- **Git:** Git is a distributed version control system used in software development to help developers track changes in their code and collaborate with others efficiently.

  Git is used for a variety of reasons, such as:

  - **Version Control:** Git allows developers to keep track of changes made to their codebase over time. Every change is recorded as a commit. This history of changes is invaluable for debugging, auditing, and understanding the evolution of the project, and it makes it easy to revert to previous versions of our code if something goes wrong.

  - **Collaboration:** Git enables multiple developers to work on the same codebase simultaneously. Each developer can have their own copy of the repository, make changes independently, and we can share our code, review each other's work, and merge changes back into the main codebase.

  - **Open Source and Community:** Git is open-source software and has a large and active community of users and contributors. This makes it easy to find support, resources, and tools for working with Git.

  - **Efficiency:** Git stores data in a very efficient way, which makes it easy to branch, merge, and revert changes.

  - **Reliability:** Git is very reliable, and it has a number of built-in features that help to prevent data loss.

  - **Git offers backup and recovery** by storing copies of code remotely. If a local copy is lost, developers can retrieve it from the remote repository, preventing data loss.

- **GitHub:** GitHub is a web-based platform for hosting and managing Git repositories. It makes it easy to collaborate on and manage Git projects, and it offers features like code sharing, issue tracking, and project management to aid teams in their collaborative efforts.

- **Some important Linux commands:**

  1. ls: Lists files and folders in the current directory.
  2. cd: Changes to a different directory.
  3. pwd: Prints the current directory you are in.
  4. mkdir: Creates a new folder.
  5. rmdir: Removes an empty folder.
  6. touch: Creates a new empty file or updates the timestamp of an existing file.
  7. rm: Removes or delete a file or folder permanently.
  8. cp: Copies a file or folder to a new location.
  9. mv: Moves or renames a file or folder.
  10. cat: Displays the contents of a file.
  11. less: Displays the contents of a file one page at a time.

12. head: Displays the first few lines of a file.
13. tail: Displays the last few lines of a file.
14. grep: Searches for text patterns in a file.
15. find: Searches for files and folders in a directory tree.
16. chmod: Changes the permissions of a file or folder.
17. chown: Changes the owner of a file or folder.
18. ps: List all currently running processes.
19. top: Displays a real-time view of running processes and system resources.
20. kill: Terminates a running process.
21. tar: Creates or extracts compressed archive files.
22. wget: Downloads files from the internet.
23. ssh: Securely connects to a remote computer.
24. scp: Securely copies files between computers.
25. df: Displays the amount of free and used disk space.
26. du: Displays the amount of disk space used by a file or folder.
27. ifconfig: Configures and displays network interfaces.
28. ping: Tests network connectivity to another computer.
29. sudo: Allows you to run commands with root privileges.
30. useradd: Adds a new user to the system.
31. passwd: Changes the password of a user.
32. shutdown: Shuts down or restarts the system.
33. reboot: Reboots the system.
34. history: Displays a list of recently executed commands.
35. alias: Creates a shortcut for a command.

Configuring Git:

- Set your name globally: **git config --global user.name "Your Name"**
- Set your email globally: **git config --global user.email "youremail@example.com"**
- List your global Git configurations: **git config --list**

Clone & Status:

- Clone a repository from a URL: **git clone <repository_url>**
- Check the status of your local repository: **git status**

Git Status:

- Untracked: These are new files that Git doesn't yet track.
- Modified: Files that have been changed.
- Staged: Files that are ready to be committed.
- Unmodified: Files that have not been changed and are in their previous state.

Add & Commit:

- Add new or changed files to the staging area: **git add <file_name>**
- Commit the changes with a commit message: **git commit -m "Your commit message"**

Push Command:

- Upload local repository changes to a remote repository (usually on GitHub): **git push origin main**

Init Command:

- Initialize a new Git repository in your current directory: **git init**
- Add a remote repository (e.g., on GitHub) as the origin: **git remote add origin <repository_url>**
- Verify the remote repositories associated with your local repository: **git remote -v**
- Check the branches in your repository: **git branch**
- Rename the default branch from "master" to "main": **git branch -M main**
- Push the changes, including the branch name change, to the remote repository: **git push origin main**

  Ensure that you replace <**repository_url**> with the actual URL of the remote repository and <**file_name**> with the actual name of the file you want to add.

