### Repository Operations:

**1. \*\*Clone:\*\***

- Command: `git clone <link to .git>`

- \*\*Purpose:\*\* To create a copy of a remote repository on your local machine.

**2. \*\*Init:\*\***

- Command: `git init`

- \*\*Purpose:\*\* To initialize a new Git repository in the current directory, setting it up for version control.

**3. \*\*Remote Add:\*\***

- Command: `git remote add origin <link to .git>`

- \*\*Purpose:\*\* To add a remote repository URL to your local repository, typically named "origin".

### Working Directory and Staging Area:

**4. \*\*Status:\*\***

- Command: `git status`

- \*\*Purpose:\*\* To show the current state of the working directory and staging area.

**5. \*\*Add:\*\***

- Command: `git add <file>` or `git add .`

- \*\*Purpose:\*\*

- `git add <file>`: To add specific changes in the working directory to the staging area.

- `git add .`: To add all changes in the working directory to the staging area.

**6. \*\*Commit:\*\***

- Command: `git commit -m "commit msg"`

- \*\*Purpose:\*\* To commit the changes in the staging area to the repository with a descriptive message.

### Branching and Merging:

**7. \*\*Branch:\*\***

- Command: `git branch <branchname>`

- \*\*Purpose:\*\* To create a new branch with the specified name.

**8. \*\*Rename Branch:\*\***

- Command: `git branch -M <branchname>`

- \*\*Purpose:\*\* To rename the current branch to the specified name.

**9. \*\*Checkout:\*\***

- Command: `git checkout <branchname>`

- \*\*Purpose:\*\* To switch to the specified branch.

**10. \*\*Merge:\*\***

- Command: `git merge <branchname>`

- \*\*Purpose:\*\* To merge changes from the specified branch into the current branch.

### Collaboration and Remote Operations:

**11. \*\*Push:\*\***

- Command: `git push origin <branchname>`

- \*\*Purpose:\*\* To upload local repository content to a remote repository on the specified branch.

**12. \*\*Pull:\*\***

- Command: `git pull`

- \*\*Purpose:\*\* To fetch and merge changes from a remote repository into your local branch.

### Other Operations:

**13. \*\*Diff:\*\***

- Command: `git diff`

- \*\*Purpose:\*\* To show the difference between the working directory and the staging area.

### GitHub-Specific Operations:

**14. \*\*Pull Request on GitHub:\*\***

- \*\*Purpose:\*\* To propose changes from one branch (source branch) to another (target branch), typically from a feature branch to the main branch on GitHub. You can add a message to describe the changes being proposed.

**Conflicts**  
Conflicts in Git and GitHub happen when two or more people make changes to the same part of a file at the same time, or when one person changes something in a file that another person has deleted.

Fixing conflicts in Git involves a few steps:

1. \*\*Identify Conflicts:\*\* When you try to merge or pull changes, Git will notify you if there are conflicts. You'll see markers in the conflicting files indicating where the conflicts occurred.

2. \*\*Open Conflicting Files:\*\* Use a text editor to open the conflicting files. Inside these files, you'll see special markers like `<<<<<<<`, `=======`, and `>>>>>>>` that indicate the conflicting sections.

3. \*\*Resolve Conflicts:\*\* Decide which changes to keep and which to discard. Remove the conflict markers (`<<<<<<<`, `=======`, and `>>>>>>>`) and make any necessary adjustments to the content to resolve the conflicts. You might need to discuss with your team to understand the conflicting changes better.

4. \*\*Save Changes:\*\* After resolving conflicts, save the changes to the file.

5. \*\*Add and Commit:\*\* Once conflicts are resolved, stage the resolved files using `git add` and then commit the changes using `git commit`.

6. \*\*Complete Merge or Rebase:\*\* If you were in the middle of a merge or rebase operation when conflicts occurred, you can now complete the operation by running `git merge --continue` or `git rebase --continue`.

7. \*\*Test:\*\* After resolving conflicts, it's a good practice to test your changes to ensure everything works as expected..