**Version Control System (VCS)**

Tracks changes in files (mainly code), helps multiple people work together, and allows going back to previous versions.

Version control system is the tools that helps to Track changes in the code ,

Popular, free and open source,fast & scalable

**Centralized Version Control System (CVCS): Example: SVN, CVS**

* **Single central server** stores all versions.
* Developers get the latest copy from the server.
* Requires **internet connection** to commit or view history.

#### Pros

* Simple to set up
* Easy to understand

#### Cons:

* If server fails = **data loss**
* Limited offline work

**Distributed Version Control System (DVCS)**

Example: **Git, Mercurial**

* Every developer has a **full copy** of the entire repo (including history).
* Can **work offline** and sync later.
* Faster operations (commit, diff, log).

#### Pros:

* Works offline
* **Safe** (backup in every copy)
* Faster performance
* **Branching and merging** is easy

#### Cons:

* Slightly more complex
* Initial clone is heavier

Git is a **Distributed Version Control System (DVCS)** that helps developers track changes in their code, collaborate with others, and manage different versions of a project.

## ****1. Basic Git Concepts****

### ****Repository (Repo)****

* A **repository** is like a project folder where Git tracks all changes.

A repository (or repo) is a folder where Git tracks all changes to your files.

It contains:

* Working Directory /Work Space : Your actual files. **Working Directory**: Where you edit files.
  + Staging Area : Files you’ve marked for the next commit.  Files ready to be committed (git add).
  + Commit History : A record of all changes.
  + **Repository (Git Directory)**: Where commits are stored.
* **Local Repo**: Stored on your computer.
* **Remote Repo**: Hosted online (GitHub, GitLab, Bitbucket).

