



Why do we create SSH and connect it with GitHub?

Short answer:

To prove to GitHub that this laptop is **YOU** — safely and permanently.



The real problem SSH solves

Before SSH, GitHub needs to answer **one question**:



“Who is trying to push code to this repository?”

GitHub **must verify your identity** before allowing:

- `git push`
 - `git pull`
 - `git clone`
-



Old way (NOT allowed anymore)

Username + Password



GitHub disabled this because it's **not secure**.



Token way (Works but annoying)

Username + Personal Access Token


Problems:


- Token can **expire**
 - Token can be **leaked**
 - You have to **re-login**
 - Windows often saves **wrong tokens**
 - Causes 403 errors (like you faced)
-



SSH way (BEST & PROFESSIONAL)

SSH uses:


-  **Private key** → stays on your laptop

-  **Public key** → stored on GitHub

Simple analogy:





SSH is like a **house key**

- You keep the **original key** (private key)
- GitHub has a **lock that matches it** (public key)

If the key matches → access allowed 




What happens during `git push` (behind the scenes)

1. Git says:
 “I am `thikekarshweta` pushing code”
2. GitHub says:
 “Prove it”
3. Your laptop shows the **private SSH key**
4. GitHub matches it with the **public key**
5.  Match → push allowed
6.  No match → Permission denied (`publickey`)



Why SSH is more secure

Reason	Why it matters
No password sent	Hack-proof
Key never leaves laptop	Safe
Encrypted connection	Secure
No expiry	Forever
No re-login	Peace 



Why ALL developers use SSH

- Google
- Amazon
- Netflix
- Kubernetes contributors

 **SSH is the industry standard**

Why we did SSH for YOU specifically

Because:

- You're learning **Kubernetes** (daily pushes)
 - You don't want **auth issues again**
 - You want **clean, professional setup**
 - You're building a **long-term repo**
-

One sentence you should remember forever

SSH tells GitHub: “This laptop belongs to Shweta.”

After SSH setup, you can:

- `git clone`
- `git pull`
- `git push`

👉 Without login prompts

👉 Without tokens

👉 Without errors

What is SSH (in very simple words)

SSH = Secure way to prove to GitHub that your laptop is YOU

Instead of typing password/token again and again,
you give GitHub a **key** once.

Real-life analogy (VERY IMPORTANT)

Imagine GitHub is your **office building** 🏢
Your laptop is **you**.

❌ Password way (old)

- Anyone who knows the password can enter
- Not safe ❌

✅ SSH way

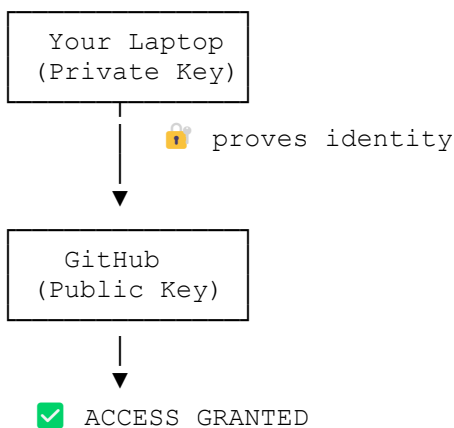
- You get a **physical key**
- Office stores a **copy of the lock**
- Only your key opens that lock 🗝️

🗝️ SSH has TWO keys (this is the core idea)

Key	Where it lives	What it does
Private key	Your laptop	Secret, never shared
Public key	GitHub	Used to verify you

🧩 SSH WORKING — VISUAL DIAGRAM

(git push / git pull)



- 👉 GitHub never sees your private key
- 👉 It only checks “**does this key match?**”

🔄 What happens during

git push


1. You type:
`git push`
2. GitHub says:
“Who are you?”
3. Your laptop answers:
“Here’s my SSH signature”
4. GitHub checks:

“Does this match the public key I saved?”

5. ☒ Yes → Push allowed
☒ No → Permission denied

Why tokens/passwords cause problems

Problem	Result
Passwords disabled	<input checked="" type="checkbox"/>
Tokens expire	<input checked="" type="checkbox"/>
Wrong token cached	<input checked="" type="checkbox"/> 403
Multiple accounts	<input checked="" type="checkbox"/> confusion

 SSH avoids **ALL** of this.

HOW TO SET UP SSH (STEP BY STEP)

We'll do it **cleanly**, once, forever.


☒ STEP 1: Create SSH key (on your laptop)

Open **Git Bash** and run:

```
ssh-keygen -t ed25519 -C "your_email@gmail.com"
```

Press **ENTER** for:

- file location
- passphrase
- confirm passphrase

 This creates:

```
~/.ssh/id_ed25519      ← private key (DO NOT SHARE)
~/.ssh/id_ed25519.pub ← public key
```

☒ STEP 2: Start SSH agent & load key

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
```

Expected:

```
Identity added
```

✅ STEP 3: Copy PUBLIC key

```
cat ~/.ssh/id_ed25519.pub
```

Copy the **entire line**
(starts with `ssh-ed25519`)

✅ STEP 4: Add key to GitHub

On GitHub:

1. **Settings**
2. **SSH and GPG keys**
3. **New SSH key**
4. Title: Shweta Laptop
5. Paste key
6. Save

🔑 Now GitHub knows your laptop.

✅ STEP 5: Test SSH login (VERY IMPORTANT)

```
ssh -T git@github.com
```

✅ Success message:

Hi thihekarshweta! You've successfully authenticated, but GitHub does not provide shell access.

👉 This means: **LOGIN SUCCESSFUL**

✅ STEP 6: Use SSH repo URL

```
git remote set-url origin git@github.com:USERNAME/REPO.git
```

Then:

```
git push
```

- 🎉 No password
 - 🎉 No token
 - 🎉 No login again
-



FINAL MENTAL MODEL (REMEMBER THIS)

SSH = Laptop identity

Public key = registered with GitHub

Private key = proof

If keys match → GitHub trusts you