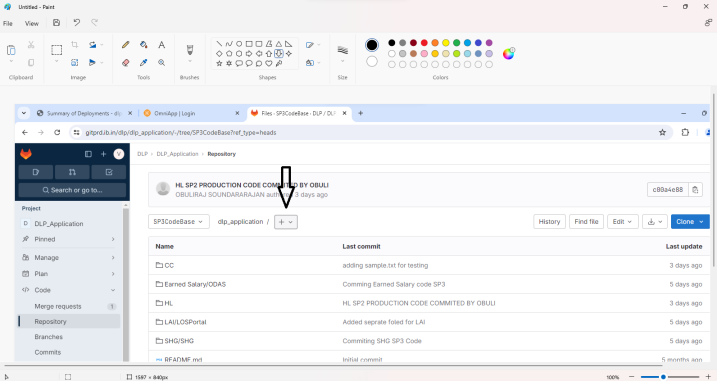
**How to commit code in GIT**

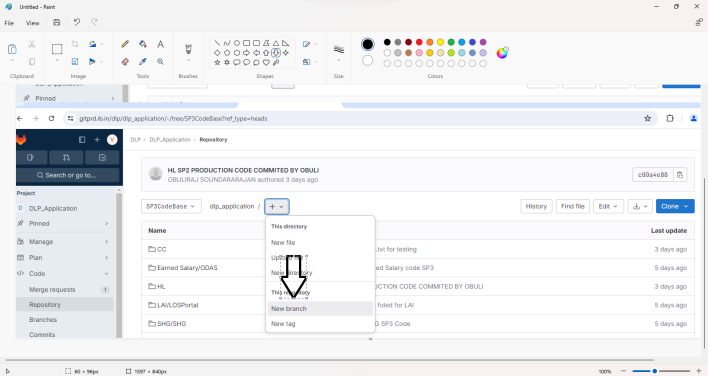
1)Install Git Bash from the path(\\10.141.104.46\share\Vishnu) (Git-2.43.0-64-bit)

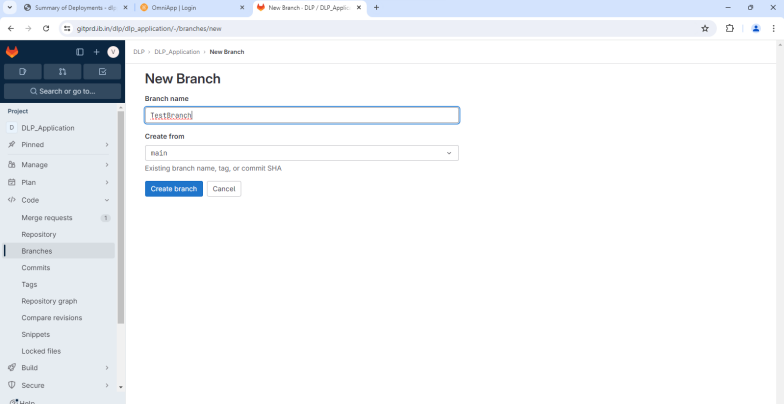
2)Create a Branch in git web app

Login into Git Site:

[https://gitprd.ib.in/dlp/dlp\_application](https://gitprd.ib.in/dlp/dlp_application" \t "_blank" \o "https://gitprd.ib.in/dlp/dlp_application)  
username : dlpng@indianbank.co.in  
password : lck1234@#







3)  
Create a folder for Git in local Folder

Open Git Bash (Right click mouse)

Go to your path in Git Bash  
cd Git Folder/

Follow the steps:

git init

git config --global http.sslVerify false  
git clone -b branch [https://gitprd.ib.in/dlp/dlp\_application.git](https://gitprd.ib.in/dlp/dlp_application.git" \t "_blank" \o "https://gitprd.ib.in/dlp/dlp_application.git)

User name password will be asked   
username : dlpng@indianbank.co.in  
password : lck1234@#

Go to your Repository Path

cd dlp\_application/

Drag the Code inside the git repository(dlp\_application) in local.

