Linux & Git Commands		
Category	**Command**	**Short Description**
Linux	mkdir <directory_name>	Creates a new folder
Linux	cd <directory_name>	Moves into the git-lab directory
Linux	ls -a	Lists files/folders in current directory, including hidden
Linux	cat <file_name>	Displays content of file
Linux	echo Hello Git > hello.txt	Creates hello.txt with text
Linux	echo Welcome >> hello.txt	Appends text to existing file
Linux	<command_name> --help	Show usage options for Linux command
Git	git help <command_name>	Opens git command documentation
Git Setup	git --version	Checks installed Git version
Git Setup	git config --global user.name "Name"	Sets Git username
Git Setup	git config --global user.email "email"	Sets Git email
Git Setup	git config --global --list	Displays Git configuration
Git Repo	git init	Initializes a local Git repository
Git Status	git status	Shows file and commit status
Git Tracking	git add hello.txt	Adds file to staging area
Git Tracking	git add .	Stages all modified files
Git Commit	git commit -m "message"	Saves changes to Git history
Git History	git log	Shows full commit history
Git History	git log --oneline	Shows compact commit history
Git Compare	git diff	Shows changes before commit
Git Branch	git branch	Lists, Create or Delete branches
Git Branch	git branch <branch_name>	Creates new branch
Git Branch	git checkout <branch_name>	Switches to other branch
Git Branch	git checkout master	Switches back to main branch
Git Remote	git remote add origin <url>	Connects local repo to GitHub
Git Remote	git remote set-url origin <url>	Updates remote repository URL
Git Remote	git remote -v	Shows linked remote URLs
Git Push	git push -u origin master	Pushes code to GitHub
Git Push	git push	Pushes latest commits
Git Branch	git branch -M master	Renames branch to master
Git Branch	git branch <branch_name>	Create a new branch
Git Branch	git checkout <branch_name>	Switch branch
Git Push	git push --set-upstream origin <branch_name>	Pushes branch to GitHub
Git Merge	git merge <branch_name>	Merge <branch_name> to master/main
Git Sync	git pull	Fetches and merges remote changes
Undo Before Commit	git restore <file_name>	Restores file to last committed version (cancels unsaved edits)
Soft Reset	git reset --soft HEAD~1	Removes last commit but keeps changes staged
Hard Reset	git reset --hard HEAD~1	Removes last commit and deletes all related changes permanently
Recovery Log	git reflog	Shows history of HEAD movements, including deleted commits
Recover Deleted Commit	git reset --hard <commit-id>	Moves repository back to a selected commit from reflog
Linux	mkdir -p <directory_name>	Create nested folders/directories as needed
Linux	touch <file_name>	Create empty file

Git Foundations Lab

Day 1 & Day 2 – Local Git History and Commit Discipline

DAY 1: Git Setup, Local History & First Commits

---

1. Day 1 Objective

By the end of Day 1, you will be able to:

- Install Git on a Windows computer
- Open Git Bash and run Git commands
- Verify Git is working
- Tell Git who you are
- Create a local Git repository
- Save your work using commits
- See how Git tracks changes over time

All work stays **only on your local machine**.

---

2. Basic Theory

Think of Git as a **history notebook for a folder**.

- Your files are the current page
- Each commit is a saved page
- Git never saves automatically
- You decide when history is written

Section 1: Git Installation on Windows

Why this step exists

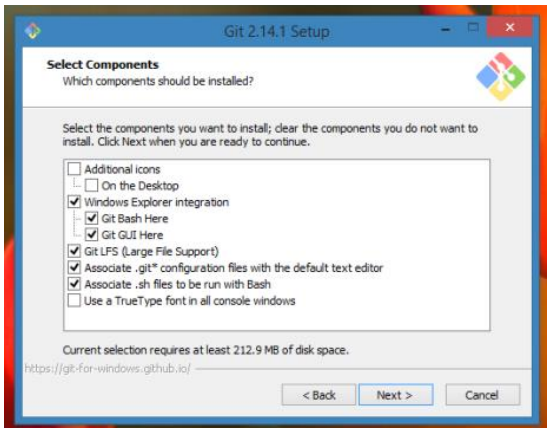
Git commands only work after Git is installed.

