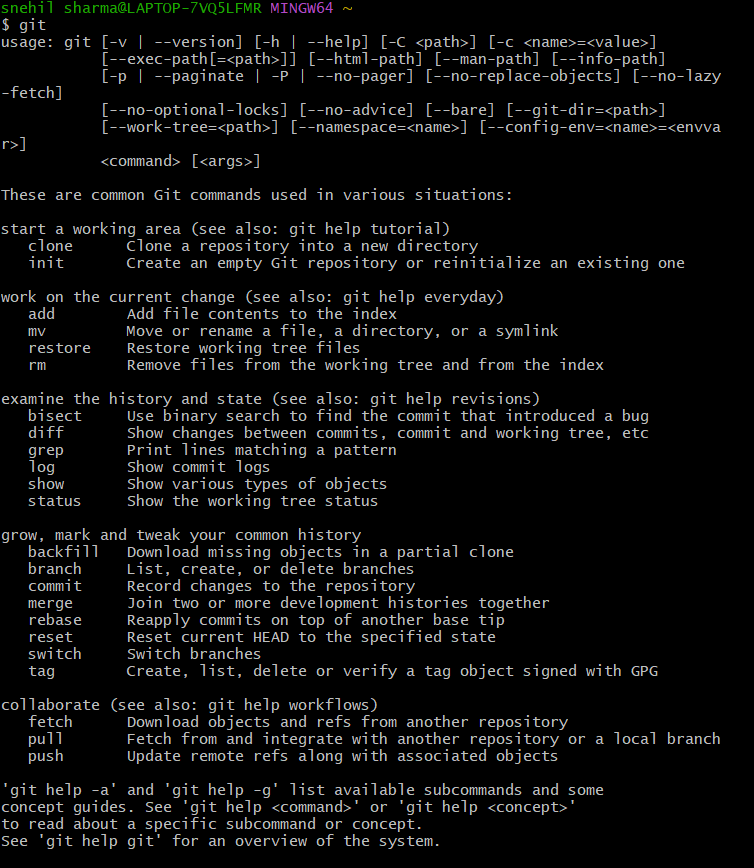
\*-> imporetent

1. git -> confirm installation of git
2. Ls -> list of files and folders in the terminal



Cd -> change the current directory

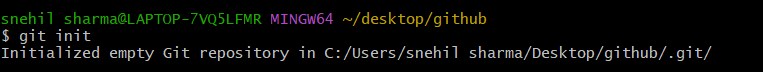


Mkdir-> make directory inside current directory



\*Git init ->

git init is the command used to **initialize a new Git repository** in your project folder. It tells Git to start tracking versions of your files in that folder.

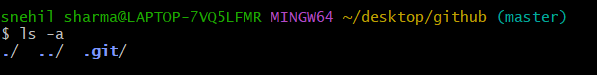


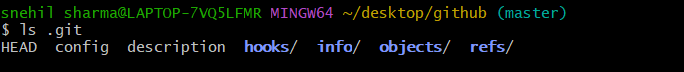
**🔹 git init kya karta hai?**

Jab aap kisi folder ke andar git init likhkar enter karte ho, to:

* Ek **.git** naam ka **chhupa hua folder (hidden folder)** create hota hai.
* Ye folder **Git ka dimaag** hota hai – isme Git sab kuch store karta hai:  
  commits, branches, logs, etc.

Ls -a -> Isse aapko .git folder dikh jayega. List of all folders including hidden folders also



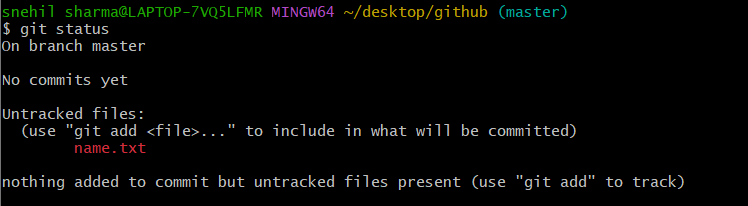


\*Touch -> create a file in current directory



\*Git status-> **🔹 Definition (As per your screenshot):**

**git status** ek Git command hai jo batata hai ki **kaunse files Git ke dwara track nahi ho rahi hain**, **kaunse changes stage par hain**, aur **ab tak koi commit hua hai ya nahi**.



