**Git and GitHub: A Beginner’s Guide with Real-World Use Cases and Commands**

**Introduction to Git and GitHub**

**What is Git?**

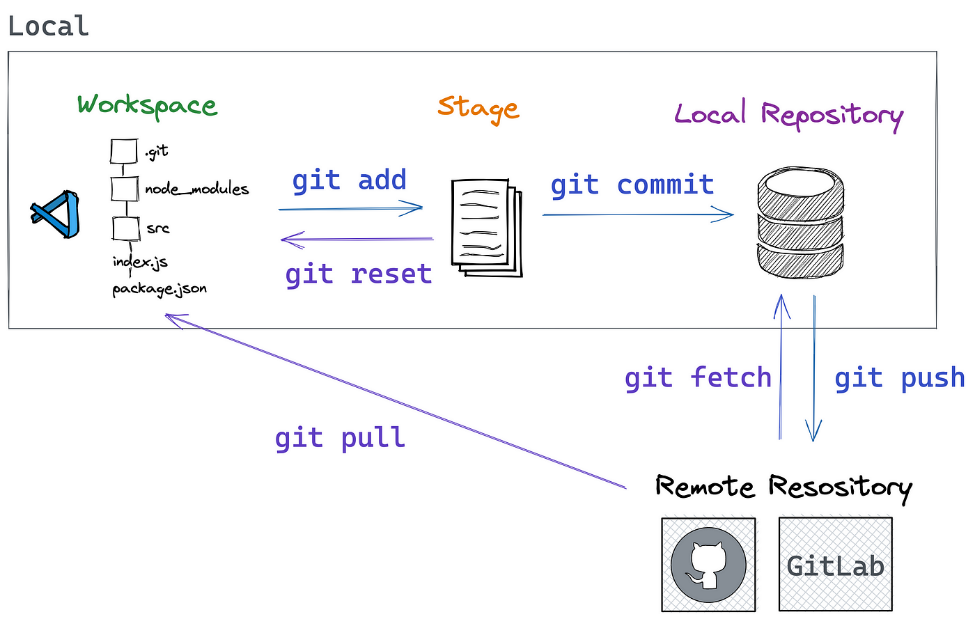
Git is a distributed version control system (VCS) that allows multiple people to work on a project simultaneously without overriding each other's changes. It helps in tracking changes to files and allows you to revert to previous versions if needed.

Imagine you're writing a big essay. Git is like a notebook where you save different versions of your essay. If you make a mistake, you can go back to an earlier version instead of starting over. It also lets multiple people edit the essay without erasing each other's work.

**What is GitHub?**

GitHub is a cloud-based platform that provides hosting for Git repositories. It allows teams to collaborate, review code, and manage projects efficiently.

GitHub is like Google Drive for your code. It stores your Git projects online so you can access them from anywhere, share them with others, and work together on them easily.



**Setting Up Git and GitHub from Scratch**

If you're new to Git and GitHub, follow this step-by-step guide to set everything up on your system.

**Step 1: Install Git**

Git needs to be installed on your local machine to use it.

**For Windows**

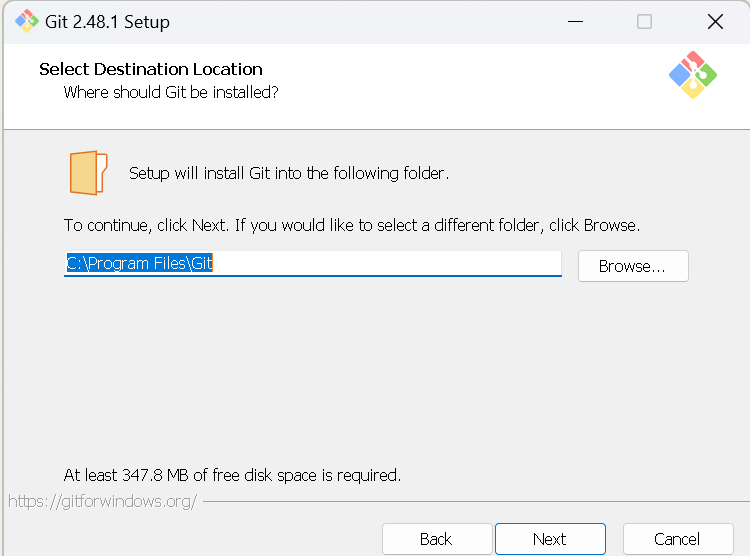
1. Download Git for Windows from [git-scm.com](https://git-scm.com/downloads).