**Commit :** A snapshot of your changes with a message. Think of it as **"Save point"** in your code history.

### Install:

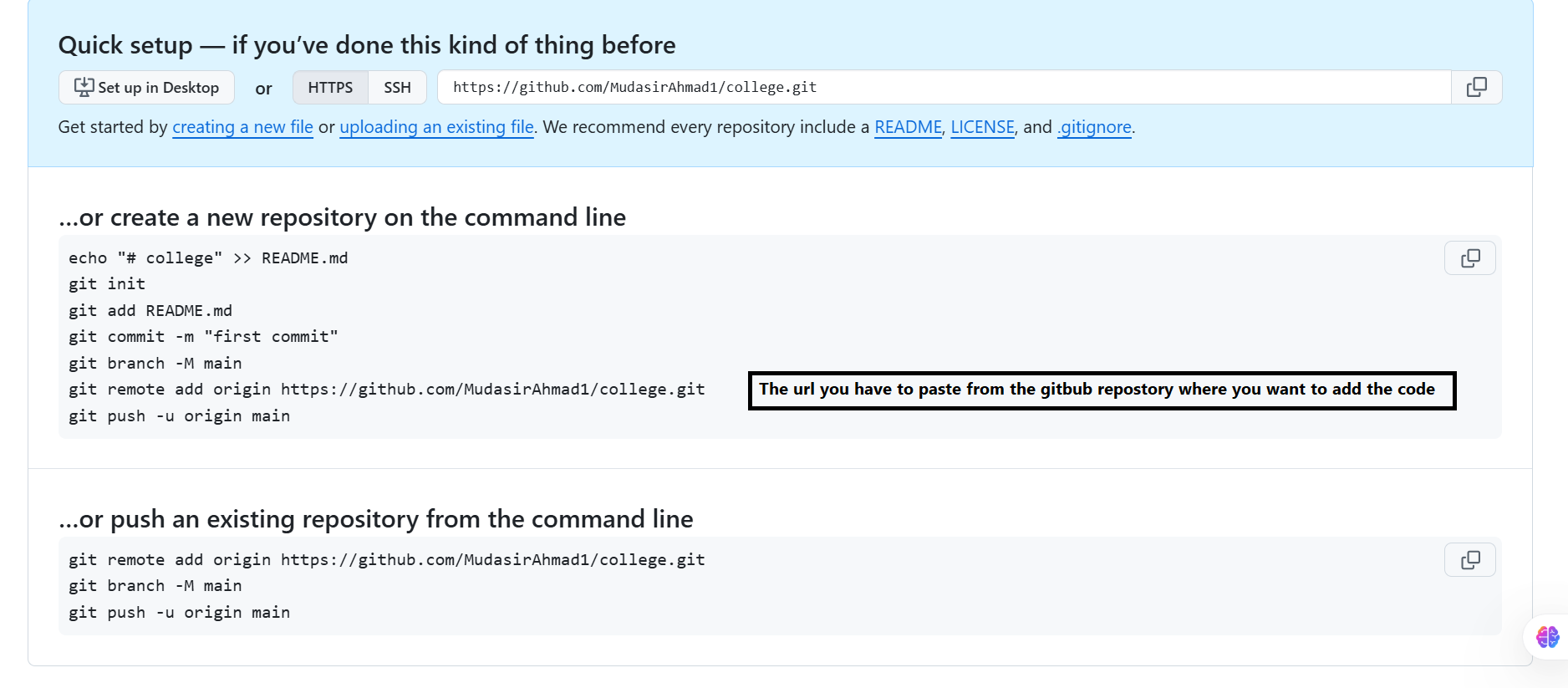
sudo apt-get install git

### Set up:

* git config --global user.name "User Name" //enter your username