1. **On branch master**  
   👉 Aap **master branch** par ho (default Git branch).
2. **No commits yet**  
   👉 Aapne ab tak koi bhi git commit nahi kiya hai. Yani Git mein abhi koi snapshot nahi banaya gaya.
3. **Untracked files:**  
   👉 Git ko ek file mili hai: name.txt  
   Lekin Git us file ko abhi **track nahi** kar raha.
4. **nothing added to commit...**  
   👉 Aapne abhi koi file git add se stage nahi ki hai.  
   **Suggestion:** Agar aap chahein to git add name.txt se us file ko track kar sakte ho.

\*git add ->

**🔹 git add – Simple Explanation in Hindi**

**git add** Git ki ek command hai jo kisi file ko **staging area** mein bhejti hai, taaki usse **commit** kiya ja sake.

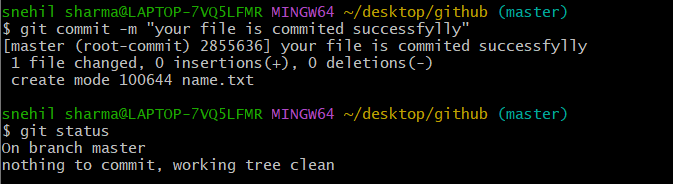


\*git commit ->

**🔹 Git me commit ka matlab**

**commit** ka matlab hota hai:  
🔒 **“Git ko officially batana ki meri file/changes ko permanently save kar lo.”**

**Git commit ek snapshot hota hai** — jaise photo khinch lena aapke project ke current state ka.  
Jab aap git commit karte ho, to Git **aapke staged changes ko history me save** kar leta hai.



Vim:

vim is a powerful text editor that you can use directly in your terminal. Here's how to use it:

vim filename.txt

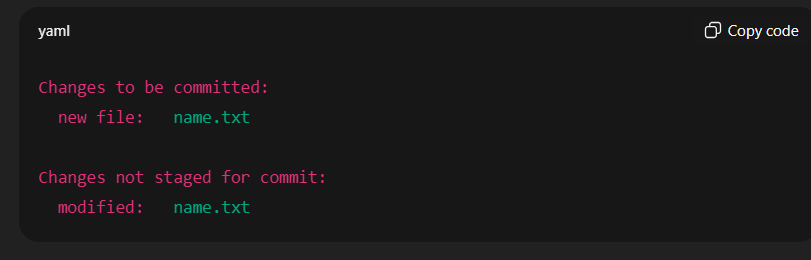
If the file doesn't exist, Vim will create it.

Cat :

File kai content ko dekhti hai

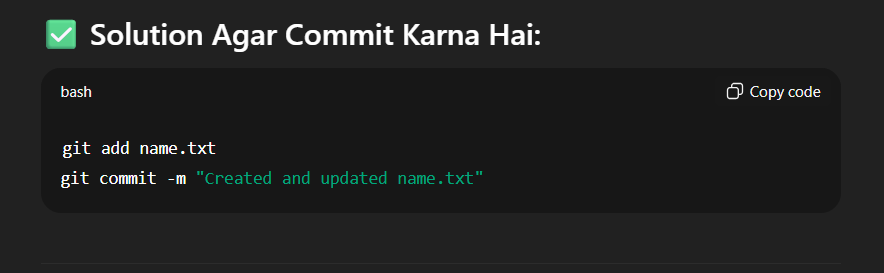
NOTE:

Agar mai koi file banata hu or use add kar deta hu or fir usme kuch changes karta hu to git status karne par ye hoga