Branching and Merging:

- Create a new branch: **git branch <branch_name>**
- Switch to a different branch: **git checkout <branch_name>**
- Create a new branch and switch to it in one command: **git checkout -b <branch_name>**
- Merge changes from one branch into another: **git merge <branch_name>**
- Delete a branch (locally): **git branch -d <branch_name>**
- Delete a branch (remotely): **git push origin --delete <branch_name>**

Remote Operations:

- Fetch changes from a remote repository: **git fetch**
- Fetch changes and merge them into the current branch: **git pull**
- View remote branches: **git remote show origin**
- Remove a remote: **git remote remove <remote_name>**
- Clone a repository with a specific branch: **git clone -b <branch_name> <repository_url>**

Viewing Commit History:

- View commit history: **git log**
- View commit history in a more condensed format: **git log --oneline**
- View a specific commit's details: **git show <commit_hash>**

Undoing Changes:

- Discard changes in a file (unstaged): **git checkout -- <file_name>**
- Discard changes in a file (staged): **git reset HEAD <file_name>**
- Revert a commit (create a new commit that undoes the changes): **git revert <commit_hash>**
- Reset to a previous commit (careful, as it rewrites history): **git reset --hard <commit_hash>**.

- **git push origin main** pushes the local main branch to the remote origin repository.
  - **origin** is the default name for the remote repository from which you cloned your local repository.
  - **main** is the default name for the main branch in a Git repository, typically containing the latest production-ready code.
  - In Git, a "**remote**" is a named reference to a repository on another server.

- **Here is a summary of the steps for creating a new repository and adding programmes to a specific folder on github by command.**

1. **Create a new repository on GitHub.**
2. **Clone the new repository to your local machine:**
   - git clone https://github.com/ankitshukla1008/pldtCodebase.git
3. **Open the repository in VS Code:**
   - cd pldtCodebase
4. **Create a new Java folder:**
   - mkdir java
5. **Change directory to the Java folder:**
   - cd java
6. **Create a new Pyramid.java file:**
   - touch pyramid.java
7. **Write our program in the Pyramid.java file.**
8. **Save, compile and run the Pyramid.java file.**
9. **Navigate to the back pldtCodebase Folder:**
   - **cd ..**
10. **Initialize a new Git repository in the Java folder:**
    - git init
11. **Add the Java folder to the Git repository:**
    - git add java Or git add .
12. **Commit the changes with a commit message:**
    - git commit -m "Add a new folder java"
13. **Push the changes to the remote repository:**

- **If this is your first commit, you need to set the remote repository as the default branch:**
  - git push origin main
- **Push the committed changes to the remote repository (GitHub):**
  - git remote add origin https://github.com/ankitshukla1008/pldtCodebase.git
  - git push -u origin main

Once you have completed these steps, our new Java folder and Pyramid.java file will be added to our new repository on GitHub.

Undoing Changes (Undo):

Case 1: staged changes (add~): If i have staged changes (using git add) that i want to unstage (undo), i can use the following command:

git reset <- file name ->
git reset If we want to unstage all changes, we can simply use this command.

Case 2: commited changes (for one step back): Following command moves the HEAD pointer one step back in history, effectively "uncommitting" the last commit.

git reset HEAD~1

Case 3: commited changes (for many commits)

To undo multiple commits but keep the changes in our working directory (soft reset), we can use:
git reset <commit hash link>

If we want to completely discard the changes from the commits (hard reset), we can use:
git reset --hard <commit hash link>

- **Fork:** A fork is a complete copy of a Git repository, including all of its code, commit history, and branches. It shares code and visibility settings with the original "upstream" repository. It's a full duplicate, but it's also independent, meaning changes made to the fork won't affect the original repository, and vice versa. Forks are commonly used for collaborative development, allowing us to work on a project independently and then synchronize changes as needed. A fork is not a rough copy. It is a complete and accurate copy of the original repository, including all of its code, commit history, and branches for a specific purpose.

