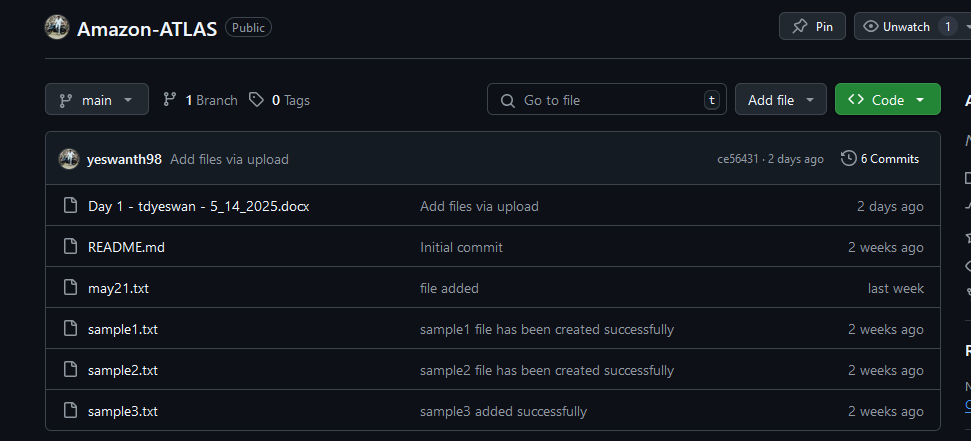
**Day 3 - 21st May 2025**

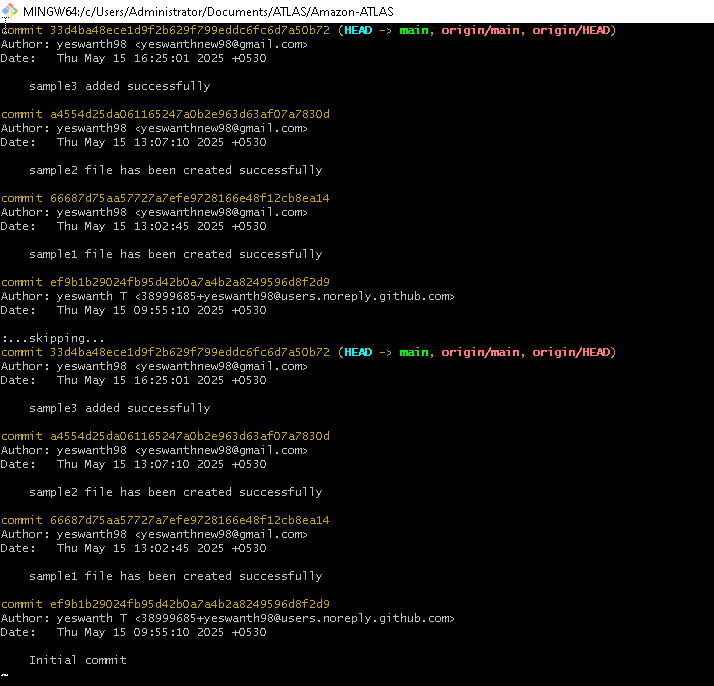
**DevOps / Cloud Fundamentals:**

**Task 1:**

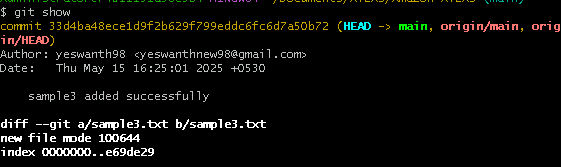
**Create a file names 21st May.txt and push it to your git hub.**



**Git log**



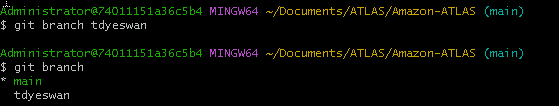
**Git show**



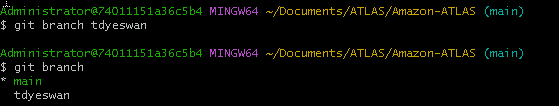
**Git branch - list all the branches git branch**



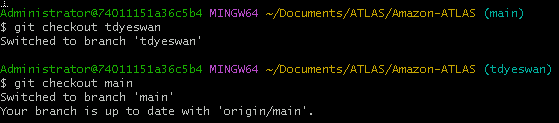
**Git branch PrasunambaBranch - create a new branch**