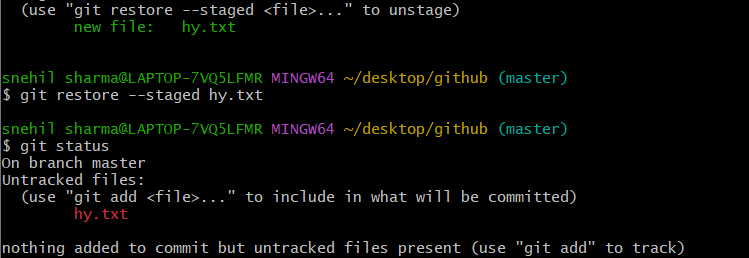
**🔍 Simple Explanation:**

* **Green (new file)**: Tumne file banayi aur git add se stage kar diya.
* **Red (modified)**: Tumne file me fir se kuch changes kiye, **lekin un naye changes ko add nahi kiya**, isliye wo **staging area me nahi gaye**.



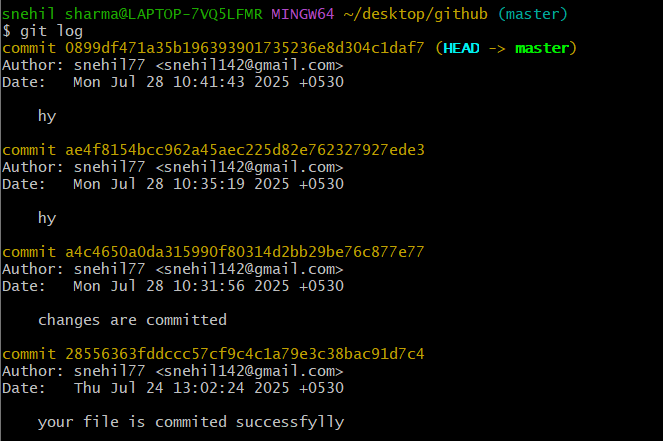
To unstage staged file we’ll use the following command:

Git restore --staged file name



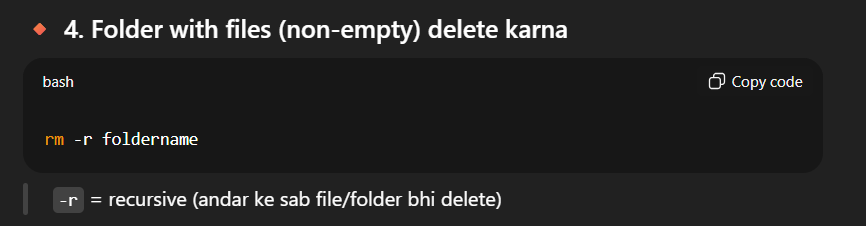
Git log:

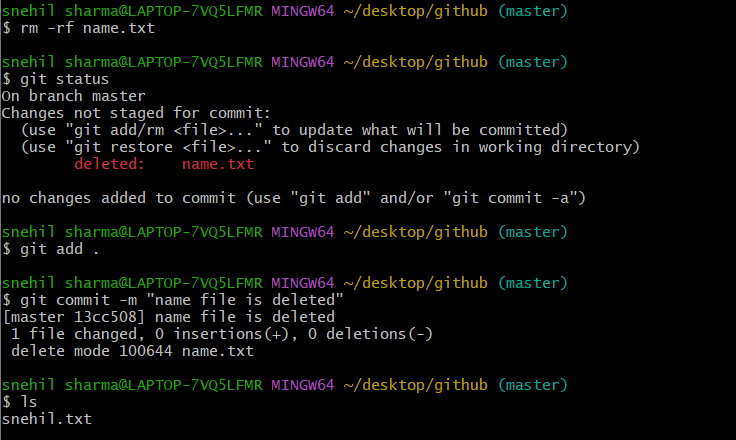
git log ek **very important command** hai jo tumhe Git me huye **commits ka pura history** dikhata hai.