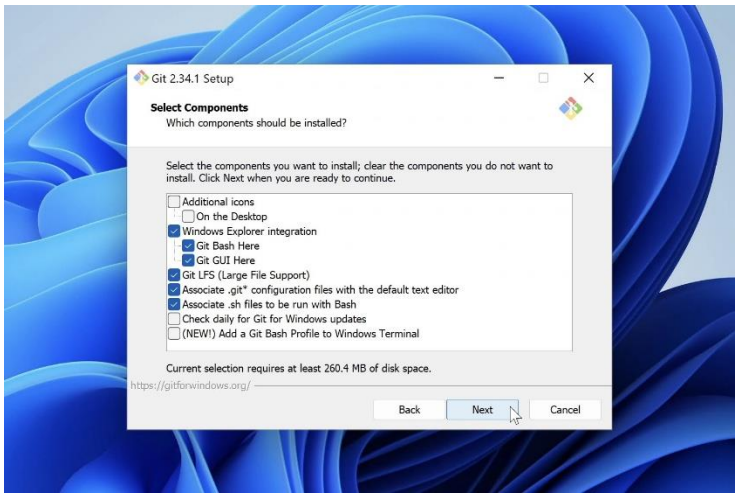
Step-by-Step Instructions

1. Open a web browser

2. Go to https://git-scm.com
3. Click **Download for Windows**
4. Run the installer
5. Keep **all default options**
6. Ensure **Git Bash** is selected
7. Finish installation

What You Should See (Installer)



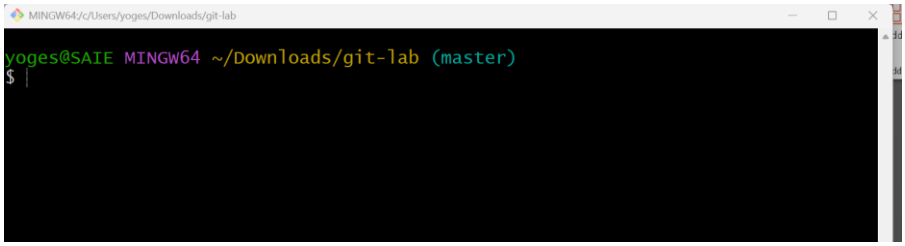


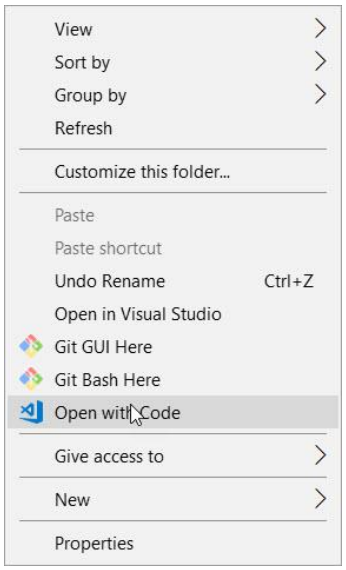
Git Commands are always run on Git Bash

To open Git Bash:

- Right-click on Desktop or inside any folder
- Click **Git Bash Here**

What Git Bash Looks Like





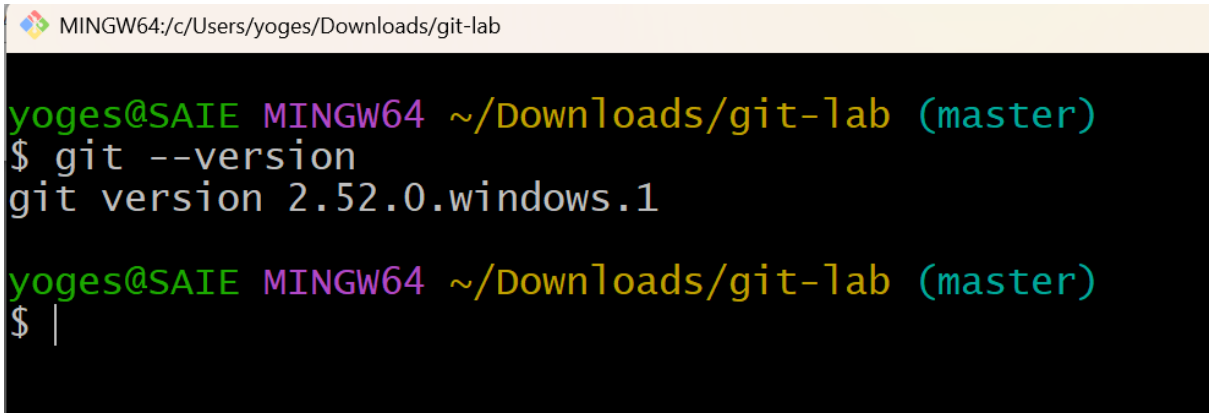
Checkpoint

- Git installed
- Git Bash opens successfully

Section 2: Verify Git Installation

git --version

What You Should See



Expected output:

git version 2.x.x

Section 3: Git Identity Configuration (15 minutes)

Why this step is important ? Git records who made each commit.

Commands

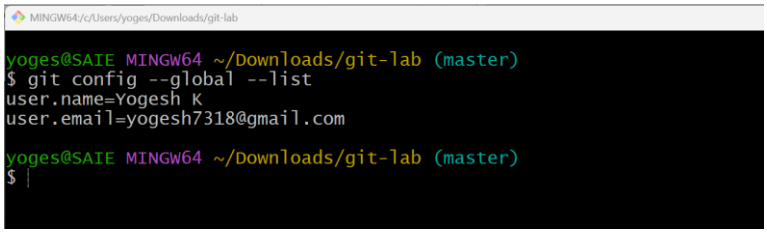
git config --global user.name "Your Full Name"

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

Verify:

git config --global --list

What You Should See



Not all Unix/Linux commands run in Git Bash. Git Bash supports many common commands like *ls, cd, mkdir, rm, cat*, but it does not include every Linux utility because it is not a full Linux operating system.

Section 4: Create Your First Repository

A repository is where Git starts tracking history.

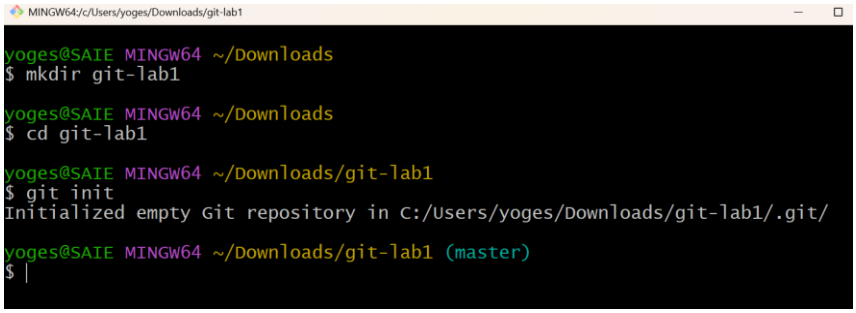
Commands to Type

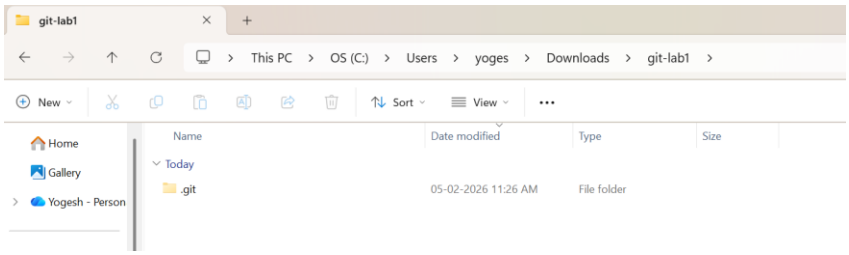
mkdir git-lab1

cd git-lab1

git init

What You Should See





Checkpoint

- .git folder exists (hidden)
- Repository initialized

Section 5: First File and First Commit - Commits permanently save your work.