A screenshot of a computer

AI-generated content may be incorrect.

Click 64bit Git for Windows Setup (Standard Installer)

1. Choose Windows and click Download.
2. Install it with default settings.



Open exe file🡺 click next (use default settings)

Once installed successfully, Open PowerShell with Administrator rights

To verify the installation, run:

git --version

**Step 2: Create a GitHub Account**

1. Go to [GitHub](https://github.com).
2. Click **Sign Up**.
3. Enter your email, create a username, and set a password.
4. Verify your email.

**On Windows**

1. Open Git Bash and run:

ssh-keygen -t rsa -b 4096 -C [your-email@example.com](mailto:your-email@example.com)

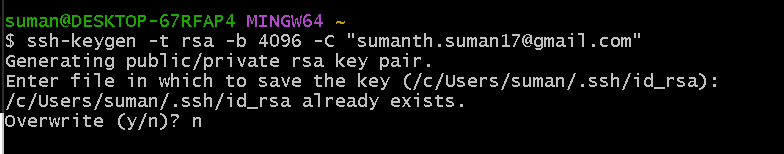
A computer screen shot of a black screen

AI-generated content may be incorrect.

Click Enter🡺 Yes

1. Open the key file in Notepad:

**c:/users/sumanth/.ssh/id\_rsa.pub**



**c:/users/sumanth/.ssh/id\_rsa.pub**

1. Open this file and copy the key.

A screenshot of a computer

AI-generated content may be incorrect.

**Add the SSH Key to GitHub**

1. Go to **GitHub > Settings > SSH and GPG keys**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Click **New SSH Key**.
2. Paste the copied key and save it.

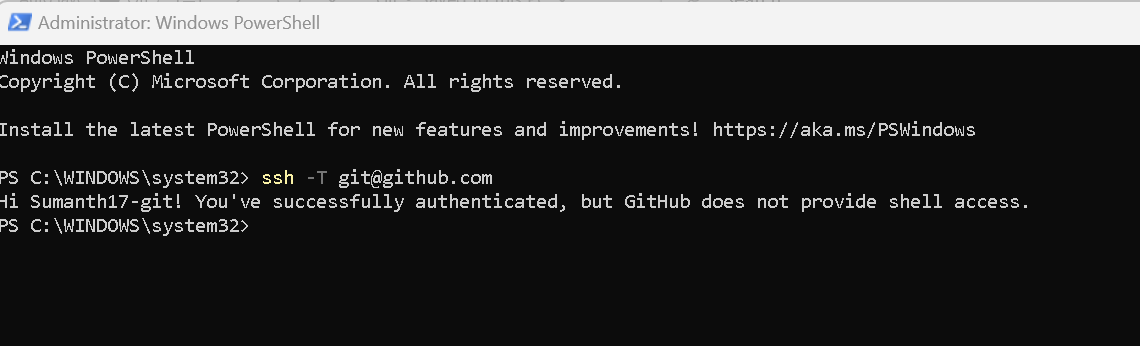
**Test SSH Connection**

Run:

ssh -T git@github.com

If successful, you'll see:

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



**Step 3: Create a New Repository on GitHub**

1. Go to GitHub and click **New Repository**.
2. Give it a name (e.g., my-first-repo).
3. Choose **Public** or **Private**.
4. Click **Create Repository**.

**Step 4: Connect Git with GitHub**

**Clone the Repository (Copy it Locally)**

git clone [git@github.com:Sumanth17-git/sre-first-project.git](mailto:git@github.com:Sumanth17-git/sre-first-project.git)

A screenshot of a computer

AI-generated content may be incorrect.

cd my-first-webservice.

