extensions:

Marketplace: Access the VS Code Marketplace directly within the editor (Extensions view in the Activity Bar or Command Palette). Website: Browse extensions on the VS Code Marketplace website. Installing Extensions:

Open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P) and type Extensions: Install Extensions. Search for the desired extension by name or functionality. Click Install next to the extension you want to install. Managing Extensions: Viewing Installed Extensions:

Open the Command Palette and type Extensions: Show Installed Extensions. Manage installed extensions (enable/disable, update, uninstall) directly from the Extensions view. Settings Sync:

Use VS Code's built-in Settings Sync to synchronize installed extensions across multiple machines. Examples of Essential Extensions for Web Development: Live Server:

Purpose: Launches a local development server with live reload capability for static and dynamic pages. Benefits: Instant preview of changes in the browser without manually refreshing. Installation: Search for Live Server in the Extensions Marketplace. ESLint:

Purpose: Integrates ESLint for JavaScript and TypeScript linting within VS Code. Benefits: Provides real-time feedback on code quality and style adherence. Installation: Search for ESLint in the Extensions Marketplace. Prettier - Code Formatter:

Purpose: Automatically formats code based on predefined rules for various languages including JavaScript, TypeScript, CSS, and JSON. Benefits: Ensures consistent code formatting across your projects. Installation: Search for Prettier - Code Formatter in the Extensions Marketplace. Debugger for Chrome:

Purpose: Allows debugging JavaScript and TypeScript code directly from VS Code in the Google Chrome browser. Benefits: Debug web applications efficiently without leaving the editor. Installation: Search for Debugger for Chrome in the Extensions Marketplace. CSS Peek:

Purpose: Enables CSS class and ID definitions peeking (i.e., navigating to their definitions) directly from HTML files. Benefits: Facilitates rapid CSS editing and debugging. Installation: Search for CSS Peek in the Extensions Marketplace. Auto Close Tag / Auto Rename Tag:

Purpose: Automatically closes HTML/XML tags or renames paired HTML/XML tags when one of them is modified. Benefits: Improves productivity by reducing manual tag management. Installation: Search for Auto Close Tag and Auto Rename Tag separately in the Extensions Marketplace. Conclusion: Extensions are instrumental in tailoring Visual Studio Code to meet specific development needs and preferences. By leveraging extensions, users can enhance productivity, streamline workflows, and improve the overall development experience across various languages and frameworks, including web development. The ease of finding, installing, and managing extensions ensures that developers can quickly adapt their coding environment to suit different projects and tasks efficiently. 6. Integrated Terminal:

- Describe how to open and use the integrated terminal in VS Code. What are the advantages of using the integrated terminal compared to an external terminal? Opening and using the integrated terminal in Visual Studio Code (VS Code) is straightforward and offers several advantages over using an external terminal. Here's how you can open and utilize the integrated terminal:

Opening the Integrated Terminal: Using Menu:

Click on Terminal in the top menu. Select New Terminal. Using Keyboard Shortcut:

Press `Ctrl+`` (backtick) to toggle the integrated terminal. Using Command Palette:

Open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P). Type View: Toggle Integrated Terminal and press Enter to open or close it. Using the Integrated Terminal: Once the integrated terminal is open, you can perform various tasks directly within VS Code:

Navigate to Project Directory: Use standard command-line commands (cd, dir, ls, etc.) to navigate to your project directory.

Run Commands and Scripts: Execute commands, run scripts (e.g., npm start), compile code, or manage files as you would in an external terminal.

Interact with Version Control: Utilize Git commands (git commit, git push, etc.) without switching to a separate Git client or terminal window.

Debugging: Launch and debug applications with tools like node, python, or other languages directly from the terminal.

