Git & GitHub Detailed Notes

1. Git Basics

Git is a version control system that tracks changes in source code.

Every project with git has a hidden `.git` folder that stores history, configs, and commits.

■■ Never modify files inside `.git/` manually.

Useful commands:

`pwd` → Print working directory

`ls -la` → List files (with hidden like .git)

`cd ..` → Go to parent directory

`git init` → Initialize a new git repository

Example:

mkdir myproject && cd myproject && git init

2. Workflow: Working Directory → Staging → Repo

Git workflow moves files across three areas:

- Working Directory → Staging Area → Local Repo → Remote Repo (GitHub)

Commands:

`git add file.txt` → Add file to staging area

`git add .` → Stage everything

`git commit -m "Message"` → Commit staged files

`git commit -am "Message"` → Add & commit all tracked files in one go

`git rm --cached file.txt` → Unstage a file

`git restore --staged file.txt` → Another way to unstage

Example:

echo "Hello" > file.txt

git add file.txt

git commit -m "Added file.txt"

3. Commits & Logs

Commits are snapshots of your project with unique IDs (hashes).

Each commit (except the first) has a parent commit.

Commands:

`git log` → Show full history

`git log --oneline` → Compact view

Example:

git log --oneline Output: e62d860 Initial commit 444f07f Added navbar

4. Configuring Git

`git config --global user.name "Your Name"` → Set your identity

`git config --global user.email "you@example.com"` → Set email

`git config --global core.editor "code --wait"` → VS Code as editor

`cat ~/.gitconfig` → Check config file

5. Branching & HEAD

Branches allow alternate timelines for development.

HEAD points to the current branch/commit.

Commands:

`git branch` → List branches

`git branch nav-bar` → Create branch

`git switch nav-bar` / `git checkout nav-bar` → Move to branch

`git switch -c dark-mode` / `git checkout -b blink-mode` → Create + switch

`git merge nav-bar` → Merge branch into current

`git branch -d nav-bar` → Delete branch

Rules: Always commit before switching branches.

6. Comparing Changes

`git diff` → Working directory vs staging

`git diff --staged` → Staging vs last commit

`git diff hash1 hash2` → Compare commits

`git diff branch1..branch2` → Compare branches

Symbols:

--- old version, +++ new version

7. Stash

Used when you need to switch branches but have uncommitted changes.

`git stash` → Save changes

`git stash list` → See stashes

`git stash pop` → Restore and remove stash

`git stash apply stash@{0}` → Apply without removing

Example:

git stash git switch bugfix git stash pop

8. Checkout, Restore & Reflog

`git checkout ` → Go to specific commit

`git checkout HEAD~2` → Go two commits back

`git restore file.txt` → Restore file to last commit

`git reflog` → Show history of HEAD (to recover lost commits)

9. Rebase

Rebase moves commits of one branch on top of another for a clean history.

`git rebase master` → Replay branch commits onto master

■■ Do NOT rebase master/main or commits already pushed to GitHub.

During rebase:

- Resolve conflicts manually
- Stage resolved files
- Run `git rebase --continue`
- To skip commit: `git rebase --skip`
- To abort: `git rebase --abort`

10. Remote & GitHub

`git remote -v` → List remotes

`git remote add origin ` → Add remote

`git remote rename old new` → Rename

`git remote remove name` → Remove remote

`git push origin main` → Push branch to remote

`git push -u origin main` → Set upstream (future pushes need only `git push`)

`git push local:remote` → Push with different branch names

`git clone ` → Clone repo

`git fetch` → Get remote info without merging

`git pull` → Fetch + merge into current branch

11. GitHub Workflow

Steps to contribute:

1. Fork repo → your account
2. `git clone ` → Copy repo locally
3. Create a new branch (`git switch -c feature`)
4. Make changes → `git add` → `git commit`
5. Push: `git push origin feature`
6. Create Pull Request (PR) on GitHub

Tips: Always work in branches, not on main.