

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

Introduction to Git and GitHub

What is Git?

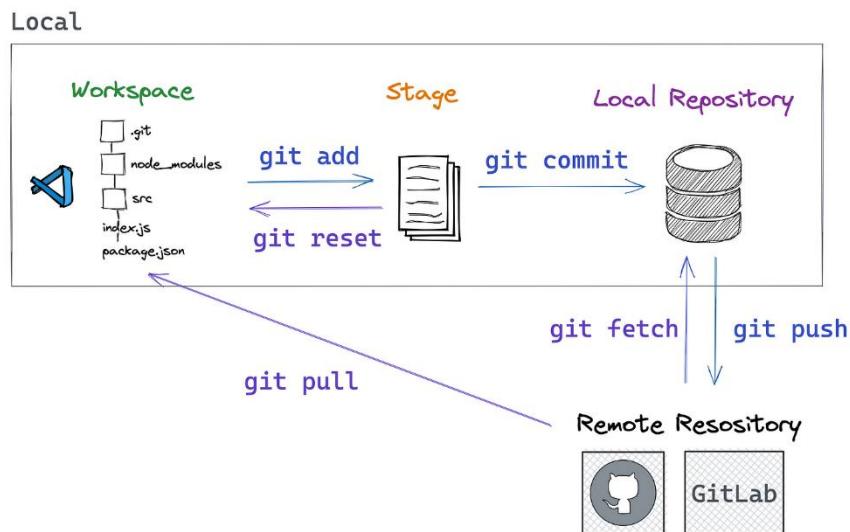
Git is a distributed version control system (VCS) that allows multiple people to work on a project simultaneously without overriding each other's changes. It helps in tracking changes to files and allows you to revert to previous versions if needed.

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.

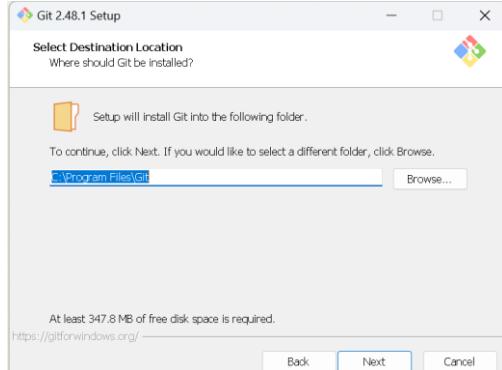
For Windows

1. Download Git for Windows from git-scm.com.



Click 64bit Git for Windows Setup (Standard Installer)

2. Choose Windows and click Download.
3. 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](#).
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
```

```
suman@DESKTOP-G7RFAP4 MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "sumanth.suman17@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/suman/.ssh/id_rsa): |
```

Click Enter ➔ Yes

2. Open the key file in Notepad:

```
c:/users/sumanth/.ssh/id_rsa.pub
```

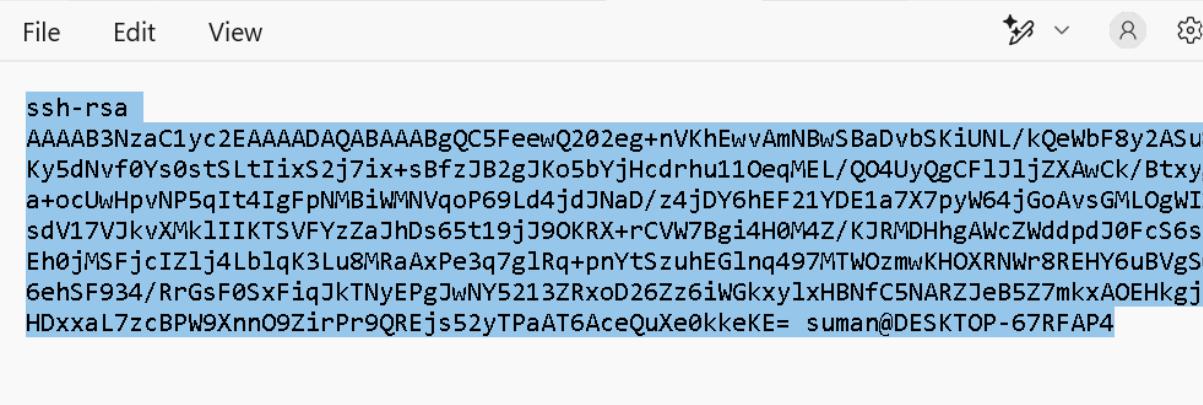
```