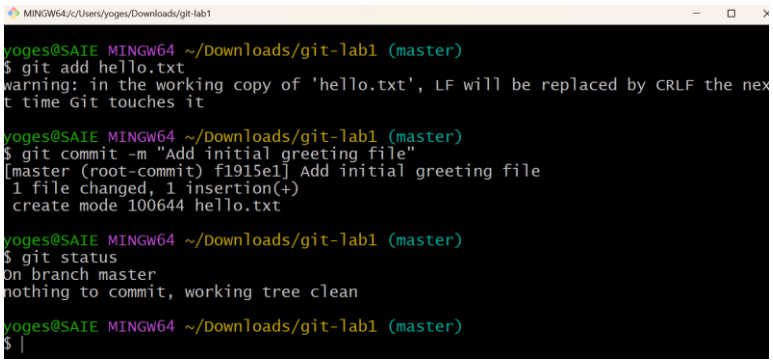
echo Hello Git > hello.txt

git status

git add hello.txt

git commit -m "Add initial greeting file"

What You Should See



Checkpoint

- Commit created
- Working tree clean

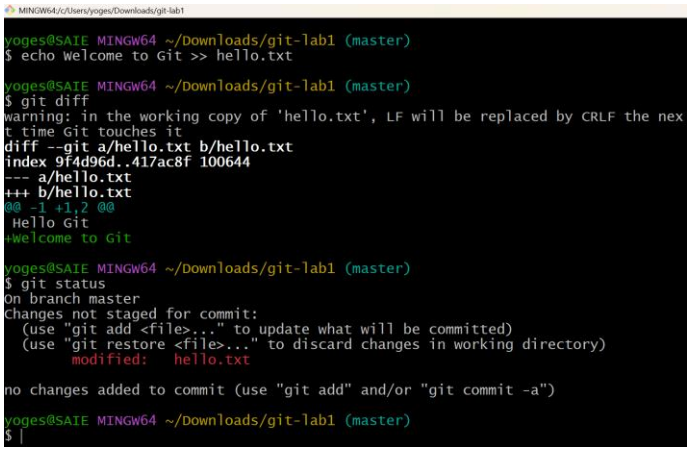
Section 6: Understanding File Changes with git diff

echo Welcome to Git >> hello.txt

git diff

git status

What You Should See



Checkpoint

- Changes visible before commit

Section 7: Commit Discipline Practice

Small, clear commits make history readable.

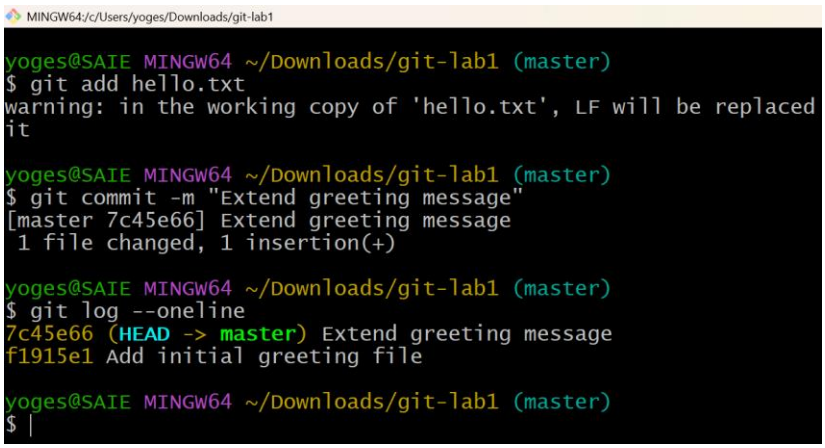
Commands

git add hello.txt

git commit -m "Extend greeting message"

git log --oneline

What You Should See



End of Day 1 Validation

Students should have:

- Git installed
- Git Bash used correctly
- At least **2 commits**
- Clean working tree

DAY 2: Local History Mastery & Commit Discipline

---

1. Day Objective

By the end of Day 2, you will be able to:

- Make multiple clean commits
- Control what goes into each commit
- Write meaningful commit messages
- Explain why commit history matters

---

2. Basic Theory

Git history is a **story of decisions**.