4)Do the following

git status  
git add Folder/  
git commit -m "Your Comment" (Must Commit with comments)  
git push

\*To delete a folder or file\*

rm -r Folder/  
git add Folder/  
git commit -m ""    
git push

\*To delete a folder or file\*

\*To pull a particular file\*

git pull origin Branch

\*To pull a particular file\*

For my git hub steps I used

Folder creation in local

Git basg CMD :

git init

git clone https://github.com/poorna-chandu123/Springboot\_Basic\_Topic\_Projects.git

git config --global http.sslVerify true

Next folder creation projects moving

i have github account and i installed git bash in my laptop

my requirement : i have some spring boot projects in my laptop i need to add this project to github how can i add this tell me and this project i need to provide public acess any one can dowload if they need

as of now i know below steps:

step 1: after login github i need to create repostory with public acess

step 2 : need to create folder for git in laptop and i need to map with repository which i created in github right

required folder strucher in git hub

Spring\_boot\_projects

|

junit projects | security projects | JDBC projects ....etc

for junit i need add junit realted projects like that remain folder realted projects i need to add

my doughts : after creating repository i need to create branch for that repositoty or what

and also if i create this junit projects | security projects | JDBC projects ....etc folders in git hub means these folder i need to treate branch or what ?

Got it! I'll guide you step by step with commands. 😊

### ****Step-by-Step Guide to Upload Your Projects to GitHub****

### ****✅ Step 1: Create a GitHub Repository****

Log in to **GitHub**.

Click on **New Repository**.

Enter repository name: **Spring\_boot\_projects**.

Set it as **Public** (so anyone can download).

Do **NOT** initialize with a README (we will create it later).

Click **Create Repository**.

Copy the **HTTPS URL** of the repository (it looks like this: https://github.com/your-username/Spring\_boot\_projects.git).

### ****✅ Step 2: Set Up Git Locally****

#### ****1️⃣ Open Git Bash on your Laptop****

Navigate to the folder where you want to store your Git projects:

cd C:/Users/your-username/Documents

(or wherever you want to keep your projects)

#### ****2️⃣ Clone the Empty Repository****

Run this command to create the **Spring\_boot\_projects** folder on your laptop:

git clone https://github.com/your-username/Spring\_boot\_projects.git

Now, move into the newly created folder:

cd Spring\_boot\_projects

### ****✅ Step 3: Create Subfolders for Projects****

Now, create the subfolders inside Spring\_boot\_projects:

mkdir junit\_projects security\_projects JDBC\_projects

Verify the structure:

ls # (On Windows, use 'dir' instead)

You should see:

junit\_projects security\_projects JDBC\_projects

### ****✅ Step 4: Add Your Existing Spring Boot Projects****

Move your projects into the correct folders. Example:

mv /path/to/your/junit\_project1 ./junit\_projects/

mv /path/to/your/security\_project1 ./security\_projects/

mv /path/to/your/jdbc\_project1 ./JDBC\_projects/

Now check:

ls junit\_projects

ls security\_projects

ls JDBC\_projects

### ****✅ Step 5: Initialize Git & Push to GitHub****

#### ****1️⃣ Add all files to Git****

git add .

#### ****2️⃣ Commit the files****

git commit -m "Added initial Spring Boot projects"

#### ****3️⃣ Push to GitHub****

git push origin main

Now, go to GitHub and refresh the page. You should see all your projects inside their respective folders. 🎉

### ****🤔 Should You Use Branches Instead of Folders?****

**Folders** ✅ → Best for organizing multiple projects in one repository (what you're doing).

**Branches** ❌ → Not recommended for your case because branches are used for different versions of the same project, not for different projects.

### ****✅ Final Step: Share Your Repository****

Since it's public, you can share the repository link with anyone:

https://github.com/your-username/Spring\_boot\_projects

Anyone can download it using:

git clone https://github.com/your-username/Spring\_boot\_projects.git

### ****🎯 Summary of Git Commands You Used****

git clone <repo-url>

mkdir <folder-name>

mv <source> <destination>

git add .

git commit -m "Your message"

git push origin main

Let me know if you have any doubts! 🚀

wait ur telling no need branches for my requirement Now ok but usually how we will use branches

for what purpose we will use branches

tell me in real time projets how we will use i need complete information

I understand below thing we will create repository by project name ex: DLP project

in this repository we can create multiple Branches right

senario in my case : my project DLP project

in this project diffrent teams diffrent Application devloping Ok ....

and every team they created own branchs for there Application Ok ...

in there respective branches they are are pushhing code Ok ...