Advantages of Using the Integrated Terminal: Contextual Integration:

The integrated terminal runs within the VS Code environment, providing seamless integration with your editor and workspace. You can directly interact with your files and projects without switching between different applications. Efficiency and Productivity:

Minimizes context switching by keeping your development tools in one place. You can quickly execute commands, run scripts, and debug without leaving your coding environment. Workspace Awareness:

The integrated terminal automatically opens at the root of your current workspace, making it easier to manage and execute commands specific to your project. Customization:

You can customize the integrated terminal's appearance, behavior, and shell (e.g., PowerShell, Command Prompt, Bash) to suit your preferences and workflow. Multi-Platform Consistency:

VS Code and its integrated terminal work consistently across different operating systems (Windows, macOS, Linux), providing a unified experience regardless of the platform you're using. Extension Integration:

Extensions like shell syntax highlighting, additional command-line tools, and terminal themes can enhance the integrated terminal's functionality based on your needs. Comparison with External Terminals: Convenience: Eliminates the need to switch between VS Code and an external terminal window, saving time and improving workflow efficiency.

Workspace Awareness: Opens directly within the context of your project, reducing the risk of executing commands in the wrong directory.

Integration: Seamlessly integrates with other VS Code features such as debugging, Git version control, and task automation.

Performance: Generally performs as well as or better than external terminals in terms of responsiveness and resource usage.

In conclusion, using the integrated terminal in VS Code offers significant advantages by enhancing workflow integration, productivity, and convenience, making it a preferred choice for many developers working within the VS Code ecosystem.

7. File and Folder Management:
    - Explain how to create, open, and manage files and folders in VS Code. How can users navigate between different files and directories efficiently? Managing files and folders in Visual Studio Code (VS Code) is essential for organizing and working on projects efficiently. Here's a comprehensive guide on how to create, open, and manage files and folders, along with tips for efficient navigation between different files and directories:

Creating and Opening Files and Folders: Creating a New File or Folder:

File: Right-click in the Explorer view (Side Bar) or use the Command Palette (Ctrl+Shift+P or Cmd+Shift+P) and type File: New File. Folder: Right-click in the Explorer view or use the Command Palette and type File: New Folder. Opening Files:

From Explorer: Double-click on a file in the Explorer view to open it. Using Command Palette: Type File: Open File in the Command Palette and select the file you want to open. Opening Multiple Files:

Hold Ctrl (Windows/Linux) or Cmd (Mac) while clicking to select multiple files in the Explorer view, then right-click and choose Open in Editor. Opening Recent Files:

Use the File menu or Ctrl+R (Cmd+R on Mac) to open a list of recently opened files. Managing Files and Folders: Renaming and Deleting:

Rename: Right-click on a file or folder in the Explorer view and choose Rename, or press F2. Delete: Right-click and choose Delete to remove a file or folder (this action moves it to the Recycle Bin or Trash). Moving and Copying:

Cut and Paste: Use Ctrl+X (Cmd+X on Mac) to cut a file or folder, navigate to the destination, and use Ctrl+V (Cmd+V on Mac) to paste. Drag and Drop: Drag files or folders within the Explorer view to move them to a different location. Search and Replace:

Use Ctrl+F (Cmd+F on Mac) for basic search and Ctrl+H (Cmd+Option+F on Mac) for search and replace within open files. Efficient Navigation Between Files and Directories: Using the Explorer View (Side Bar):

Navigate through files and folders by expanding or collapsing directory structures in the Explorer view. File Switching:

Tabs: Click on open file tabs at the top of the editor to switch between files quickly. Recent Files: Use Ctrl+Tab (Cmd+Tab on Mac) to cycle through recently opened files. Go to File:

Use Ctrl+P (Cmd+P on Mac) to open the Quick Open menu and type the name of the file you want to open. VS Code will suggest matching files based on the current workspace. Go to Symbol:

Use Ctrl+Shift+O (Cmd+Shift+O on Mac) to open the Go to Symbol menu, allowing you to navigate to specific functions, classes, or symbols within the current file. Navigation History:

Use Alt+Left (Cmd+Left on Mac) and Alt+Right (Cmd+Right on Mac) to navigate backward and forward through navigation history, allowing you to return to previously visited locations. Tips for Efficiency: Workspace Folders: Add multiple folders to your workspace (File > Add Folder to Workspace) to work on different projects simultaneously.

Custom Keybindings: Customize keybindings (File > Preferences > Keyboard Shortcuts) to assign shortcuts for frequent actions.

Extensions: Install file management extensions like File Utils, Path Intellisense, or Project Manager to enhance file and folder navigation and management capabilities.