suman@DESKTOP-67RFAP4 MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "sumanth.suman17@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/suman/.ssh/id_rsa):
/c/Users/suman/.ssh/id_rsa already exists.
Overwrite (y/n)? n

```

c:/users/sumanth/.ssh/id_rsa.pub

3. Open this file and copy the key.



The screenshot shows a code editor window with a single line of text selected. The text is a long string of characters representing an RSA public key. The line starts with 'ssh-rsa' and ends with the user's email address and the host name. The background of the editor is light blue, and the selected text is highlighted in white.

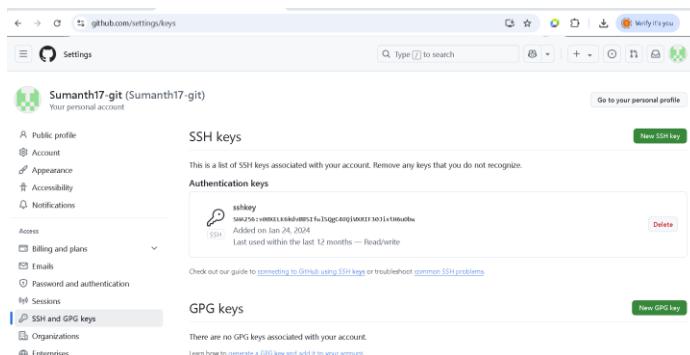
```

ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQgQC5FeewQ202eg+nVKhEwvAmNBwSBaDvbSKiUNL/kQewbF8y2ASu
Ky5dNvf0Ys0stSLtIixS2j7ix+sBfzJB2gJKo5bYjHcdrhu11OeqMEL/Q04UyQgCF1J1jZXAwCk/Btxy
a+ocUwHpvNP5qIt4IgFpNMBiWMNVqoP69Ld4jdJNaD/z4jDY6hEF21YDE1a7X7pyW64jGoAvsGML0gWI
sdV17VJkvXMk1IIKTSVFYzZaJhDs65t19jJ9OKRX+rCVW7Bgi4H0M4Z/KJRMDHhgAWcZWddpdJ0FcS6s
Eh0jMSFjcIZljqK3Lu8MRaAxPe3q7glRq+pnYtSzuhEGlnq497MTWOzmwKHOXRNRWr8REHY6uBVgS
6ehSF934/RrGsF0SxFiqJkTNyEPgJwNY5213RxoD26Zz6iWGkxylxHBNfc5NARZJeB5Z7mkxAOEHkgj
HDxxaL7zcBPW9Xnn09ZirPr9QREjs52yTPaAT6AceQuXe0kkeKE= suman@DESKTOP-67RFAP4

```

Add the SSH Key to GitHub

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



2. Click New SSH Key.

3. 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.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> ssh -T git@github.com
Hi Sumanth17-git! You've successfully authenticated, but GitHub does not provide shell access.
PS C:\WINDOWS\system32>
```

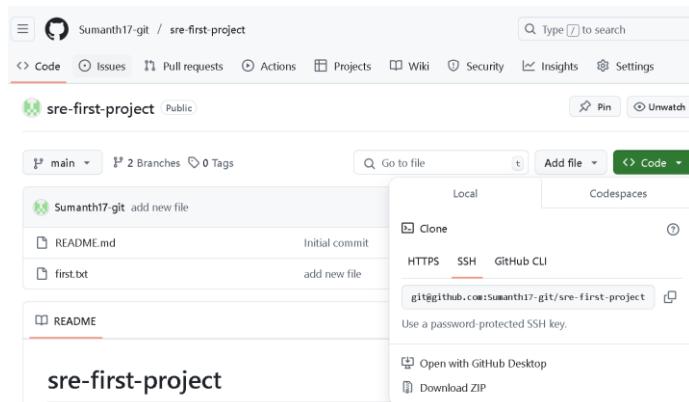
Step 3: Create a New Repository on GitHub