* git config --global user.email "email@email.com" //enter github email
* git config -- list //show the details username and email

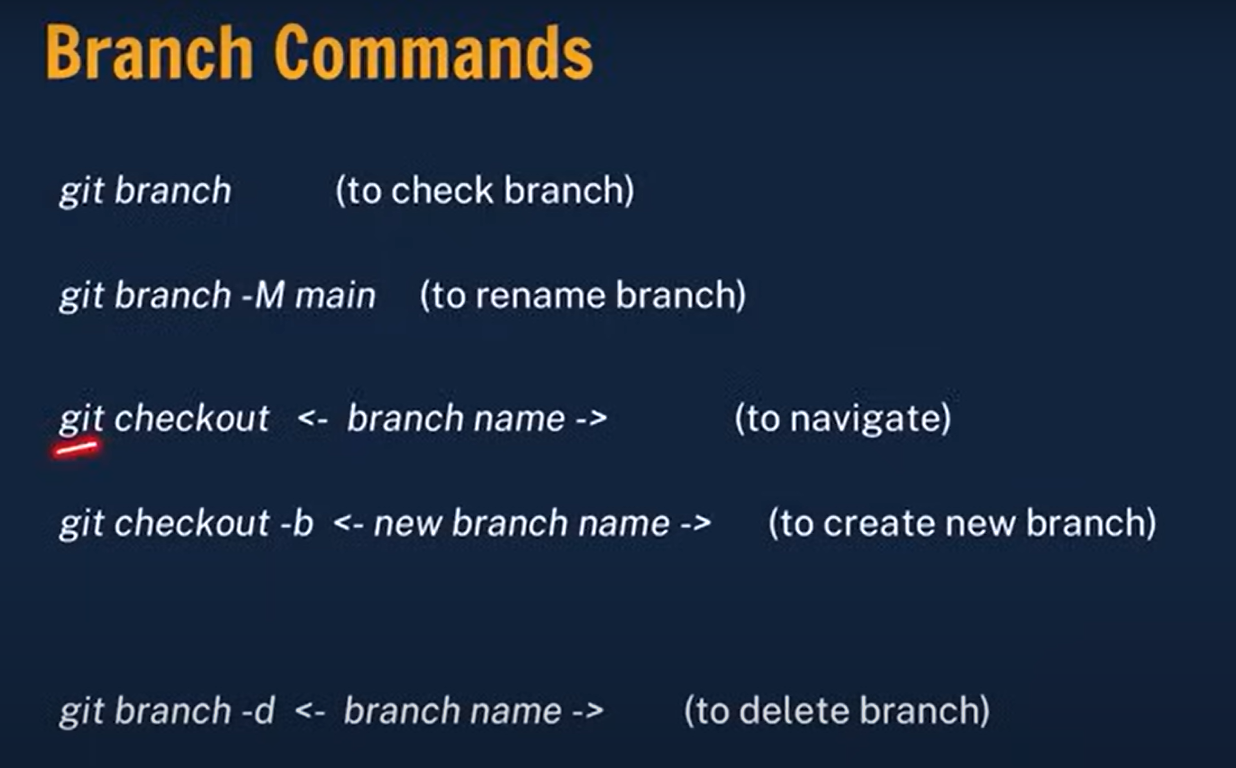
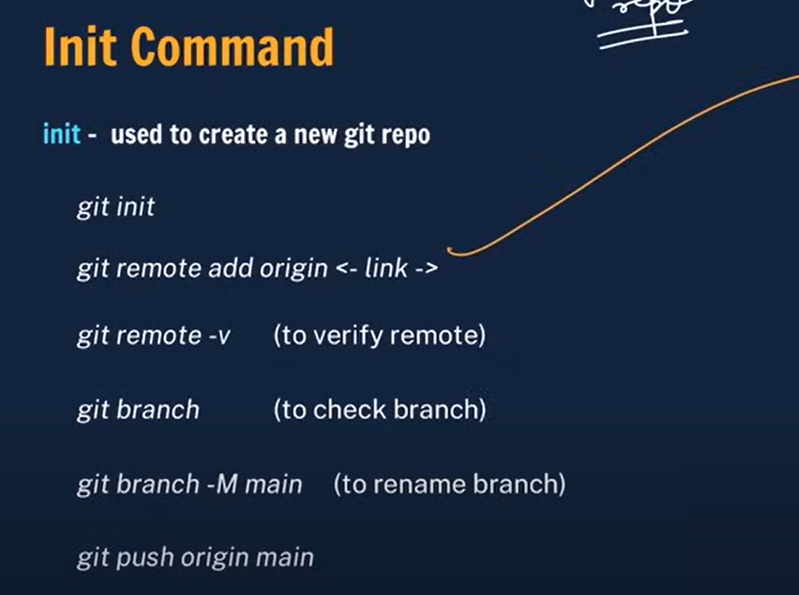
### Create a New Project:

* mkdir project
* cd project
* git init . (initialise empty git repo in my folder) initialize the new repo
* ls -la (check my folder)
* git add . (meaning add all files )
* git add filename (meaning add filename only)



Clone :cloning a repository on your local machine // if u see a repo in the git and want to download on your system simple you copy the url and follow the command .

* git clone <https://github.com/rohitg00/DevOps_Books.git> //like this
* Git status //**git status** shows the current state of your Git working directory and staging area.
* Ls –a //shows the hidden file to check git file is there enter ls and you will see .git file
* Untracked : new file that git does not track yet..
* Modified: changed.
* Staged : fie is ready to be commited.
* Unmodified : unchanged.
* Git push origin main // upload local repo content to remote repo.



**Branch: A branch represents an independent line of development.**

* **Each task has one separate branches after done and review then merge with master.**
* **Parallel development.**
* **Error free master branch.**
* **Git branch branch-name //you can create like that too**

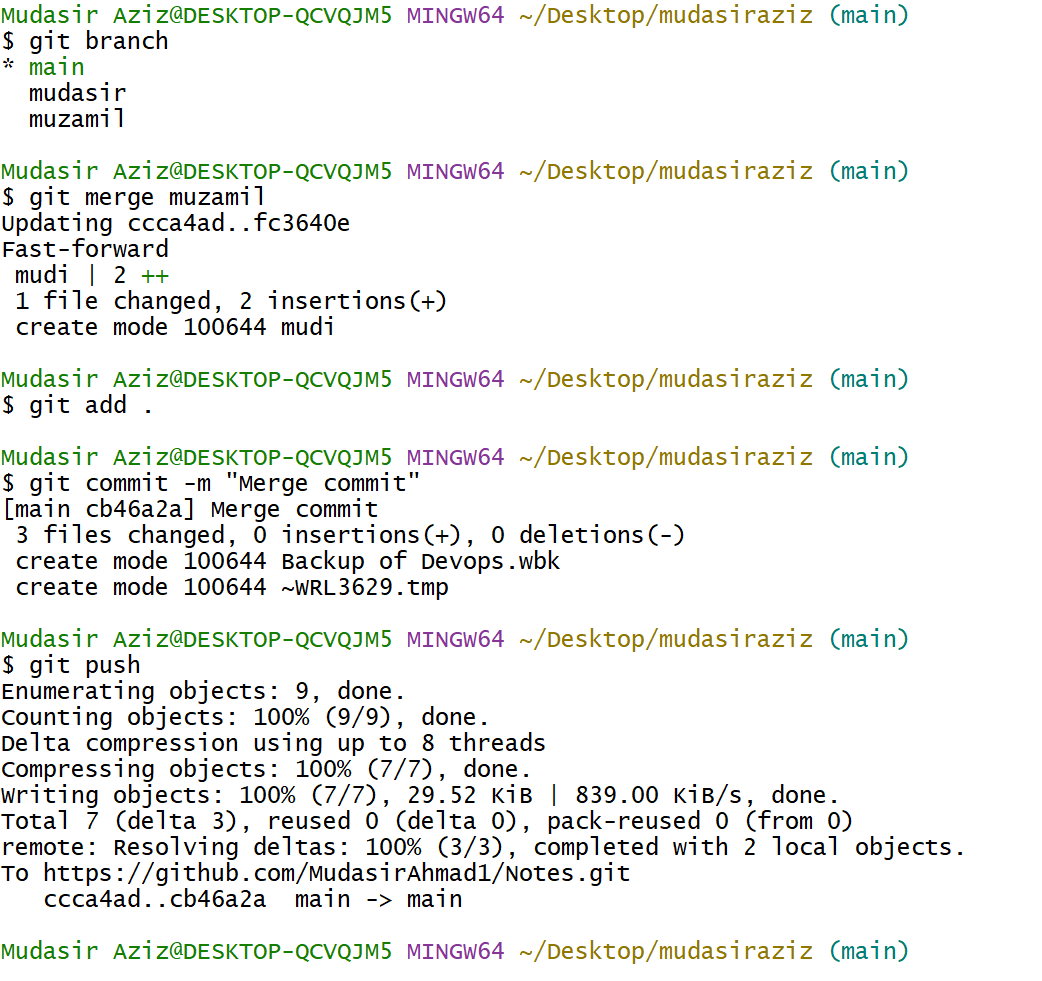
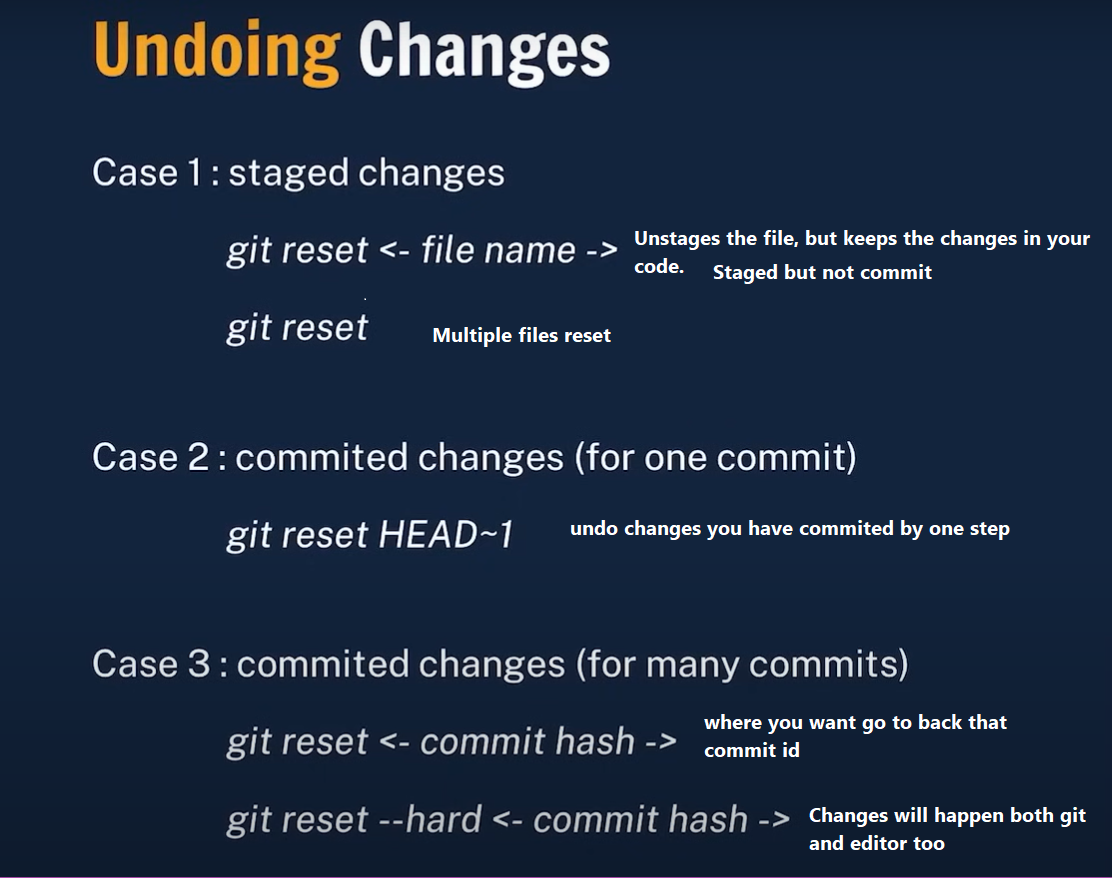
## ****Handling Merge Conflicts:****