**3. GitHub Permission Issues**

If SSH works but you still can't push, check:

git remote -v

If the URL starts with https://, update it to SSH:

git remote set-url origin [git@github.com:Sumanth17-git/sre-first-project.git](mailto:git@github.com:Sumanth17-git/sre-first-project.git)

git config --global user.name "Your Name"

git config --global user.email "your-email@example.com"

If SSH keeps failing, switch to HTTPS:

git remote set-url origin <https://github.com/Sumanth17-git/sre-first-project.git>

**Step 5: Make Your First Commit**

1. Create a new file: Newtestfile.txt

A screenshot of a computer

AI-generated content may be incorrect.

After you added any changes in the repository ,run

git status

this would give you list of files have updated/created

A screen shot of a computer program

AI-generated content may be incorrect.

1. Add it to Git:

git add Newtestfile.txt

or

git add -A or git add .

1. Commit the changes:

git commit -a -m "updated the changes"

A screen shot of a computer program

AI-generated content may be incorrect.

**7. Viewing Commit History**

git log

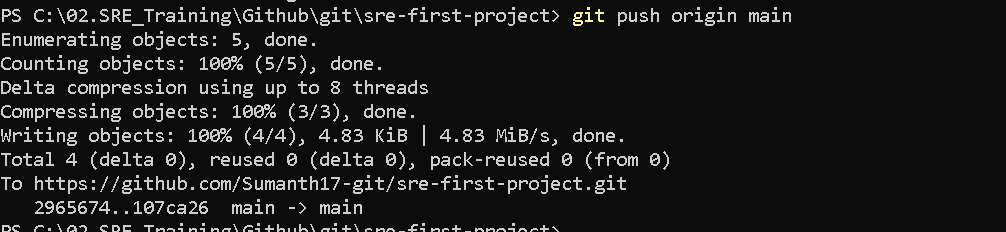
 **Purpose:** Displays commit history.

 **Real-world Use Case:** Checking who made what changes and when

**To see a summary:**

**git log –oneline**

1. Push the changes to GitHub:

git push origin main  


The changes are successfully uploaded into github repository. Let’s verify in GitHub, as you can see we have successfully pushed the changes to GitHub( i.e. Cloud storage) for future use.

A screenshot of a computer

AI-generated content may be incorrect.

**Pulling Changes from GitHub**

**git pull origin <branch\_name>**

* Purpose: Updates the local repository with the latest changes.
* Real-world Use Case: A developer ensures they have the latest code before making changes.

Now we will discuss what is branching in Git

**🔹 What is Branching in GitHub?**

**Branching** in GitHub (and Git) is a feature that allows you to create independent versions of your code to work on new features, bug fixes, or experiments without affecting the main project.

Think of **branches** like separate workspaces where you can make changes safely and later merge them back into the main branch

**🔹 Why Use Branching?**

✅ Work on **new features** without disturbing the main code  
✅ Fix **bugs** without affecting ongoing development  
✅ Collaborate with a team by working on separate branches  
✅ Experiment safely and **merge only when ready**

**Hands-On: GitHub Branching Example**

**Check Existing Branches**

git branch

 Shows all **local** branches.

 The current branch is marked with \*.

**To see remote branches:**

git branch -r

**2️.Create a New Branch**

**git branch sre-change**

* This creates a new branch called feature-branch, but you’re still on the current branch.

To **create and switch** to it immediately:

git checkout -b sre-change

**3️.Switch to a Different Branch**

git checkout sre-change

OR in modern Git versions (recommended):

git switch sre-change

4.Make changes in Branch and commit it in branch.

git add -A

git commit -a -m “committed the changes”

**4️.Push the Branch to GitHub**

After making changes, push your branch to GitHub:

git push origin sre-change

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Now I have made changes only branch **sre-change** and these changes are not merged into **main** branch.

To Merge the changes from **sre-change** branch to **main** branch

git merge <branch\_name>

* Purpose: Combines changes from one branch into another.
* Real-world Use Case: After completing a feature, it is merged into main.

Example:

Switch to main branch and merge the sre-change

git checkout main

git merge sre-change

A screen shot of a computer

AI-generated content may be incorrect.