Rm -r

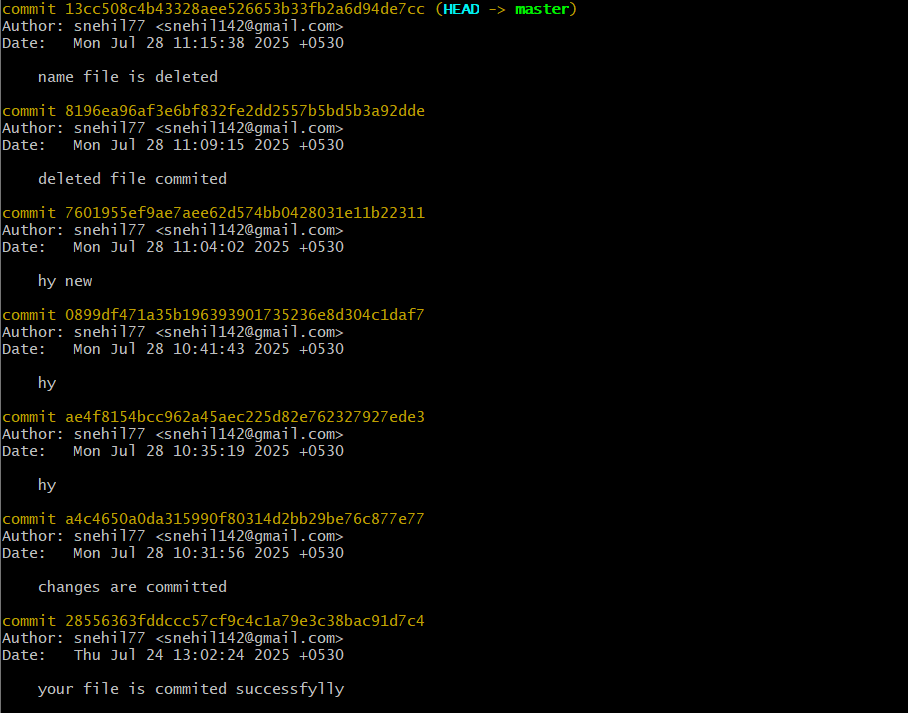
Rm -> remove file or folder



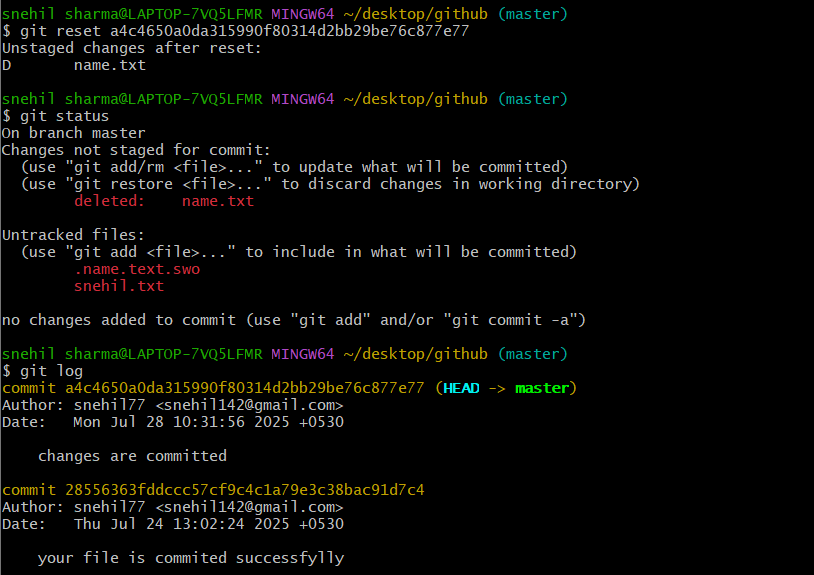


Remove commit from commit history

Hum kisi bhi particular commit ko nahi hata sakte



Changes are commited ko hi agar mujhe rakhna hai or upar wale sare delete karne hai to me rser karunga iski id se