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
```



```
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
```

```
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



After you added any changes in the repository ,run

```
git status
```

this would give you list of files have updated/created

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Newfile.pub
    testing.txt

nothing added to commit but untracked files present (use "git add" to track)
```

2. Add it to Git:

```
git add Newtestfile.txt
```

or

```
git add -A or git add .
```

3. Commit the changes:

```
git commit -a -m "updated the changes"
```

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Newfile.pub
    testing.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\02.SRE_Training\Github\git\sre-first-project> git add -A
PS C:\02.SRE_Training\Github\git\sre-first-project> git add .
PS C:\02.SRE_Training\Github\git\sre-first-project> git commit -a -m "updated the changes"
[main 107ca26] updated the changes
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Newfile.pub
 create mode 100644 testing.txt
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

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

4. Push the changes to GitHub:

git push origin main

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 4.83 KiB | 4.83 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Sumanth17-git/sre-first-project.git
  2965674..107ca26  main -> main
ps C:\02.SRE_Training\Github\git\sre-first-project>
```

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.

The screenshot shows a GitHub repository named 'sre-first-project' under the user 'Sumanth17-git'. The 'Code' tab is selected. The main branch is shown with four commits:

- Sumanth17-git updated the changes (1 minute ago)
- Newfile.pub updated the changes (1 minute ago)
- README.md Initial commit (3 hours ago)
- first.txt add new file (3 hours ago)
- testing.txt updated the changes (1 minute ago)

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`

2.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`

3.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"`

4.Push the Branch to GitHub

After making changes, push your branch to GitHub:

`git push origin sre-change`

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin sre-change
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 5.25 KiB | 2.62 MiB/s, done.
Total 8 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/Sumanth17-git/sre-first-project.git
  9796cdf..6488f39  sre-change -> sre-change
```

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

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\02.SRE_Training\Github\git\sre-first-project> git merge sre-change
Merge made by the 'ort' strategy.
Newtestfile.txt      |  0
.../dt-account-management-users-3-9-2025.csv | 44 ++++++-----+
second.txt          |  1 +
srefile.txt         |  0
testing.pub         | Bin 0 -> 59904 bytes
5 files changed, 45 insertions(+)
create mode 100644 Newtestfile.txt
create mode 100644 mydocuments/dt-account-management-users-3-9-2025.csv
create mode 100644 second.txt
create mode 100644 srefile.txt
create mode 100644 testing.pub
```

It will open the vim editor like below

Save and exit using below command

wq!



And then push the changes to main branch

```
git push origin main
```

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 323 bytes | 323.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Sumanth17-git/sre-first-project.git
  107ca26..1962884  main -> main
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

The screenshot shows a GitHub repository named 'sre-first-project'. The main branch has 2 branches and 0 tags. The commit history for the main branch is as follows:

Commit	Message	Time Ago
Sumanth17-git Merge branch 'sre-change'	1962884 · 2 minutes ago	9 Commits
mydocuments	added new folder	3 hours ago
Newfile.pub	updated the changes	19 minutes ago
Newtestfile.txt	Added a new file	24 minutes ago
README.md	Initial commit	3 hours ago
first.txt	add new file	3 hours ago
second.txt	testing	3 hours ago
srefile.txt	files added	4 minutes ago
testing.pub	testing	23 minutes ago
testing.txt	updated the changes	19 minutes ago

To Delete the branch after merging into main branch

```
git branch
```

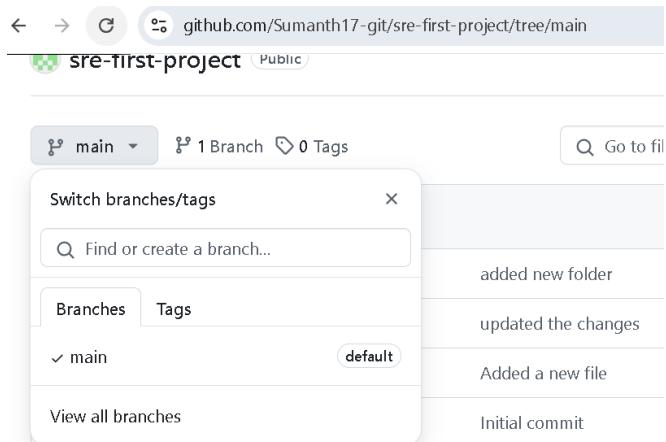
```
git branch -D sre-change
```

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git branch
* main
  sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project> git branch -D sre-change
Deleted branch sre-change (was 6488f39).
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