It will open the vim editor like below

Save and exit using below command

**wq!**

A computer screen with blue text

AI-generated content may be incorrect.

And then push the changes to main branch

git push origin main

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

To Delete the branch after merging into main branch

git branch

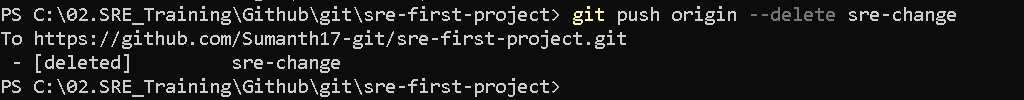
git branch -D sre-change

A screen shot of a computer code

AI-generated content may be incorrect.

To delete the branch on GitHub UI

**git push origin –delete sre-change**



A screenshot of a computer

AI-generated content may be incorrect.

**Troubleshooting**

To Check the commit history

git log –oneline

To check the configuration

git config –list

To Setup the configuration.

git config --global user.name "Your Name"

git config --global user.email [your-email@example.com](mailto:your-email@example.com)

**GitHub Permission Issues**

If SSH works but you still can't push, check:

git remote -v

If the URL starts with https://, update it to SSH:

git remote set-url origin [git@github.com:your-username/repository-name.git](mailto:git@github.com:your-username/repository-name.git)

If SSH keeps failing, switch to HTTPS:

git remote set-url origin https://github.com/your-username/repository-name.git

**GitHub/SRE Use Case – Interview Point of View**

**Scenario 1: How do you apply the Protecting the Main Branch in GitHub**

**Use Case:** You want to **prevent accidental pushes** to main and **enforce code reviews**.

**Step 1: Enable Branch Protection**

1. Go to **GitHub > Your Repository > Settings**.
2. Click on **Branches** → **Add branch protection rule**.
3. Set:
   * **Require pull request reviews before merging** ✅
   * **Require status checks before merging** ✅
   * **Restrict who can push to this branch** ✅

A screenshot of a computer

AI-generated content may be incorrect.

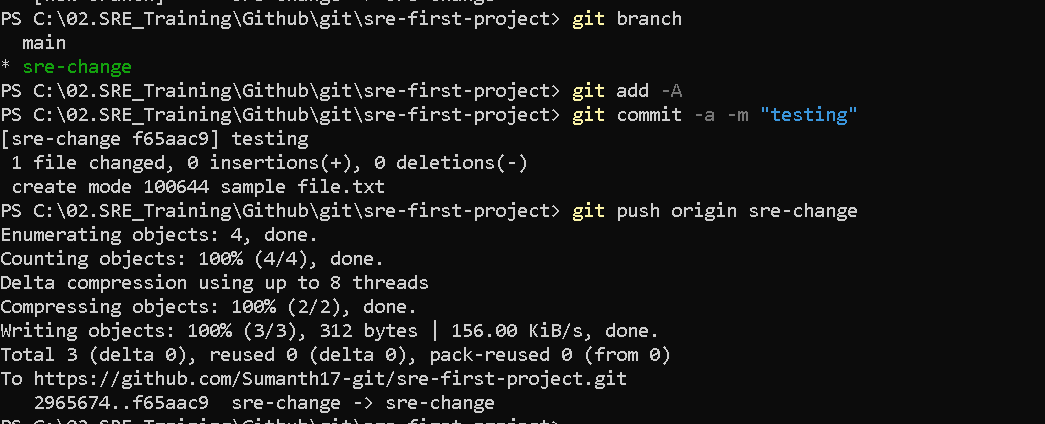
**How do you create the pull request from GitHub UI**

Firstly go the Repository 🡺 Settings 🡺 Collaborators 🡺Add your friend/teammate Github id. So he will approve the change you make.

**A screenshot of a computer

AI-generated content may be incorrect.**

When you push any changes into branch.



Now we need to merge this change into main branch. We need to create the pull request for this

Click Pull Requests 🡺 New pull request 🡺

A screenshot of a computer

AI-generated content may be incorrect.