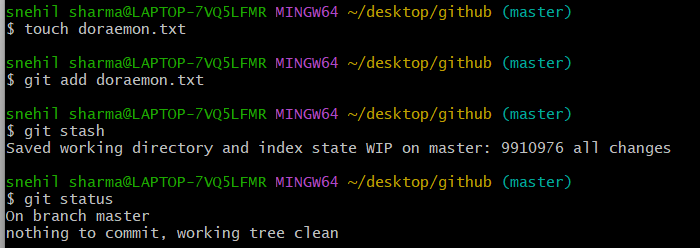


Git stash:

git stash tumhare **working directory ke changes ko temporarily save** karta hai, taki tum clean working state me kuch aur kar sako (jaise branch switch).

 Sab **uncommitted changes** save ho jaate hain

 Working directory clean ho jaata hai



Git stash pop:

git stash pop **tumhare latest stash ko wapas working directory me laata hai**  
**Aur** stash list me se **delete** bhi kar deta hai.



**git stash clear — Simple Explanation**

**🧾 Command:**

bash

Copy code

git stash clear

**🧨 Kya karta hai?**

Git ke andar jitne bhi **stash save kiye gaye hain**, **sabko permanently delete** kar deta hai.

**GitHub repository** aur **Git repository** related hote hain, lekin **dono alag cheezein** hoti hain. Chalo simple language me difference samjhte hain:

**🔁 Git Repository (Local)**

* **Kahan hoti hai:** Aapke **computer ke andar**.
* **Kaise banti hai:** Jab aap git init chalate ho.
* **Kya hota hai:** Ye ek folder hota hai jisme .git hidden folder ban jata hai, jisme:
  + Code ka pura history
  + Branches, commits, changes ka record hota hai.

✅ Ye **offline** hoti hai — sirf aapke system pe.

**☁️ GitHub Repository (Remote)**

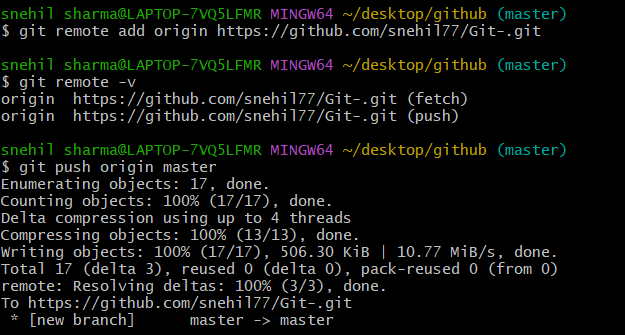
* **Kahan hoti hai:** **GitHub website pe** (ya koi aur service: GitLab, Bitbucket).
* **Kaise banti hai:** GitHub pe "New Repository" button se.
* **Kya hota hai:** Ye ek **online copy** hoti hai aapke project ki, jisko:
  + Aap git push karke update karte ho.
  + Duniya ke kisi kone se access kar sakte ho.
  + Team ke log bhi collaborate kar sakte hain.

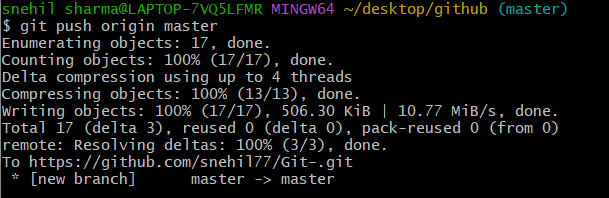
✅ Ye **online** hoti hai — ek remote backup aur sharing tool.