### ****What is a Conflict:**** Occurs when Git cannot automatically merge changes (e.g., two people edit the same part of a file).

### ****How to Resolve:****

1. Open the conflicting file.
2. Look for conflict markers (**<<<<<<<**, **=======**, **>>>>>>>**).
3. Edit the file to resolve the conflict.
4. Mark as resolved:
5. git add <file>
6. git commit

If you deleted a file accidentally: git restore filename



### ****Q1: What is Git?****

* **Answer:** Git is a distributed version control system used to track changes in source code during software development.
* Git tags : allows give you meaning name of the commit
* Git ignore : ignore specifics files which we don’t want to commit or add

### ****Q2: What is the difference between**** git pull ****and**** git fetch****?****

* **Answer:**
  + **git fetch**: Downloads changes from the remote but does not merge them.
  + **git pull**: Downloads changes and merges them into your current branch.

### ****Q3: What is a Git branch?****

* **Answer:** A branch is a separate line of development that allows you to work on features or fixes without affecting the main codebase.

### ****Q4: How do you resolve a merge conflict?****

* **Answer:** Identify the conflicting file, edit it to resolve the conflict, stage the resolved file, and complete the merge with **git commit**.

### ****Q5: What is the purpose of**** .gitignore****?****

* **Answer: .gitignore** specifies files or directories that Git should ignore (e.g., logs, temporary files).

### ****Q6: What is the difference between**** git merge ****and**** git rebase****?****

* **Answer:**
  + **git merge**: Combines branches while preserving the history.
  + **git rebase**: Reapplies commits to create a linear history.

### ****Q7: What is a detached HEAD state?****

* **Answer:** A detached HEAD occurs when you check out a specific commit instead of a branch. Any changes made here will be lost unless you create a new branch.

### ****Q8: How do you undo a commit?****

* **Answer:**
  + To undo the last commit but keep changes:

git reset --soft HEAD~1

* + To completely remove the last commit:

git reset --hard HEAD~1

### ****Q9: What is the difference between**** origin ****and**** upstream****?****

* **Answer:**
  + **origin**: Your forked repository.
  + **upstream**: The original repository from which you forked.

### ****Q10: What is a**** bare ****repository?****

* **Answer:** A bare repository has no working directory and is used for sharing code (e.g., on GitHub).

### ****Explain****git stash****?****

* Temporarily saves uncommitted changes:
* git stash # Save changes
* git stash pop # Restore changes

### ****What is a "fast-forward" merge?****

* When Git moves a branch pointer forward without creating a merge commit.

**Commit Small & Often** (Atomic commits)  
**Write Clear Commit Messages** (e.g., "Fix login bug" not "Update code")  
**Use Branches for Features/Bugs**  
**Pull Before Push** (Avoid conflicts)  
**Rebase for Clean History** (Instead of merge)  
**Never Commit Secrets** (Passwords, API keys)