To delete the branch on GitHub UI

```
git push origin --delete sre-change
```

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin --delete sre-change
To https://github.com/Sumanth17-git/sre-first-project.git
 - [deleted]      sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project>
```



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
```

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
```

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

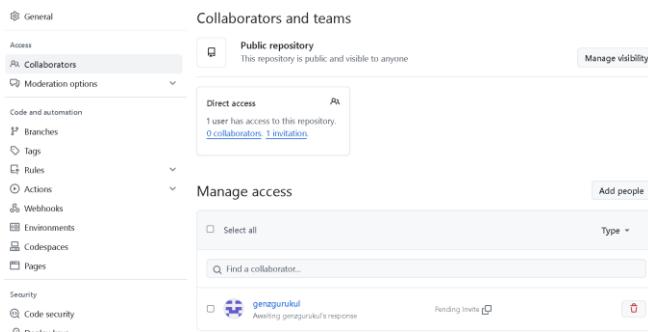
1. Go to **GitHub > Your Repository > Settings**.
2. Click on **Branches → Add branch protection rule**.
3. Set:

- Require pull request reviews before merging**
- Require status checks before merging**
- Restrict who can push to this branch**

The screenshot shows the 'Branch name pattern' field containing 'main'. Below it, under 'Protect matching branches', two rules are listed: 'Require a pull request before merging' (checked) and 'Require approvals' (checked). A dropdown menu indicates 'Required number of approvals before merging: 1'.

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.



When you push any changes into branch.

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git branch
  main
* sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project> git add -A
PS C:\02.SRE_Training\Github\git\sre-first-project> git commit -a -m "testing"
[sre-change f65aac9] testing
  1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 sample file.txt
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin sre-change
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 312 bytes | 156.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Sumanth17-git/sre-first-project.git
  2965674..f65aac9  sre-change -> sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

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 →

 Sumanth17-git / sre-first-project           Code  Issues  Pull requests  Actions  Projects  Wiki  Security  Insights  Settings

 sre-change had recent pushes 1 minute ago

Compare & pull request

Filters ▾

Labels 9

 Milestones 0

New pull request

Choose base : main and compare : sre-change

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

The screenshot shows a GitHub pull request interface. At the top, there's a header with user info ('Sumanth17-git / sre-first-project'), a search bar, and various navigation icons. Below the header is a navigation bar with links: Code (which is underlined in red), Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area has a title 'Comparing changes' and a sub-instruction: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#)'. Below this, there's a dropdown menu for 'base: main' and another for 'compare: sre-change'. A green checkmark icon indicates 'Able to merge'. A note says 'These branches can be automatically merged.' At the bottom, there's a blue bar with the text 'Discuss and review the changes in this comparison with others. [Learn about pull requests](#)' and a green button labeled 'Create pull request'.

Click Create Pull request

Add assignee and updated title and description

Add a title

Add a description

[Write](#) [Preview](#)

Add your description here...

Reviewers

No reviews

Assignees

Assign up to 10 people to this pull request

Type or choose a user

Suggestions

Sumanth17-git Sumanth17-git

None yet

Click Create Pull Request

The screenshot shows the GitHub pull request creation interface. At the top, there's a search bar with 'testing'. Below it, a section for 'Add a description' with 'Write' and 'Preview' tabs, and a rich text editor. To the right, there are sections for 'Assignees' (Sumanth17-git), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone). A note says 'Development' and 'Use Closing keywords in the description to automatically close issues'. A 'Create pull request' button is at the bottom.

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

This screenshot shows a GitHub pull request for 'testing' #1. It's a merge conflict between the 'main' branch and the 'sre-change' branch. The pull request is blocked because a review is required. A self-assigned user (Sumanth17-git) is listed. The review section shows a 'Review required' message and a 'Merge is blocked' message.

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

This screenshot shows the GitHub pull request review interface for 'testing' #1. It displays a 'Finish your review' dialog with options to leave a comment, approve, or request changes. The 'Approve' option is selected.

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.**

The screenshot shows a GitHub pull request merge interface. At the top, it says "testing #1 Open" and "Sumanth17-git wants to merge 1 commit into `main` from `sre-change`". Below this, there's a "No description provided." section. The commit history shows a merge commit from the `testing` branch. A note indicates "Sumanth17-git self-assigned this 4 minutes ago". A review by "genzgurukul" is shown with a green checkmark icon and the message "approved these changes now". In the bottom right corner of the main area, there are two buttons: "Merge pull request" and "View".

Now your changes are successfully merged into Main Branch

The screenshot shows a GitHub pull request merge interface. At the top, it says "Merged" and "testing #1". Below this, a note says "Pull request successfully merged and closed. You're all set — the `sre-change` branch can be safely deleted." There is a "Add a comment" section with "Write" and "Preview" buttons, and a rich text editor toolbar below it.

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.

Applies to 1 branch

The screenshot shows the GitHub repository settings for the 'main' branch. It highlights a 'Protect matching branches' section. Under this section, two rules are listed: 'Require a pull request before merging' (which is checked) and 'Require approvals' (which is also checked). A dropdown menu indicates that 1 approval is required before merging.

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`

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git log --oneline
1962884 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'sre-change'
5488f39 files added
107ca26 updated the changes
ca2fbb3 testing
e9fbcd1 Added a new file
9796cdf added new folder
c00789c testing
2965674 add new file
c19fd78 Initial commit
```