| **Step** | **Description** |
| --- | --- |
| 1️⃣ | Aap apne system pe git init se ek **Git repository** banate ho |
| 2️⃣ | GitHub pe ek **GitHub repository** create karte ho |
| 3️⃣ | Fir git remote add origin <URL> se dono ko jod dete ho |
| 4️⃣ | git push se local se remote (GitHub) pe code bhejte ho |

git remote add origin https://github.com/snehil77/Git-.git

| **Part** | **Meaning** |
| --- | --- |
| git | Git command-line tool ka use kar rahe ho |
| remote | Remote repository (jaise GitHub) se related command |
| add | Naya remote repository link add karna |
| origin | Remote ka **naam** (shortcut name). Yeh commonly "origin" hi hota hai |
| https://github.com/snehil77/Git-.git | Remote repository ka actual **URL** (GitHub par jo repo banayi hai uska link) |





| **Part** | **Meaning** |
| --- | --- |
| git | Git command tool |
| push | Code ko **remote repo me bhejna** |
| origin | Remote repo ka naam (by default origin hota hai) |
| master | Branch ka naam (jise aap push kar rahe ho) |

**📚 Concept: Git Branching and Merging (with visual demo)**

**Git mai commit karne par branch ka structure banta hai jisme master or main branch by default main hoti hai or ye publically accessible hoti hai isiliye jab bhi hum koi naya feature bana rahe hote hai to alag branch banate hai baad mai us branch ko master branch kai saath merge kar dete hai**

**🔹 Initial State:**

* Ek linear commit history hai:  
  C0 → C1 → C2 → C3 → C4

**✅ Step-by-Step Explanation:**

**🟢 Step 1: git branch feature**

* Ye command ek **nayi branch** banati hai jiska naam hai feature
* feature branch ban jaati hai **C4 par base le kar**

cpp

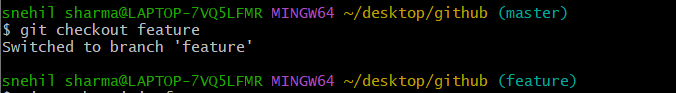
Copy code

\* C4

|

|--> (new branch: feature)

**🟢 Step 2: git checkout feature**

****

* Ab aap feature branch pe switch kar jaate ho
* Uske baad commit commands diye gaye — jisse feature branch pe naye commits ban jaate hain:

nginx

Copy code

C4 → C5 → C6 (on feature branch)

**🟢 Step 3: git checkout main**

* Ab aap wapas **main** branch pe aate ho (jo ab bhi C4 pe hi hai)

**🟢 Step 4: git commit (on main)**

* Ab main branch pe ek alag commit hota hai → C7  
  Iska matlab ye hua ki main aur feature dono branches **alag direction me grow** ho rahe hain:

css

Copy code

→ C5 → C6 (feature)

/

C4 →

\

→ C7 (main)

**🔁 Ab Problem Kya Hai?**

* Dono branches (feature aur main) ne C4 ke baad apna-apna kaam kiya hai.
* Agar aapko **feature branch ka kaam main me lana hai**, to aapko **merge** karna hoga.

**🟢 Step 5: git merge feature**

* Jab aap main branch pe hote ho aur ye command chalate ho:

bash

Copy code

git merge feature

* Git dekhta hai ki feature aur main dono ne C4 ke baad alag changes kiye hain — to Git dono ka **merge commit** banata hai.

👉 Ye **merge commit** dono branches ko ek jod deta hai — jisse final structure ban jaata hai:

java

Copy code

→ C5 → C6 (feature)

/ \

C4 → C7 ------------------> Merge Commit (C8)

**✅ Final Outcome:**

* main branch pe ab feature branch ke changes bhi aa gaye.
* Code safe bhi raha, aur sab kuch history ke saath traceable hai.

**🧠 Key Concepts Samjhe Is Video Se:**

| **Concept** | **Meaning** |
| --- | --- |
| git branch feature | Nayi branch banana |
| git checkout feature | Us branch pe switch karna |
| git commit | New changes add karna |
| git merge feature | Do branches ko mila dena |
| Merge Commit | Jab dono branches me changes ho to Git ek extra commit banata hai to merge them |

**🔀 What is a Fork?**

**Fork** ek GitHub feature hai jisse aap **kisi aur ke GitHub project ka full copy** apne GitHub account me bana lete ho — taaki aap **independent changes** kar sako bina original project ko touch kiye.