**Git branch**



**Git checkout targetbranch name**



**Git checkout PrasunambaBranch (this will switch the branch)**

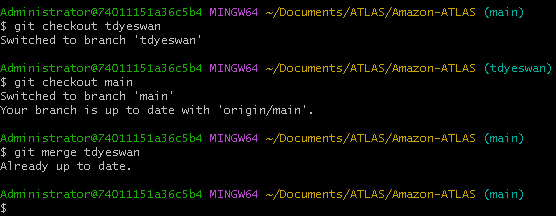
**If i want to merge PrasunambaBranch to master branch**

**Git checkout master**

**(first move to the target branch)**

**Git merge PrasunambaBranch**

**(merger PrasunambaBranch to master branch**

****

**Task 2:**

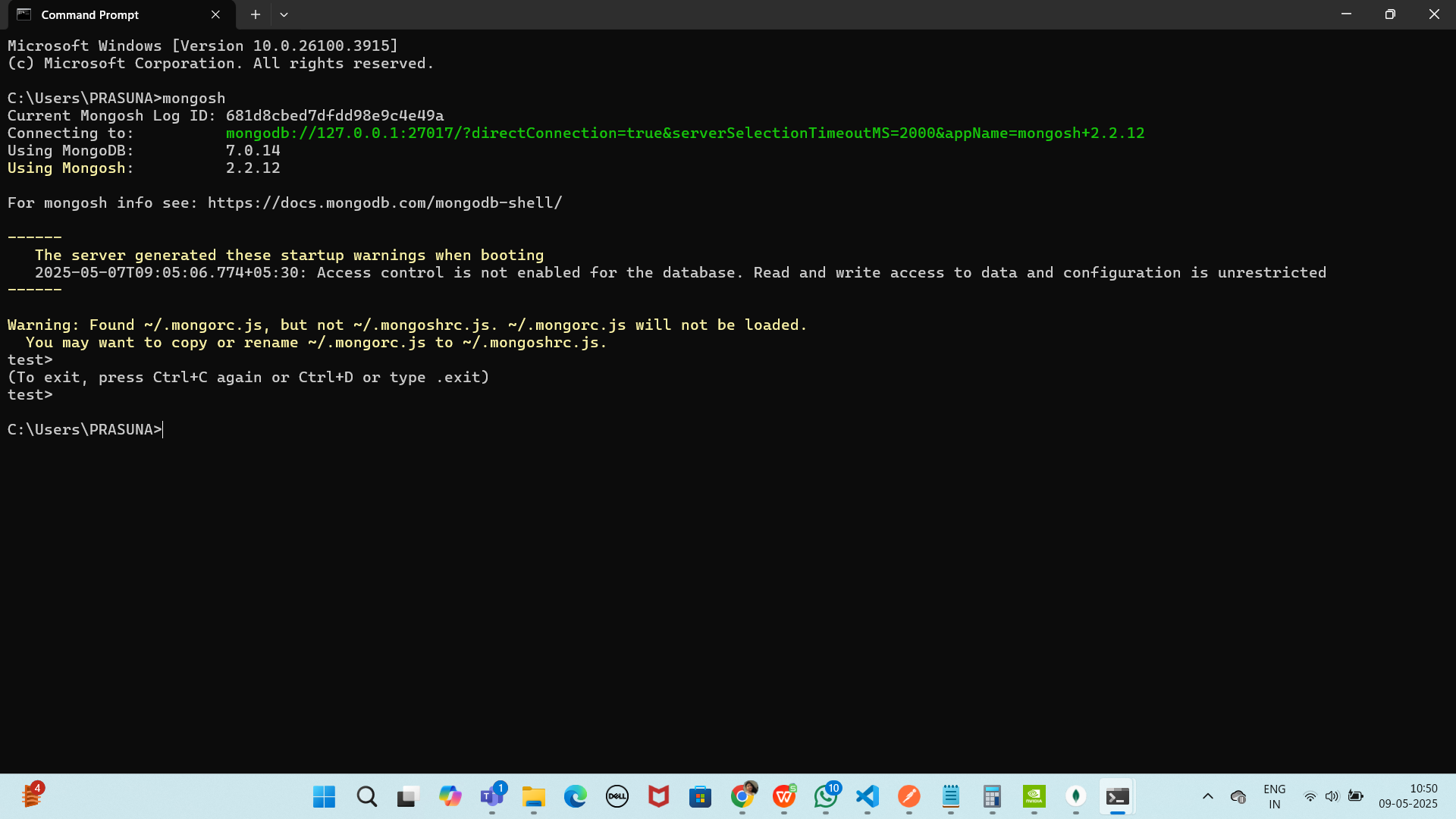
**ACID - definitions**

**Atomicity:** A transaction is treated as a single, indivisible unit. Either all of its operations succeed, or none of them do, ensuring the database remains in a consistent state.

