Day 3 - 21st  May 2025

DevOps / Cloud Fundamentals:

Devops Basics

Introduction to Git, Git- Tracking your changes, Git understanding index area and local repositories, Git branches, Merging branches in Git.

Introduction to cloud, Introduction to Jenkins,  Introduction to Chef, Introduction to Containers, Introduction to DevOps — CI/CD, Meaning, Principles & Examples, DevOps Lifecycle — Different Phases Explained with Examples, Introduction to Docker, Purpose of using Docke

Complete  Git tutorial

<https://docs.google.com/document/d/1SXfPIwDttDk-XP5_mX7YizIuJGQ2vKMX/edit?usp=sharing&ouid=103020917863623365252&rtpof=true&sd=true>

Task 1:

Recap of Last session:

Create a file names 21st May.txt and push it to your git hub.

5min 9.47 to 9.52

Git log

Git show

Git branch - list all the branches

Git branch PrasunambaBranch - create a new branch

Git branch

Git checkout targetbranch name

Git checkout PrasunambaBranch (this will switch the branch)

If i want to merge PrasunambaBranch  to master branch

Git checkout master

  (first move to the target branch)

Git merge PrasunambaBranch

(merger PrasunambaBranch  to master branch

=======================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

=======================================================================

Database Fundamentals

**NoSQL**

Relational vs Non Relational Databases: Relational vs Non Relational Databases

Introduction to NoSQL: Downloading the required Software,  NoSQL Architecture with MongoDB

CRUD and the MongoDB Shell: Basics, MongoDB Advantages, MongoDB Data Modelling, MongoDB Tools

Querying Data: Databases and Collections, Querying Collections, Working with Operators, Referencing a Database, Querying Dates

Manipulating Data: Inserting Data into Collections, Updates, Deletes, Atomic Operations, Removing Data, Capped Collections

Relational vs Non Relational Databases: Relational vs Non Relational Databases

—-----------------------------------------------------------------------------------------------------------------

Home Task:

**git stash :**

Git stash is a command used to temporarily store changes that are not ready to be committed to the repository. It’s helpful when you want to switch branches, but you don’t want to commit your changes or lose them. Stashing allows you to save your work in progress and come back to it later.

When you run “git stash”, Git will:

1. Take all the changes in your working directory and index (staging area) and save them as a “stash”.
2. Reset your working directory and index to the last commit, effectively removing the changes from view.
3. You can later apply the stashed changes back to your working directory using “git stash apply” or “git stash pop” commands.

It’s a convenient way to manage your work-in-progress changes without cluttering your commit history or losing any work.

**Important Command for git stash :**

# To stash an item   
git stash  
  
# To see the stashed items list  
git stash list  
  
# To apply stashed items  
git stash apply stash@{number}  
  
# To clear the stash item  
git stash clear

**git reset :**

Git reset is a command used to undo changes in your working directory to staging area.

git reset <file name>  
        or  
git reset .

See the workflow of git reset :

Git reset and git revert

Git reset – unstage (after add u can unstage)

Add X reset

Git revert - uncommit (after commit )

Commit X revert

**Removing Untracked File :**

git clean -n   #(ask for delete yes or no)  
git clean -f   #(forcefully)

**Tags :**

It is used to gives meaningful names to a specific version in the repository.

git tag -a <tag name> -m <message> <commit-id>  
  
# To see the list of tags  
git tag  
  
# To see particular commit content by using tag  
git show <tag name>  
  
# To delete a tag  
git tag -d <tag name>

ACID

A - Atomicity

• All operations in a transaction must succeed, or none of them will.

• It's "all or nothing" - if any part fails, the entire transaction is rolled back.

C - Consistency

• Data must be valid according to defined rules.

• Transaction brings database from one valid state to another valid state.

I – Isolation

• Multiple transactions execute independently without interference.

• Changes made by one transaction aren't visible to others until committed.

D - Durability

• Once a transaction is committed, changes are permanent.

• Data will survive system failures/crashes.

RDBMS Doc 4.pdf

<https://drive.google.com/file/d/1cgSZ6wmPmFbx-tC78lt6zC9UBYiDJlyP/view?usp=sharing>

NOSQL Doc 5 pdf

<https://drive.google.com/file/d/1PDqGKE766Z-rG4GVh3ycwJyrraiJV-my/view?usp=sharing>