By mastering these techniques, you can effectively create, open, organize, and navigate between files and directories in Visual Studio Code, optimizing your productivity and workflow management while working on projects of varying complexity.

8. Settings and Preferences:
    - Where can users find and customize settings in VS Code? Provide examples of how to change the theme, font size, and keybindings. In Visual Studio Code (VS Code), users can find and customize settings to tailor their coding environment according to their preferences. Here's how you can locate and customize settings, along with examples of changing the theme, font size, and keybindings:

Finding and Customizing Settings: Settings Interface:

Open the Settings UI by clicking on the gear icon (⚙) in the bottom left corner of the Activity Bar or by pressing Ctrl+, (Cmd+, on Mac). Alternatively, use the Command Palette (Ctrl+Shift+P or Cmd+Shift+P) and type Preferences: Open Settings (UI). Settings JSON File:

Open the settings.json file directly to edit settings in JSON format. Access it through File > Preferences > Settings and click on the {} icon in the top right corner to edit in JSON mode. Examples of Customization:

1. Changing the Theme: Using the Settings UI: Open the Settings UI (Ctrl+, or Cmd+,). In the search bar, type Color Theme. Choose a theme from the dropdown list (e.g., Dark+, Light+, Material Theme). Using settings.json: json Copy code "workbench.colorTheme": "Material Theme Darker"
2. Adjusting Font Size: Using the Settings UI:

Open the Settings UI (Ctrl+, or Cmd+,). In the search bar, type Font Size. Adjust the Editor: Font Size setting. Using settings.json:

json Copy code "editor.fontSize": 14 3. Customizing Keybindings: Using the Settings UI:

Open the Settings UI (Ctrl+, or Cmd+,). In the search bar, type Keybindings. Click on Open Keyboard Shortcuts (JSON) to edit keybindings in JSON format. Example of Adding a Custom Keybinding in keybindings.json:

json Copy code // Place your key bindings in this file to overwrite the defaults [ { "key": "ctrl+shift+l", "command": "editor.action.insertLineAfter", "when": "editorTextFocus" } ] Additional Tips: Search and Filter: Use the search bar in the Settings UI to quickly find specific settings or options.

Workspace Settings: VS Code allows you to customize settings globally (user settings) or per workspace. Workspace settings override user settings when specified.

Extensions: Some extensions (e.g., themes, language-specific settings) may add their own configurations to the settings UI or settings.json.

Sync Settings: If you use VS Code across multiple devices, you can synchronize your settings using Settings Sync to maintain consistency.

By exploring and customizing these settings, users can create a personalized development environment in Visual Studio Code that enhances productivity and matches their workflow preferences effectively. 9. Debugging in VS Code:

- Outline the steps to set up and start debugging a simple program in VS Code. What are some key debugging features available in VS Code? Setting up and starting debugging in Visual Studio Code (VS Code) involves a few straightforward steps. Below is a guide to help you get started with debugging a simple program, along with an overview of key debugging features available in VS Code:

Steps to Set Up and Start Debugging: Open Your Project in VS Code:

Launch VS Code and open the folder containing your project or create a new file if starting from scratch. Create or Open the Program File:

Create a new file or open an existing file where your code resides (e.g., app.js for JavaScript, main.py for Python). Add Debug Configuration:

Click on the Debugging icon in the Activity Bar on the side (or press Ctrl+Shift+D). Click on the gear icon (⚙ Configure or Add Configuration). Select the environment for your program (e.g., Node.js for JavaScript, Python for Python). VS Code will generate a default launch.json file in the .vscode folder with the basic configuration. Set Breakpoints:

Navigate to the line(s) in your code where you want to set breakpoints (places where the debugger will pause execution). Click in the gutter next to the line number or press F9 to set/unset a breakpoint. Start Debugging:

Press F5 or click Run > Start Debugging from the top menu. Alternatively, use the green play button in the Debug sidebar. Debugging Controls:

Once the debugger is running, you can use controls in the Debug toolbar: Step Over: Execute the current line and move to the next line in the current function. Step Into: Jump into the function call at the current cursor position. Step Out: Finish executing the current function and return to the caller. Continue: Resume execution until the next breakpoint or program completion. Restart: Restart the debugging session. Stop: Terminate the debugging session. Key Debugging Features in VS Code: Variable Inspection:

