Git (Version Control) :

1. Git Clone :

```
git clone {GIT-REPO-URL}
```

- When you perform a `git clone`, the data is cloned from the default branch of the repository. Traditionally, this default branch has been named `master`. However, many repositories have shifted to using `main` as the default branch name.
- Cloning from a particular branch :

```
git clone -b <branch_name> <repository_url>
```

2. To See Branch :

- To see all the branches (Both local and remote):

```
git branch -a
```

- To see all the local branches :

```
git branch
```

- To see all the remote branches :

```
git branch -r
```

- **Verbose Listing**: You can add the `-v` option to get more information about each branch, such as the commit message that the branch points to:

```
git branch -av
```

- **Graphical Representation**: For a more visual representation of branches and their commits, you can use:

```
git log --oneline --graph --decorate --all
```

3. To switch / change Branches :

- To switch or change branches in Git use the `git checkout` command.

```
git checkout <branch_name>
```

- If you want to switch to the previously checked out branch (often useful for toggling between two branches), you can use:

```
git checkout -
```

- **Switch to a Remote Branch** :
  If you want to switch to a branch that exists on the remote repository (but isn't tracked locally yet), you can use:

```
git checkout -t <remote_name>/<branch_name>
```

Here, `<remote_name>` is typically `origin`, and `<branch_name>` is the name of the branch on the remote repository.

Notes:

- **Detached HEAD State**: If you checkout a specific commit rather than a branch name (`git checkout <commit_hash>`), Git enters a "detached HEAD" state, where you are not on any branch. It's important to create a new branch or checkout an existing branch to continue working safely.

- **Uncommitted Changes**: Git will prevent you from switching branches if you have uncommitted changes that conflict with the switch. You can either commit your changes (`git commit -m "message"`) or stash them (`git stash`) before switching branches.

4. Create / delete / rename Branch :

- **Create a New Branch and Checkout**:
  To create a new branch and immediately switch to it, use the `-b` option with `git checkout`

```
git checkout -b <new_branch_name>
```

- Create a Branch : To create a new branch in Git, you use the `git branch` command followed by the branch name:

```
git branch <branch_name>
```

- Delete a Branch : To delete a branch in Git, you use the `git branch -d` command followed by the branch name. Make sure you are not currently on the branch you want to delete.

```
git branch -d <branch_name>
```

- If the branch has unmerged changes (commits that are not in the current branch), Git will refuse to delete it unless you use `-D` option, which forces deletion:

```
git branch -D <branch_name>
```

- Rename a Branch : To rename a branch in Git, you use the `git branch -m` command followed by the current branch name and the new branch name.

```
git branch -m <current_branch_name> <new_branch_name>
```

Notes:

- **Creating from Current Branch**: If you want to create a new branch starting from your current branch, you can use `git branch <new_branch_name>` while on that branch.
- **Deleting the Current Branch**: You cannot delete the branch you are currently on. Switch to another branch first (`git checkout <other_branch>`), then delete it.
- **Renaming Branches**: Renaming a branch only changes its name locally. If you have already pushed the branch to a remote repository, you'll need to push the renamed branch and delete the old one on the remote as well.

5. Stage Changes (`git add`) :

- The `git add` command in Git is used to stage changes for commit. It tells Git which files you want to include in the next commit snapshot.
- `git add` stages changes in your working directory, preparing them to be included in the next commit on whatever branch you are currently on.
- When you modify files in your working directory, Git initially considers these changes as "un-staged". This means Git is aware of the changes but has not yet recorded them as part of your project's history.
- By using `git add`, you specify which modifications (or new files) you want to include in the next commit.
- **Selecting Files for Commit**:
  - You can add specific files or directories to the staging area by specifying their paths with `git add`. For example:

```
git add file1.txt     # Stage a specific file

git add directory/    # Stage all files in a directory
```

```
git add file1.txt file2.txt # Stage 2 files
```

- Alternatively, you can stage all modified files (excluding untracked files). This command stages all changes in the current directory and its subdirectories :

```
git add .
```

6. Commit :

- The `git commit` command is used to save your changes to the local repository in Git. When you commit, you are recording a snapshot of the currently staged changes in your project, along with a descriptive message.

What `git commit` Does:

1. **Creates a New Commit**:
   - A commit is a snapshot of the current state of your project's files that have been staged using `git add`.
   - Each commit has a unique identifier (a SHA-1 hash) that allows Git to track changes over time.
2. **Records Changes**:
   - Commits record the differences (or "diffs") between the current state of the project and the previous state.
   - Only changes that have been staged (using `git add`) are included in the commit.
3. **Adds Metadata**:

- Each commit includes metadata such as the author, the commit message, and the timestamp.
- The commit message, provided by you, describes the changes made and why they were made.

4. **Maintains Project History**:
    - Commits form a history of your project's development, allowing you to review and revert to previous states if needed.
    - The project history is a series of commits, often visualized as a graph of branches and merges.

```
git commit -m "Your commit message"
```

- **Interactive Commit**:

  ```
  git commit
  ```

  This command opens the default text editor, allowing you to write a more detailed commit message interactively.

- **Amending a Commit**:

  ```
  git commit --amend
  ```

  This command modifies the most recent commit, allowing you to change the commit message or add new changes to the previous commit.