Git and Linux Command Guide

Linux Commands (Necessary Before Understanding Git Commands)

1. `cat <filename>`
   Example: `cat notes.txt`
   Displays the content inside `notes.txt`.

2. `ls`
   Lists files and directories in the current directory.

3. `cd ..`
   Moves up one directory level (to the parent directory).
   Example:
   If you're in `/home/user/Desktop`
   `cd ..` takes you to `/home/user`.

4. `ls -a`
   Lists all files, including hidden files (those starting with a dot `.`).
   Example:
   `ls -a` will show files like:
   `.  ..  .bashrc  .gitignore  myfile.txt  .git`

Git Introduction

- Git saves changes/history of code or project.
- Useful when many developers are working on the same project.
- It tracks changes, bugs, and saves every version of code as a new file.
- Useful to resolve bugs and maintain history of code.

How to Use Git?

Start Tracking

- **To start tracking a folder:**
  `git init`
  (Start tracking this folder)

- **To start tracking a file inside a folder:**
  `git add <filename>`
  (Git, please version control this file)

- **Track all files:**
  `git add .`
  (Starts tracking all the files)

Git Status Codes

- `U` - Untracked
- `A` - Index added (file added to git index)
- `M` - Modified

Inside `.git` folder, in the index folder, Git stores info about files to track.
Untracked files are not shown by `cat index`.

`git status`

Displays currently tracked files in a simple UI instead of using:
`cd .git && cat index`

All Git commands perform read/write operations inside the `.git` folder.
Tracking happens inside your project folder.

If you transfer your project folder (via pendrive, etc.), the `.git` folder also gets transferred.

If you type `git status`, it shows uncommitted and staging-phase files.

Staging and Committing

Add to Staging

```
git add .
```

Commit

```
git commit -m "Message to send when committing"
```

Example:

```
git commit -m "Add login form validation"
```

This creates a snapshot of your project's state and tracks changes over time.

- **Commit = Checkpoint (Snapshot)**
- Every commit has a unique ID (hash)

View Commit History

`git log`

Shows commit history:

- Who made the commits?
- Commit ID
- Date of commit
- Commit message

Diff and Commit Details

- `git diff`: Shows exact line-by-line changes that are not committed yet.
- `git cat-file -p <commit_id>`: Shows commit message.
- `git cat-file -s <commit_id>`: Shows commit size.
- `git cat-file -t <commit_id>`: Shows the type of action (e.g. commit).

Git is **case-sensitive**.
Files in `.git` are compressed and not human-readable by default.
`git cat-file -p <hash>` decompresses and shows the content.

Most Used Git Commands

```
git init                  --|
git add .                 --|----> 98% of usage
git commit -m "Message"   --|
git log      --|
git status   --|----> 0.5% of usage
```

How Is the Git Commit Hash Made?

1. Read all files in the folder.
2. Hash it using **SHA1** algorithm.
3. The result is your commit hash.

If no changes are made between `add` and `commit`, the hash remains the same.
Git shows:
"nothing to commit, working tree clean"

Git Internals and CLI Best Practices

- Hash = Unique identifier of commit.

- Never interact with hashes manually.
- Never modify `.git` folder directly.
- Always use Git CLI.

Going Back and Forward in Commits

Reset to a Commit

```
git reset <commit_id>
```

To go forward/backward to specific commits.

Delete/Discard a Commit

```
git reset --hard <commit_id>
```

Deletes commits. Cannot go forward/backward afterward.

.git Folder Location

- It lives in your **local machine**.

Collaboration: Local vs Remote

Suppose you're sharing code with a friend via pendrive:

- Copy the project to pendrive → `.git` folder goes too.
- Friend can type `git log` to see all commits.
- If your friend makes a new commit and gives it back:
  - You'll also see their commits.

Problem:

Pendrive is **not** a central source of truth.

If both people modify the same file, which version wins?

Solution: GitHub

- GitHub acts as a **central repository**.
- Both people can push/pull code remotely.
- No conflict. Smooth collaboration.

Upload Files to GitHub

1. Create a repo on GitHub.
2. Copy the "git remote add origin" command:

```
git remote add origin
https://github.com/dharmikbhuva2784/Git_GitHub_Concepts.git
git push -u origin main
```

This pushes your whole code to GitHub.

Assignment

- Company: **ChaiCohort**
- Goal: Prepare onboarding documentation

When an employee or intern joins:

- They should get a **guide on Git usage**

Task:

Create a repo named `ChaiCohort` and add a file with:

- Basic Git commands
- Full Git summary and `.git` folder architecture
- Collaboration precautions
- Commit rules:
  - Message should be meaningful
  - If solving a ticket, message should include ticket ID