- **What is Git workflow and what are the different stages of a Git workflow?**
  A Git workflow is a process that teams use to manage their code development process using Git. It defines how developers create, branch, merge, and push their code changes. A well-defined Git workflow can help to improve the quality of code, reduce conflicts, and make it easier to collaborate with other developers.
  There are many different Git workflows, but most of them share the following stages:

  1. Branching: A new branch is created from the main branch to develop a new feature or fix a bug.
  2. Development: Developers work on their changes in the branch and commit their changes regularly.
  3. Testing: Once the changes are complete, they are tested to ensure that they work as expected.
  4. Review: Other developers review the changes to ensure that they are high quality and meet the team's standards.
  5. Merging: Once the changes have been reviewed and approved, they are merged into the main branch.
  6. Deployment: The changes are deployed to the production environment.

- **What are some common Git commands?**
  - → git init: Initializes a new Git repository in the current directory.
  - → git clone: Creates a copy of a remote repository on your local machine.
  - → git add: Stages changes for commit.
  - → git commit: Records changes in the repository along with a commit message.
  - → git status: Shows the status of your working directory and staged changes.
  - → git diff: Displays the differences between the working directory and the most recent commit.
  - → git branch: Lists, creates, or deletes branches.
  - → git checkout: Switches to a different branch or commit.
  - → git merge: Combines changes from one branch into another.
  - → git pull: Fetches changes from a remote repository and merges them into the current branch.
  - → git push: Pushes your changes to a remote repository.

- → git remote: Lists and manages remote repositories.
- → git log: Displays a history of commits.
- → git reset: Resets the repository to a previous state.
- → git stash: Temporarily saves changes that are not ready to be committed.
- → git tag: Lists, creates, or deletes tags to label specific commits.
- → git blame: Shows who made changes to a file and when.
- → git config: Configures Git settings, including user information and preferences.

- **What is Git cloning, and when would you use it?**

  Git cloning is the process of creating a copy of a remote Git repository on our local machine. When we clone a repository, we are creating a complete copy of the repository, including all of its code, commit history, and branches.

  There are many reasons why you might want to clone a Git repository.

  - **Starting a New Project:** When we begin working on a project, we clone the repository to set up our local development environment.
  - **Collaborative Development:** Cloning allows team members to work on the same project simultaneously, each having their own copy for making changes.
  - **Open Source Contribution:** If we want to contribute to an open-source project, cloning the project's repository is the initial step to access the code and contribute changes.
  - **Backup and Version Control:** Cloning provides a local backup of the repository, ensuring we have a copy of the code and its version history.
  - **Offline Work:** It enables us to work on the code, make changes, and commit them locally even when we are not connected to the internet.

    To clone a Git repository, we can use the following git clone command:

    git clone https://github.com/google/guava

- **How do you add changes to the staging area?**

  To add changes to the staging area in Git:

  1. **Use git add followed by file names or . (dot) to stage changes.**

     git add file1.txt file2.txt or git add .

  2. **Verify staging with git status.**

  3. **Commit staged changes with git commit**

     git commit -m "Your commit message here"

  4. **Push Changes to a Remote Repository**:

     git push origin main

- **What are Git tags, and why would you use them?**

  Git tags are labels that we can create to mark specific points in the history of our repository. They are typically used to mark important milestones or releases, such as a version, release, or major project update.

  Use of git tags:

  To mark important milestones or releases: This makes it easy to track the progress of our project and identify specific points in time when changes were made.

  To provide a reference point for other developers: This allows other developers to easily identify the commit that contains a specific feature or change.

  To create and build artifacts: Tags can be used to create build artefacts, such as release notes or installers.