Branches ❌ → Not recommended for your case because branches are used for different versions of the same project, not for different projects.

As u mentioned above so evry time they push the code with same branch then how the things will happend i am getting confustion kindly clreare

Great question! Let me explain branches in **real-time projects** step by step. 🚀

## ****1️⃣ What Are Git Branches?****

A **branch** in Git is like a separate workspace where developers can work on features, fixes, or experiments **without affecting the main project**.

The default branch is usually called main (or master in older projects).

Teams create **separate branches** to develop features, fix bugs, or work on different parts of the project.

## ****2️⃣ How Are Branches Used in Real-Time Projects?****

### ****Scenario: DLP Project****

Let's say your company is developing a **DLP (Data Loss Prevention) application**.

Different teams are working on different **applications (modules)** inside this project.

✅ **Main Repository Name:** DLP\_Project  
✅ **Teams & Their Modules:**

| **Team Name** | **Working On** | **Branch Name** |
| --- | --- | --- |
| Team A | Authentication Module | auth-feature |
| Team B | Payment Module | payment-feature |
| Team C | Dashboard UI | dashboard-ui |

Each team **creates its own branch** and works independently.

## ****3️⃣ How Do Teams Use Branches?****

### ****Step 1: Developers Create a New Branch****

When a developer starts working on a new feature, they **create a new branch**:

git checkout -b auth-feature

Now they are working on the auth-feature branch.

### ****Step 2: Developers Work and Push Code****

They add files and commit their code:

git add .

git commit -m "Added login functionality"

git push origin auth-feature

Now, the branch auth-feature exists on GitHub.

### ****Step 3: Code Review & Merging the Branch****

Once the feature is **complete**, the developer creates a **Pull Request (PR)** from auth-feature → main branch.  
A senior developer reviews the code, and if everything is fine, they merge the branch into main.

git checkout main

git pull origin main

git merge auth-feature

git push origin main

After merging, the feature is now part of the main project.

## ****4️⃣ What Happens if Two Teams Work on the Same File?****

Sometimes, multiple teams might work on the **same file**, leading to **merge conflicts**.  
For example, if both auth-feature and payment-feature modify config.yaml, Git will **ask you to resolve conflicts manually** before merging.

Steps to resolve conflicts:

Git will show **conflicted files**.

Open the file and **manually edit** to keep the correct changes.

Mark it as resolved:

git add <file>

git commit -m "Resolved conflict in config.yaml"

Push the changes.

## ****5️⃣ Branching Strategies in Real-Time Projects****

Companies use different **branching models**. The two most common ones are:

### ****🔹 A. Git Flow (Used in Large Teams)****

main → Stable production-ready branch.

develop → All new features are merged here before moving to main.

feature/\* → New features are developed in separate branches.

hotfix/\* → Urgent bug fixes for production.

release/\* → Final testing before release.

✅ **Example Workflow:**

A developer creates a new feature branch:

git checkout -b feature/login-page develop

Works on code and pushes it:

git commit -m "Added login page"

git push origin feature/login-page

When complete, merges into develop:

git checkout develop

git merge feature/login-page

When a new release is ready, develop is merged into main:

git checkout main

git merge develop

git push origin main

### ****🔹 B. Trunk-Based Development (Used in Startups)****

**No** develop **branch** – everything is merged directly into main.

Developers use **feature branches**, but merge them quickly (within a day or two).

Works well for **small, fast-moving teams**.

✅ **Example Workflow:**

A developer creates a branch for a small feature:

git checkout -b small-fix

Works on it and merges **directly into** main:

git checkout main

git merge small-fix

git push origin main

## ****6️⃣ Summary: When to Use Branches?****

| **Use Case** | **Branch Strategy** |
| --- | --- |
| New Feature Development | Create a **feature branch**, then merge into main |
| Bug Fixes | Create a **hotfix branch**, then merge into main |
| Multiple Teams Working on One Project | Each team has its own **feature branch** |
| Testing Before Release | Use a **release branch** |
| Urgent Fixes for Live Application | Use a **hotfix branch** |

## ****🎯 Answering Your Doubt****

"Every time they push the code to the same branch, how do things work?"