Choose base : main and compare : sre-change

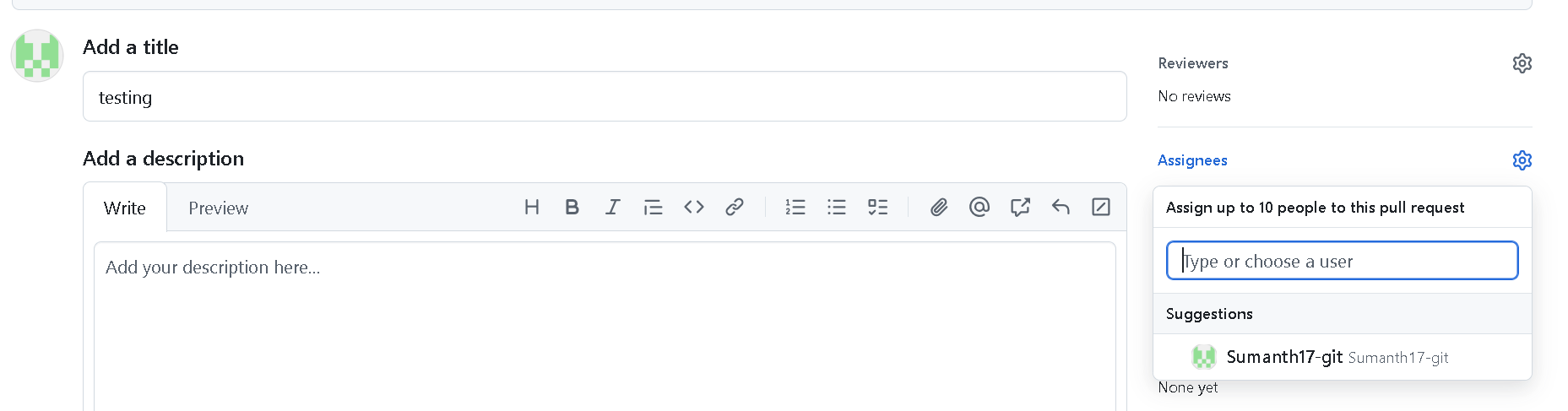
As we are planning the merge the changes from sre-change branch to main branch

A screenshot of a computer

AI-generated content may be incorrect.

**Click Create Pull request**

Add assignee and updated title and description



Click Create Pull Request

A screenshot of a computer

AI-generated content may be incorrect.

Now you can see , Review is required , so your team-mate/manager has to approve your changes.

A screenshot of a computer

AI-generated content may be incorrect.

Now your manager will approve the change after review the changes.

A screenshot of a computer

AI-generated content may be incorrect.

**Click Submit Review .**

Once your manager the Pull request , you can merge the changes into Main Branch , **Click Merge Pull Request and confirm the merge.**

A screenshot of a chat

AI-generated content may be incorrect.

Now your changes are successfully merged into Main Branch

A screenshot of a chat

AI-generated content may be incorrect.

**Scenario 2: How do you Secure Authentication Using GitHub Personal Access Token (PAT)**

**Use Case:** Instead of SSH, you want to use HTTPS authentication with **Personal Access Tokens**.

**Step 1: Generate a GitHub Token**

1. Go to **GitHub > Settings > Developer Settings > Personal Access Tokens**.
2. Click **Generate New Token**.
3. Select:
   * repo (Full control)
   * workflow (GitHub Actions)
4. Copy the **token**.

**Step 2: Set Up Git to Use the Token**

Replace your GitHub remote URL:

git remote set-url origin https://your-username@github.com/your-username/repository-name.git

Now, when pushing:

git push origin main

Git will **prompt for your token** instead of a password.

A screenshot of a chat

AI-generated content may be incorrect.

Now, **no one can push directly to main without review**.