View the current value of variables in the Debug Sidebar or hover over them in the code editor. Watch Expressions:

Monitor specific variables or expressions by adding them to the Watch view. Call Stack:

See the function call hierarchy leading to the current breakpoint or execution point. Conditional Breakpoints:

Set breakpoints that only trigger under certain conditions, enhancing debugging flexibility. Debug Console:

Interact with your program during debugging sessions using the Debug Console to execute commands and evaluate expressions. Multi-threaded Debugging (for supported languages):

Manage and debug multiple threads simultaneously, viewing their status and switching between them as needed. Integrated Terminal:

Access the integrated terminal (`Ctrl+``) for additional debugging commands or interactions with the debugging environment. Exception Handling:

Configure how exceptions are handled during debugging sessions, enabling you to catch and examine errors. Tips for Effective Debugging: Use Logpoints: Insert log messages directly into the code without modifying it permanently. Debugging Extensions: Install language-specific debugging extensions for enhanced debugging capabilities (e.g., Python, Java). Remote Debugging: Debug applications running on remote machines or containers using VS Code’s Remote Development extensions. By leveraging these debugging features and following these

steps, developers can effectively identify and resolve issues in their codebase, ensuring smoother and more efficient development workflows in Visual Studio Code. 10. Using Source Control: - How can users integrate Git with VS Code for version control? Describe the process of initializing a repository, making commits, and pushing changes to GitHub. Integrating Git with Visual Studio Code (VS Code) for version control allows developers to manage their code repositories directly within the editor. Here's a step-by-step guide on how to initialize a repository, make commits, and push changes to GitHub using VS Code:

Prerequisites: Install Git:

Ensure Git is installed on your machine. You can download it from git-scm.com and follow the installation instructions. Set Up GitHub Account:

Create a GitHub account if you don't have one. You'll need this to push your local repository changes to GitHub. Initializing a Repository: Open Your Project in VS Code:

Launch VS Code and open the folder containing your project. Initialize Git Repository:

Open the integrated terminal in VS Code (`Ctrl+``) and navigate to your project directory if it's not already open. Initialize a new Git repository by running the following command: bash Copy code git init This command initializes a new Git repository in your project folder. Stage Files for Commit:

In VS Code, open the Source Control view by clicking on the Git icon in the Activity Bar on the side (Ctrl+Shift+G). You'll see a list of changes (untracked files). Click on the + button next to each file or use the + button at the top of the Source Control view to stage all changes. Making Commits: Commit Changes:

After staging your changes, enter a commit message in the text box at the top of the Source Control view. Click the checkmark icon (✓) or press Ctrl+Enter to commit the changes. View Commit History:

You can view the commit history by clicking on the clock icon in the Source Control view to see a list of recent commits. Pushing Changes to GitHub: Create a GitHub Repository:

Go to GitHub and create a new repository. Note down the repository URL (e.g., https://github.com/username/repository-name.git). Set Remote Repository:

In the VS Code integrated terminal, add your GitHub repository as the remote origin: bash Copy code git remote add origin Replace with your GitHub repository URL. Push Changes to GitHub:

Push your committed changes from your local repository to GitHub: bash Copy code git push -u origin master This command pushes the changes in your local master branch to the origin remote repository on GitHub. Authenticate with GitHub:

If prompted, authenticate with your GitHub credentials to complete the push operation. Additional Git Operations in VS Code: Pull Changes: Use the git pull command to fetch and merge changes from the remote repository to your local repository. Branch Management: Create,

switch, and merge branches using VS Code's Source Control view or the integrated terminal. Discard Changes: Discard local changes using the Source Control view's Discard Changes option or git checkout -- command in the terminal. Tips for Effective Git Usage: Commit Frequently: Make small, meaningful commits with descriptive messages to track changes effectively. Use Branches: Utilize branches for feature development, bug fixes, or experimental changes to keep the master branch stable. Review Diffs: Review file differences before staging and committing changes using the Source Control view. By following these steps and utilizing VS Code's integrated Git features, developers can efficiently manage version control for their projects and collaborate effectively using GitHub or other Git repositories.

References https://chatgpt.com/