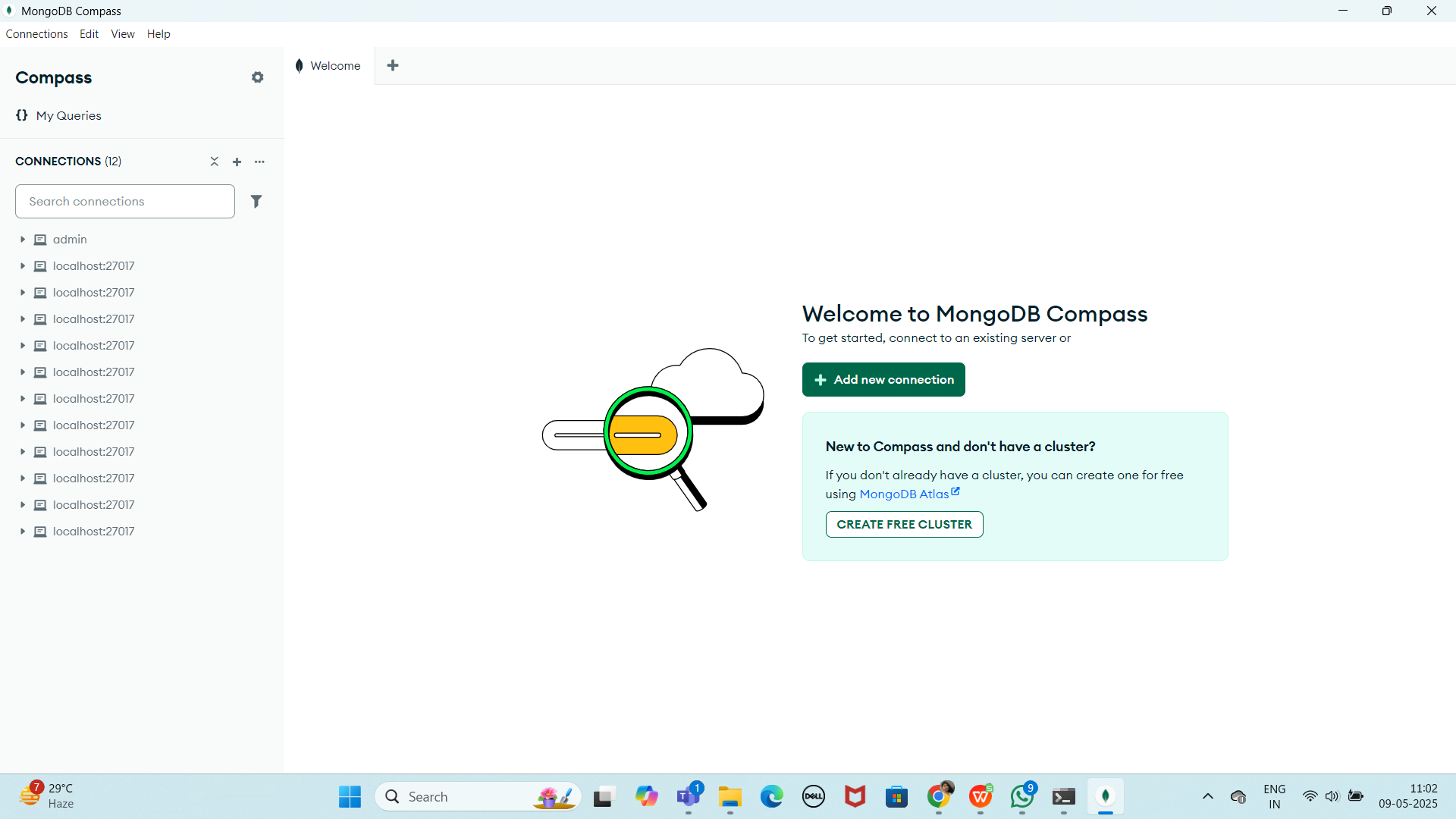
**Consistency**: A transaction must bring the database to a valid state. It must adhere to all rules and constraints defined for the database, ensuring that data integrity is maintained.

**Isolation:** Concurrent transactions do not interfere with each other, preventing data conflicts or inconsistencies.

**Durability:** Once a transaction is committed (successfully completed), its changes are permanently stored and will survive system crashes or failures.



mongo db compass



Make sure 2 cmd s are open

1. Cmd - mongod
2. Cmd - mongosh

Create a database

Use Blog (this is my database)

Remember: In MongoDB, a database is not actually created until it gets content!