  Create a tag: git tag v1.0.0

  Push the tag: git push origin v1.0.0

Git and GitHub

- **What are Git commits, and how do they work to changes the local repository?**
  Git commits are snapshots of our repository at a specific moment that are created by developers to record changes to the codebase. To commit changes, we should first stage them with git add, and then use git commit to create a new commit with a clear and concise commit message describing the changes. This is a fundamental process in Git for tracking and managing the history of our codebase.
  Here is a brief example of how to commit changes to our local repository:
  # Stage the changes that you want to commit
  git add .

  # Create a commit with a commit message
  git commit -m "This is a commit message."

- **How do you push changes to a remote repository on GitHub?**
  To push changes to a remote repository on GitHub:
  1. **Configure our Git identity with our name and email using git config commands.**
     git config --global user.name "Your Name"
     git config --global user.email "youremail@example.com"

  2. **Stage our changes with git add and create a commit using git commit with a meaningful message.**
     git add .
     git commit -m "Description of your changes"

  3. **Link Your Local Repository to a GitHub Repository** (If we haven't link):
     git remote add origin https://github.com/<username>/<repository>.git

  4. **Push your changes to GitHub with git push -u origin main (replace "main" with our branch name if needed).**
     git push origin main

- **How do you pull changes from a remote repository in Git?**
  To pull changes from a remote repository in Git:
  1. Navigate to our local repository.
     cd /path/to/your/repo

  2. Check our current branch.
     git branch

  3. Use git pull followed by the remote name (e.g., origin) and the branch name to pull changes. If it's the same branch, we can use git pull without arguments.
     git pull origin main

  4. Resolve any conflicts if they occur.

  5. Optionally, push our updated branch to the remote repository:
     git push origin main

- **What is a pull request and what are the benefits of using pull requests?**
  A pull request (PR) is a way to propose changes to a codebase. It is a collaborative process that allows other developers to review and comment on our changes before they are merged into the main codebase.

  To create a pull request, we first need to create a branch in our local repository and make our changes. Once we are happy with our changes, we can push our branch to the remote repository. Then, we can create a pull request to propose our changes to the maintainer of the codebase.

Benefits of using pull requests:

1. **Code review**: Pull requests allow for thorough code review, where other developers can examine and comment on our changes before merging them.
2. **Collaboration**: Multiple developers can work together on a feature or bug fix, enhancing collaboration and knowledge sharing.
3. **Transparency**: Pull requests provide visibility into the changes being proposed, the discussions around them, and the decision-making process.
4. **Reduced number of bugs**: Code reviews in pull requests help catch and fix bugs, leading to a more reliable codebase.
5. **Improved code quality**: With feedback and suggestions from peers, pull requests help maintain and improve code quality.
6. **Increased developer engagement**: Developers are more engaged and invested in the codebase when they have a voice in the review process.
7. **Approval Workflow**: Many Git platforms allow for approval workflows, ensuring that changes are reviewed and approved before merging.

- **How do you use Git to track the progress of a project?**
  1. Initialize: Create a Git repository with git init.
  2. Stage and Commit: Use git add and git commit for tracking changes.
  3. Branching: Create branches to isolate work on different features.
  4. Merging: Use git merge to consolidate changes from branches.
  5. View History: Check the project's evolution with git log.
  6. Collaborate: Share the repository, use pull requests, and review code changes.
  7. Tags: Mark important milestones with Git tags.
  8. Integration: Link Git with project management and issue tracking systems.
  9. Documentation: Keep project docs up to date in the repository.

- **What is a Git conflict, and how can you resolve it?**

  A Git conflict occurs when two or more commits have made changes to the same file or line of code. This can happen when two people are working on the same repository and make changes to the same file without communicating with each other. It can also happen when we merge two branches that have both made changes to the same file.

  To resolve a Git conflict:

  1) Identify the conflicting files.
  2) Open the conflicting files in a text editor.
  3) Identify the conflicting sections of the files.
  4) Decide which changes we want to keep.
  5) Save the files and commit the changes
  6) Push or share our changes.