**📦 Example se Samjho:**

Socho kisi ne GitHub pe ek project banaya:

arduino

Copy code

https://github.com/elonmusk/spacex-project

Ab aap usme changes karna chahte ho — par directly allowed nahi hai.

Toh aap **“Fork”** button press karte ho 👇  
➡ GitHub us project ka **poora copy** aapke account me bana deta hai:

bash

Copy code

https://github.com/snehil77/spacex-project

**🧠 Fork vs Clone:**

| **Feature** | **Fork** | **Clone** |
| --- | --- | --- |
| Kya karta hai | GitHub repo ka **copy banata hai** | Project ko **local system pe lata hai** |
| Kahan hota hai | GitHub account me | Aapke computer me |
| Use kab | Jab aap dusre ke repo ko modify karna chahte ho without touching original | Jab aap code par kaam karna chahte ho |
| Remote link | Forked repo ka link | Original ya forked dono ho sakta hai |

**✅ When to Use Fork?**

* Aap open-source project me contribute karna chahte ho.
* Aap kisi aur ke project ko customize karna chahte ho.
* Aap original repo me access nahi rakhte.

Bahut badhiya question, Snehil!  
Tumne ab tak origin suna hoga — ab **upstream** bhi samajh lo easily. 👇

**🔗 What is Upstream URL in Git?**

**upstream ek Git remote hota hai jo refer karta hai original GitHub repo ko jisko tumne fork kiya tha.**  
Isse tum original repo se latest updates le sakte ho.

**🔁 Fork Wale Scenario se Samjho:**

**💡 Suppose:**

* **Original repo:** https://github.com/elonmusk/spacex-project
* **Tumhara forked repo:** https://github.com/snehil77/spacex-project

**✅ Ab tumhare paas 2 remote URLs hote hain:**

| **Remote Name** | **URL** | **Role** |
| --- | --- | --- |
| origin | https://github.com/snehil77/spacex-project | ➤ Tumhara personal copy (forked repo) |
| upstream | https://github.com/elonmusk/spacex-project | ➤ Original repo jahan se tumne fork kiya |

**⚙️ Upstream ka Use Kab Hota Hai?**

Jab original repo me naye commits aaye ho aur tum **apne fork ko update** karna chahte ho.

**🔨 How to Set Up Upstream URL:**

bash

Copy code

git remote add upstream https://github.com/elonmusk/spacex-project

Check karne ke liye:

bash

Copy code

git remote -v

Output kuch aisa dikhega:

bash

Copy code

origin https://github.com/snehil77/spacex-project (fetch)

origin https://github.com/snehil77/spacex-project (push)

upstream https://github.com/elonmusk/spacex-project (fetch)

upstream https://github.com/elonmusk/spacex-project (push)

**🔄 Upstream se Changes Pull Karna:**

bash

Copy code

git fetch upstream

git checkout main

git merge upstream/main

**🎯 Summary:**

| **Term** | **Meaning** |
| --- | --- |
| origin | Tumhara remote forked repo |
| upstream | Original repo jisko tumne fork kiya |
| Use | Upstream se latest changes laane ke liyse |

Chaho toh mai tumhare liye **fork → upstream set → sync** ka full flow likh ke de sakta ho real GitHub project ke liye. Bolo toh?

Ask ChatGPT

**🔁 Typical Fork Flow:**

1. Kisi repo ko fork karo (GitHub button se).
2. Apne forked repo ko clone karo:

bash

Copy code

git clone https://github.com/snehil77/their-project.git

1. Changes karo, commit karo.
2. GitHub pe push karo.
3. Fir **Pull Request** bhejo original repo me (taaki wo aapke changes accept kare).