**1. Setup and Configuration**

* git version :The command git version is used to check the version of git.

**git --version**

* gitconfig: Configures Git settings. Commonly used to set up user information

**gitconfig --global user.name "Your Name"**

**gitconfig --global user.email "youremail@example.com"**

* **gitconfig --list**: Displays all the Git configurations for the current user.

**2. Repository Management**

* **git init**: Initializes a new Git repository in the current directory

gitinit

* **git clone**: Creates a copy of an existing Git repository from a remote source (e.g., GitHub) to your local machine

git clone https://github.com/username/repository.git

**3. Staging and Committing**

* **git status**: Shows the status of changes in your working directoryand staging area. It tells you which files are untracked, modified, or ready to be committed.

**git status**

* **git add**: Adds changes in the working directory to the staging area.

**git add filename # Adds a specific file**

**git add . # Adds all changes in the directory**

* **git rm / git restore**:Unstage the added files

**git rm –cached filename**

**git rm –staged filename**

**git restore –staged filename**

* **git commit**: Commits the staged changes to the repository with a descriptive message. The -m option allows you to include a commit message.

**git commit -m "Commit message describing changes"**

**4. Branching and Merging**

* **git branch**: Lists all branches or creates a new branch

**git branch # Lists all branches**

**git branch branch-name # Creates a new branch**

**git branch -d <branch\_name> # Deletes a branch**

* **git checkout**: Switches to a different branch

**git checkout branch-name # Switches to an existing branch**

**git checkout -b new-branch # Creates and switches to a new branch**

* **git merge**:Merges the specified branch into the current branch. This command integrates the changes from the feature branch into the main branch.

**git checkout main # Switch to the main branch**

**git merge branch-name # Merge branch-name into main**

1. **Undoing Changes**

* gitrestore :used to **undo changes** in your working directory or staging area without affecting commit history.

**git restore <file> Restores file to last committed version**

**git restore --staged <file> Moves file from staging back to working area**

**git restore –source=<commit><file> Gets version of the file from a commit**

* git reset :Removes the specified file from the staging area but leaves the working directory unchanged. git reset --hard can also reset the working directory and staging area to the last commit.

**git reset <file>:**

* git revert**:** Creates a new commit that undoes the changes from a specified commit, leaving the history intact**.**

**git revert <commit>:**

1. **Viewing History**

* git log**:** Shows a history of commits in the repository, including commit hashes, messages, and timestamps.

**git log Detailed**

**git log –oneline Concise**

**git log -online –graph Graphical**

* git diff**:**Displays differences between various commits, the working directory, and the staging area. git diff without arguments shows changes not yet staged.

**git diff**

* git show**:**Shows the details of a specific commit, including the changes made and the commit message.

**git show <commit>**

**Working with GIT HUB**

* **git remote:** Adds a new remote repository. This is useful when you want to link your local repository with a remote one, like GitHub.

**git remote add <name><url>**

* **git push**: Used to transfer the commits or pushing the content from the local repository to the remote repository. The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

**git push –u origin <master | current directory>**

* **git clone**: used to create a local working copy of an existing remote repository.The command downloads the remote repository to the local computer.

**gitclone <remote url>**

* **git fetch** : Downloads changes from a remote repository without applying them to your working directory. You can later merge these changes.

**git fetch <remote>**

* **git pull**: Fetches and integrates changes from a remote repository by rebasing instead of merging. This creates a linear history.

**git pull <remote><branch>**

**Git Patch: A Powerful Tool for Sharing and Applying Changes**

A patch file contains the diffs between two versions of a file or set of files. It’s a way to capture changes made in a commit, so you can:

* Send them to others (via email or other means).
* Apply them to a different repository (or different branch).

**Key Benefits of using Patches:**

* **Offline Contributions**: You can make changes without pushing them to a remote repo.
* **Easy Collaboration**: Share small changes as patches instead of full repositories.
* **Lightweight**: Patches are lightweight and easy to transfer via email.
* **Open Source Contributions**: Submit patches to a project without direct commit access.
* **Code Reviews**: Share changes with team members for review without pushing to a remote.
* **Collaborative Development**: When working in a restricted environment (e.g., no internet), patches can be used to exchange changes.
* **Backup of Changes**: Save patches as backups before applying risky changes.

**Creating a Git Patch**

**git format-patch :**creates a patch. Generates one or more .patch files

**git format-patch -1 HEAD** -1 refers to last commit

HEAD refers to current commit

**(**createsa . patch file with the name**0001-<commit-message>.patch )**

**git format-patch HEAD~3** Patches for the last 3 commits

(generates 3 .patch files, each corresponding to one commit)

**You can send a .patch file to someone via email, file sharing, or any other method.**

**Applying a Git Patch**

**gitapply :** This will apply the changes from the .patch file to your working directory (but not commit them yet).

**git apply <patch-file>**

**Inspect the changes, stage and commit the changes to your repository.**