- **How do you use Git to collaborate with other developers on a project?**

  To collaborate with other developers on a project using Git, we can follow these steps:

  1) Create a repository on a Git hosting service.
  2) Clone the repository to our local machine.
  3) Create a branch for each feature or bug fix.
  4) Make changes and commit them to our branch.
  5) Push our changes to the remote repository.
  6) Create a pull request (or merge request) for code review and merging.

7) After approval, merge our changes into the main branch.
8) Keep our local copy up to date.
9) Maintain effective communication with our team.

- **How do you use Git to revert to a previous version of a file or project?**
  To revert to a previous version of a file or project in Git, use the git checkout command.
  To revert to a previous version of a file:
  1. Identify the commit that contains the version of the file we want to revert to.
  2. Run the following command:
     git checkout <commit-hash> -- <file-path>
     Replace <commit-hash> with the commit hash and <file-path> with the path to the file we want to revert.

  To revert to a previous version of an entire project:
  1. Identify the commit you want to revert to.
  2. Use git reset to reset the entire project:
     git reset --hard <commit-hash>
     Replace <commit-hash> with the commit hash to which you want to revert the project**.**

- **How do you use GitHub to create and manage issues?**
  Creating Issues:
  - Go to your repository and click the Issues tab.
  - Click the New issue button.
  - Enter a title and description for the issue.
  - Assign labels, assignees, and milestones (optional).
  - Click the Submit new issue button.

  Managing Issues:
  - Assign and reassign issues as needed.
  - Use labels to categorize and prioritize issues.
  - Track progress with milestones.
  - Engage in discussions through comments.
  - Close resolved issues.
  - Reference issues in pull requests and commits.
  - Use GitHub's search and filtering for organization.
  - Configure issue templates and custom workflows as needed.

PS E:\Programs\Git> git --version

PS E:\Programs\git\LocalRepo> cd ..

PS E:\Programs\git> rm LocalRepo

PS E:\Programs\git> mkdir LocalRepo

PS E:\Programs\git> cd LocalRepo

PS E:\Programs\git\LocalRepo> git init → Initialized empty Git repository in E:/Programs/Git/LocalRepo/.git/

PS E:\Programs\git\LocalRepo> touch index.html →Create a new empty file inside a current folder like LocalRepo

PS E:\Programs\git\LocalRepo> git add . →git status →git commit -m "Add initial files"

PS E:\Programs\git\LocalRepo> git remote add origin https://github.com/ankitshukla1008/codingHubBasics.git

PS E:\Programs\git\LocalRepo> git remote -v → to verify remote

Git and GitHub

PS E:\Programs\git\LocalRepo> git branch to check branch

PS E:\Programs\git\LocalRepo> git branch -M main to rename branch

PS E:\Programs\git\LocalRepo> git pull origin main --allow-unrelated-histories

PS E:\Programs\git\LocalRepo> git push origin main

PS E:\Programs\Git\pldtCodebase> git branch see all branches

PS E:\Programs\Git\pldtCodebase> git checkout -b feature1 create a new branch

PS E:\Programs\Git\pldtCodebase> git checkout main switch to another branch “main”

PS E:\Programs\Git\pldtCodebase> git branch -d feature2 delete a branch, (Current branch should be another)
git diff <branch name> git merge <branch name>

PS E:\Programs\Git\pldtCodebase> git pull origin main used to fetch and download content from a remote repo and immediately update the local repo to match that content.

In pyramid.java file we can create our program then save, compile and run,

PS E:\Programs\git\pldtCodebase> git init

PS E:\Programs\git\pldtCodebase> git add java

PS E:\Programs\git\pldtCodebase> git commit -m "Add java to the repository"

PS E:\Programs\git\pldtCodebase> git push origin main → Now our created folder and pyramid have been added inside our new repository.

॥ॐ नमः शिवाय॥

|| ॐ तत्पुरुषाय विद्महे, महादेवाय धीमहि, तन्नो रूद्र प्रचोदयात् ||

♨ᘓ★ᘒ♨