Collection using mongosh

db.createCollection(“posts”)

db.posts.insertOne({“title”: “1st Post”})

Blog> db.posts.insertOne({"title":"my 1st post"})

{

acknowledged: true,

insertedId: ObjectId('682da63fae74a79c09c4e49b')

}

Remember: In MongoDB, a collection is not actually created until it gets content!

Inserting documents

insertOne()

Blog> db.posts.insertOne([

{

... title: "Post Title 1",

... body: "Body of post.",

... category: "Sports",

... likes: 1,

... tags: ["news", "Sports"],

... date: Date()

... })

{

acknowledged: true,

insertedId: ObjectId('682da717ae74a79c09c4e49c')

}

Plz Note: If you try to insert documents into a collection that does not exist, MongoDB will create the collection automatically.

... }]

)

Find the data:

Blog> db.posts.find()

[

{ \_id: ObjectId('682da63fae74a79c09c4e49b'), title: 'my 1st post' },

{

\_id: ObjectId('682da717ae74a79c09c4e49c'),

title: 'Post Title 1',

body: 'Body of post.',

category: 'News',

likes: 1,

tags: [ 'news', 'events' ],

date: 'Wed May 21 2025 15:42:39 GMT+0530 (India Standard Time)'

}

]

db.posts.findOne()

Give object to fetch

Querying Data:

Blog> db.posts.findOne('my 1st post')

MongoInvalidArgumentError: Query filter must be a plain object or ObjectId

Blog> db.posts.find({category:"News"})

[

{

\_id: ObjectId('682da717ae74a79c09c4e49c'),

title: 'Post Title 1',

body: 'Body of post.',

category: 'News',

likes: 1,

tags: [ 'news', 'events' ],

date: 'Wed May 21 2025 15:42:39 GMT+0530 (India Standard Time)'

}

]

Projection:

Both find methods accept a second parameter called projection.

This parameter is an object that describes which fields to include in the results.

Note: This parameter is optional. If omitted, all fields will be included in the results.

Blog> db.posts.find({}, {title: 1, date: 1})

[

{ \_id: ObjectId('682da63fae74a79c09c4e49b'), title: 'my 1st post' },

{

\_id: ObjectId('682da717ae74a79c09c4e49c'),

title: 'Post Title 1',

date: 'Wed May 21 2025 15:42:39 GMT+0530 (India Standard Time)'

}

]

Have you noticed that the \_id field is also included. This field is always included unless specifically excluded.

We use a 1 to include a field and 0 to exclude a field.

Blog> db.posts.find({}, {\_id: 0, title: 1, date: 1})

[

{ title: 'my 1st post' },

{

title: 'Post Title 1',

date: 'Wed May 21 2025 15:42:39 GMT+0530 (India Standard Time)'

}

]

Update a Document

Blog> db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 20 } } )

{

acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0

}

Try doing updateMany command also also..

Upsert – will update if found and insert if the doc not found

Blog> db.posts.updateOne(

... { title: "Post Title 3" },

... {

... $set:

... {

... title: "Post Title 3",

... body: "Body of post.",

... category: "Sports",

... likes: 50,

... tags: ["news", "Sports"],

... date: Date()

... }

... },

... { upsert: true }

... )

{

acknowledged: true,

insertedId: ObjectId('682daec91cac9dbc5ac4a079'),

matchedCount: 0,

modifiedCount: 0,

upsertedCount: 1

}

You try updatemany also

TRy delete to delete a doc

db.posts.deleteOne({ title: "Post Title 5" })

Try delete many also

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MongoDB Query Operators

There are many query operators that can be used to compare and reference document fields.

Comparison

The following operators can be used in queries to compare values:

$eq: Values are equal

$ne: Values are not equal

$gt: Value is greater than another value

$gte: Value is greater than or equal to another value

$lt: Value is less than another value

$lte: Value is less than or equal to another value

$in: Value is matched within an array

Logical

The following operators can logically compare multiple queries.

$and: Returns documents where both queries match

$or: Returns documents where either query matches

$nor: Returns documents where both queries fail to match

$not: Returns documents where the query does not match

Evaluation

The following operators assist in evaluating documents.

$regex: Allows the use of regular expressions when evaluating field values

$text: Performs a text search

$where: Uses a JavaScript expression to match documents

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Try the above…

Doc 6 mongodb

<https://docs.google.com/document/d/1tQZiTnnS9hOreFR7DkEwVPyNhDjoNI0WOHUu0RkbQH8/edit?usp=sharing>

Home Task: in GIT

**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>