**Scenario 3 : How do you Bug Fixing in Production using GitHub process**

**Use Case: Urgent Bug Fix**

**A critical bug is found in production, affecting all users. The bug needs to be fixed immediately.**

**Step-by-Step Hands-On Example**

**Step 1: Create a Hotfix Branch**

**git checkout -b hotfix-urgent-bug**

**Step 2: Fix the Bug and Commit**

**Alice modifies the faulty code in checkout.js.**

git add checkout.js

git commit -m "Fixed checkout bug in production"

**Step 3: Push and Merge Directly into Production**

**Since this is an urgent fix, the team merges it into the main branch immediately.**

git checkout main

git merge hotfix-urgent-bug

git push origin main

**Scenario 4: Ho does Stashing Changes to Work on GitHub**

**Use Case: Switching Tasks Without Losing Work**

For example Bob is working on a new feature, but an urgent bug needs fixing.

Step 1: Stash the Current Work

git stash

Step 2: Switch to the Bug Fix Branch

git checkout -b hotfix-payment-error

Bob fixes the bug, commits, and pushes.

Step 3: Restore the Stashed Work

git checkout feature-payment

git stash pop

**Scenario 5: How to do Undoing Mistakes**

**Use Case: Reverting a Mistaken Commit**

Bob accidentally committed sensitive credentials.

Step 1: Identify the Commit

git log --oneline

Step 2: Revert the Commit

git revert <commit\_id>

This creates a new commit that undoes the previous commit.

**Scenario 6: How to Resolve the Merge Conflicts**

**Use Case: Two Developers Modify the Same File**

1. **Alice updates cart.js and pushes changes.**
2. **Bob also modifies cart.js in a different branch and tries to merge.**
3. **Conflict occurs.**

Step-by-Step Hands-On Example

Step 1: Bob Tries to Merge

git checkout main

git merge feature-cart-update

Error: Merge conflict in cart.js.

Step 2: Resolve the Conflict

Bob opens cart.js, manually fixes the conflicting lines, and saves the file.

Step 3: Add and Commit the Resolved File

git add cart.js

git commit -m "Resolved merge conflict in cart.js"

git push origin main

**Scenario 7: Revert a Specific Commit (If You Want to Undo an Older Commit)**

If you want to revert a specific commit, find the commit hash from git log --oneline and run:

git log –oneline

A screen shot of a computer

AI-generated content may be incorrect.

If you want to revert last change “files added” ,copy the commit id

git revert <commit-hash>

For example, if you want to revert commit 6488f39

git revert 6488f39

wq!

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

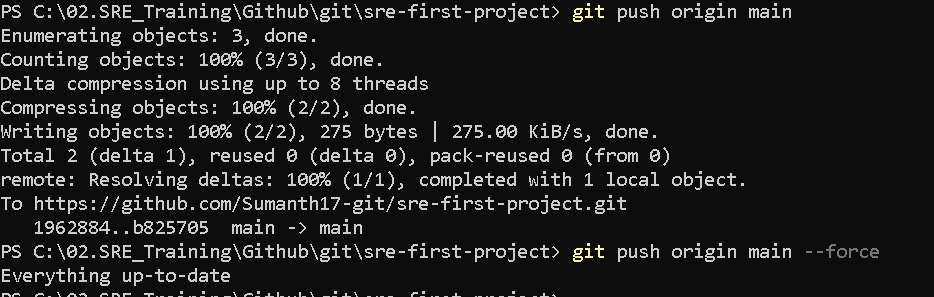
AI-generated content may be incorrect.

This creates a new commit that undoes the faulty one, ensuring a rollback without rewriting history.

**Pushing the Changes After Revert**

If you used **reset** and already pushed your changes, you may need to force push:

git push origin master –force



**Scenario 8.Scenario: A Developer Deleted a Critical Branch**

Problem:

A teammate accidentally deleted a critical feature branch before merging.

Solution:

1. Find the deleted branch in Git reflog

git reflog

1. Restore the branch

git checkout -b recovered-branch <commit-hash>

1. Push it back to GitHub

git push origin recovered-branch

1. Inform the team and prevent accidental deletions by enabling branch protection rules.

Prevention Best Practices:

✅ Enable protected branches to prevent accidental deletions  
✅ Set up branch policies in GitHub  
✅ Use feature flags so incomplete features don’t need long-lived branches

A screenshot of a computer program

AI-generated content may be incorrect.