**Each team works on their own branch** (auth-feature, payment-feature, etc.).

**When they complete their work, they merge it into** main through a **Pull Request**.

**Merge conflicts** can happen if two teams modify the same file – Git will ask to resolve them.

Now, do you have any doubts? Or do you need a hands-on example? 😊🚀

No i didnt understand

see below u told

Team Name Working On Branch Name

Team A Authentication Module auth-feature

Team B Payment Module payment-feature

Team C Dashboard UI dashboard-ui

Step 1: Developers Create a New Branch

When a developer starts working on a new feature, they create a new branch:

git checkout -b auth-feature

Now they are working on the auth-feature branch.

they create a new branch: that means in auth-feature branch in side again new branch will create or what and main mean first auth-feature branch is main a? and fetures devloped perpouse created onces verstion branches with same branch name right

doubt if same branch neame it will overrite na

Good question! Let me explain in a **clear and simple way** with a proper **structure**. 🚀

## ****🔹 How Git Branches Work in a Real Project****

Branches in Git are like **different workspaces** where developers can work on features **without affecting each other**.

### ****✅ Step 1: The Main Repository (****main ****Branch)****

When you create a GitHub repository, by default, it has a **main branch** (previously called master).

git clone https://github.com/company/DLP\_Project.git

cd DLP\_Project

git branch # Shows "main" as the default branch

This main branch is the **stable version** of the project.

Developers **do NOT work directly on** main – they create **new branches** for development.

### ****✅ Step 2: Teams Create Their Own Branches****

Now, let's say **Team A** is working on the **Authentication Module**.

📌 **They create a new branch from** main using:

git checkout -b auth-feature # Creates and switches to "auth-feature" branch

Now, the developer is in the auth-feature branch.

This branch is **separate** from main, so any changes here will **not affect** main until merged.

Similarly, **Team B** creates its own branch:

git checkout -b payment-feature

Now there are **three branches** in the repository:

git branch

# Output:

# main

# auth-feature

# payment-feature

💡 **Branches are independent from each other!**

### ****✅ Step 3: Developers Work & Push Their Changes****

Now, Team A works inside their auth-feature branch and adds new code:

git add .

git commit -m "Added login page"

git push origin auth-feature

This pushes auth-feature to GitHub **without affecting** main.

Meanwhile, Team B is also working and pushing changes to payment-feature:

git push origin payment-feature

**Both teams are working independently.** ✅

### ****✅ Step 4: Merging Branches to**** main

Once Team A finishes the Authentication Module, they want to **merge their code into** main.

**Create a Pull Request (PR) in GitHub** (to review the code before merging).

Once approved, merge auth-feature into main:

git checkout main

git pull origin main # Get the latest changes

git merge auth-feature # Merge auth-feature into main

git push origin main # Push updated main branch

After merging, auth-feature **can be deleted** (optional, but recommended):

git branch -d auth-feature

git push origin --delete auth-feature

## ****🔹 Answering Your Doubt****

"If I create a new branch, does it get created inside another branch?"

🔹 **No! A new branch is not created inside another branch.**

Every branch starts from an **existing branch**.

If you run git checkout -b auth-feature, it means:  
✅ "Create a new branch called auth-feature **from the current branch (main)**."

"If multiple developers work on the same branch, will it overwrite?"

🔹 **No, Git does not overwrite automatically!**

When two developers work on the same branch, they need to **pull the latest changes** before pushing:

git pull origin auth-feature

If two developers edit the **same file**, Git will show a **merge conflict**, and they must manually fix it.

"So every time we need a new branch for a new feature?"

🔹 **Yes!**

For every new feature or fix, developers create a **new branch**.

Once merged into main, the branch can be **deleted** to keep the repository clean.

## ****🎯 Summary****

✅ main → The stable version of the project.  
✅ Developers create **feature branches** (auth-feature, payment-feature).  
✅ Each branch is **independent** and **does not overwrite** other branches.  
✅ After merging into main, the feature branch is **deleted**.