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!

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git log --oneline
1962884 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'sre-change'
6488f39 files added
107ca26 updated the changes
ca2fbdb3 testing
e9fbbed1 Added a new file
9796cdf added new folder
c00789c testing
2965674 add new file
c19fd78 Initial commit

PS C:\02.SRE_Training\Github\git\sre-first-project> ^C
PS C:\02.SRE_Training\Github\git\sre-first-project> git revert 6488f39
[main b825705] Revert "files added"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 srefile.txt
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

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
```

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 275 bytes | 275.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Sumanth17-git/sre-first-project.git
  1962884..b825705  main -> main
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin main --force
Everything up-to-date
PS C:\02.SRE_Training\Github\git\sre-first-project>
```

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
```

2. Restore the branch

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

3. Push it back to GitHub

```
git push origin recovered-branch
```

4. 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

```
PS C:\02.SRE_Training\Github\git\sre-first-project> git reflog
b825705 (HEAD -> main, origin/main, origin/HEAD) HEAD@{0}: revert: Revert "files added"
1962884 HEAD@{1}: merge sre-change: Merge made by the 'ort' strategy.
107ca26 HEAD@{2}: checkout: moving from sre-change to main
6488f39 HEAD@{3}: commit: files added
ca2fbb3 HEAD@{4}: checkout: moving from main to sre-change
107ca26 HEAD@{5}: commit: updated the changes
2965674 HEAD@{6}: checkout: moving from sre-change to main
ca2fbb3 HEAD@{7}: commit: testing
e9fbbed1 HEAD@{8}: commit: Added a new file
9796cdf HEAD@{9}: commit: added new folder
c00789c HEAD@{10}: commit: testing
2965674 HEAD@{11}: checkout: moving from main to sre-change
2965674 HEAD@{12}: commit: add new file
c19fd78 HEAD@{13}: clone: from https://github.com/Sumanth17-git/sre-first-project.git
PS C:\02.SRE_Training\Github\git\sre-first-project> git checkout -b sre-change 2965674
Switched to a new branch 'sre-change'
PS C:\02.SRE_Training\Github\git\sre-first-project> git branch
  main
* sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project> git push origin sre-change
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'sre-change' on GitHub by visiting:
remote:     https://github.com/Sumanth17-git/sre-first-project/pull/new/sre-change
remote:
To https://github.com/Sumanth17-git/sre-first-project.git
 * [new branch]      sre-change -> sre-change
PS C:\02.SRE_Training\Github\git\sre-first-project>
```