Now, is it clear?

here main means DLP\_Project right and it is repository na not branch right in DLP\_Project repository under we are creating branches right

Yes, you're almost correct! Let me clarify it further:

### ****✅ Main Repository & Branches****

**DLP\_Project** → This is the **repository** on GitHub (your project's container).

Inside **DLP\_Project**, there are different **branches** (like main, auth-feature, payment-feature).

**The** main **branch is just one branch inside the repository** (it is not the repository itself).

### ****📌 Understanding the Structure****

Think of your GitHub repository like this:

DLP\_Project (Repository)

│

├── main (Branch - The stable version)

│

├── auth-feature (Branch - Team A working on Authentication)

│

├── payment-feature (Branch - Team B working on Payments)

│

└── dashboard-ui (Branch - Team C working on UI)

### ****✅ So, to clarify:****

**DLP\_Project** is a **repository** (not a branch).

main **is a branch inside DLP\_Project** (like other branches).

Developers create **feature branches** inside the repository.

### ****🔹 Answering Your Doubt****

"Here main means DLP\_Project, right?"  
❌ **No** → main is just a \*\*branch inside DLP\_Project. ✅ DLP\_Project` is the **repository**.

"DLP\_Project is a repository, not a branch, right?"  
✅ **Yes!** The repository is **DLP\_Project**. It contains multiple branches.

"In DLP\_Project repository, we are creating branches, right?"  
✅ **Exactly!** All branches exist **inside** the DLP\_Project repository.

### ****📌 Commands Recap****

1️⃣ **Clone the repository (DLP\_Project) from GitHub**

git clone https://github.com/company/DLP\_Project.git

cd DLP\_Project # Move into the repo

2️⃣ **Check which branch you're on (default is** main**)**

git branch

3️⃣ **Create a new branch for a feature (e.g.,** auth-feature**)**

git checkout -b auth-feature

4️⃣ **Push the new branch to GitHub**

git push origin auth-feature

### ****🎯 Final Summary****

✅ **DLP\_Project** is the **repository** (not a branch).  
✅ main **is just a branch inside** DLP\_Project (the stable version).  
✅ **Other branches** (auth-feature, payment-feature) are created inside the repository.

Now, is everything **100% clear**? 😊🚀

Yes! You’re almost correct, but let me **fine-tune** your understanding a bit. 😊

### ****✅ Is**** main ****a Tree of All Branches?****

🔹 **No,** main **is not automatically a tree of all branches.**  
🔹 main only contains code that has been **merged into it** from other branches.  
🔹 If a branch (auth-feature, payment-feature) is **not merged** into main, its code will **not** be available in main.

### ****✅ When Does Code from Other Branches Appear in**** main****?****

✅ Code from different branches (auth-feature, payment-feature) will be **available in** main only **after merging**.

📌 **Example Scenario:**  
Let's say this is the current state of your repository (DLP\_Project):

DLP\_Project (Repository)

│

├── main (Branch - The stable version)

│ ├── app.java ✅

│ ├── config.yaml ✅

│

├── auth-feature (Branch - Authentication team)

│ ├── login.java 🚀 (New feature, NOT in `main` yet)

│

└── payment-feature (Branch - Payment team)

├── payment.java 🚀 (New feature, NOT in `main` yet)

### ****🔹 Before Merging:****

The main branch **does NOT have** login.java **or** payment.java **yet.**

These files exist **only inside their respective branches**.

### ****🔹 After Merging:****

Once auth-feature and payment-feature are merged into main, the updated structure looks like this:

DLP\_Project (Repository)

│

├── main (Branch - Now includes merged features)

│ ├── app.java ✅

│ ├── config.yaml ✅

│ ├── login.java ✅ (Merged from `auth-feature`)

│ ├── payment.java ✅ (Merged from `payment-feature`)

│

Now, main contains **all the latest features**. 🎉

### ****📌 Key Takeaways****

✅ main **does NOT automatically contain** all branch code.  
✅ Code from branches **must be merged** into main.  
✅ After merging, the new features **become part of** main.

💡 **Final Answer:** **Yes,** main **eventually contains all the branch-related code, but only after merging.**

Now, is this **fully clear**? 😊🚀