Each commit should answer:

- What changed?
- Why did it change?
- When did it change

Section 1: Incremental Change Tracking

Repeat **three times**:

1. Edit hello.txt
2. Run **git diff**
3. Commit with a clear message

Example messages:

- Add opening greeting
- Clarify welcome message
- Add closing line

Section 2: Staging Area Control

- Make two edits to hello.txt
- Only once run **git add hello.txt**
- Observe **git status**

Section 3: Good vs Bad Commit Messages

Bad

update

fix

changes

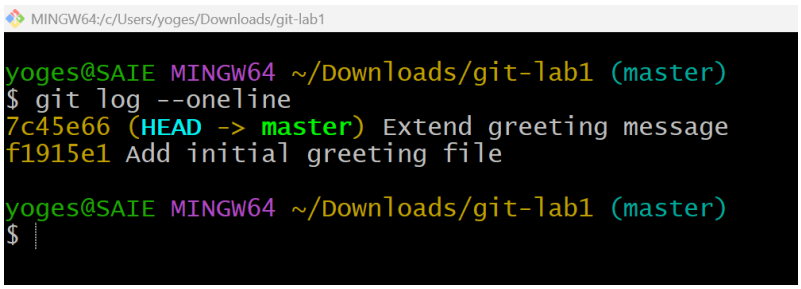
Good

Clarify greeting for new users

Improve readability of welcome message

Section 4: Commit History Review

git log --oneline



Final Validation

Students must have:

- Minimum **5 commits**
- Clear commit messages
- Clean working tree
- Confidence using Git Bash

Final Outcome

After Day 2, you can:

- Use Git confidently from Git Bash
- Track and explain changes visually and logically
- Write professional commit messages
- Understand why Git history matters

Lab Day 3 – GitHub Remote Repository

Objective

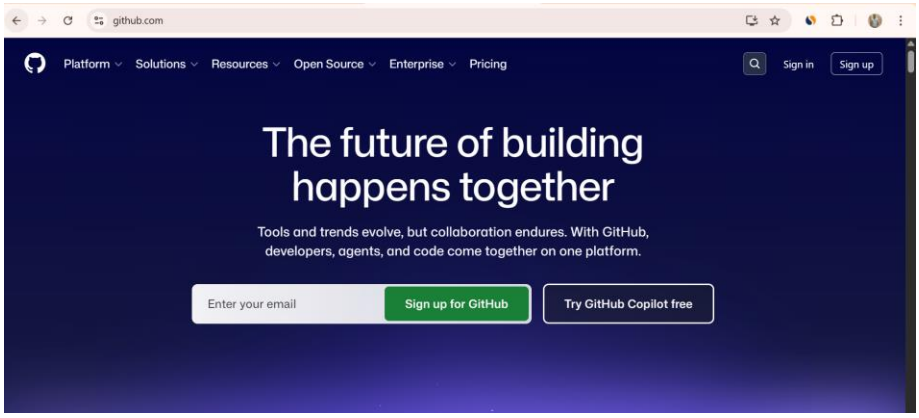
Upload the existing git-lab folder to GitHub.

Basic Theory

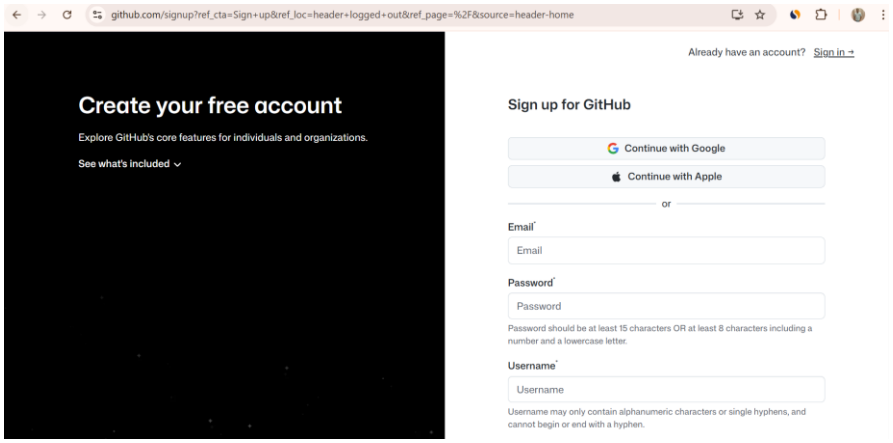
Git works on computer.
GitHub works on internet.
Both store the same files.

Step-by-Step Procedure (WITH SCREENSHOTS)

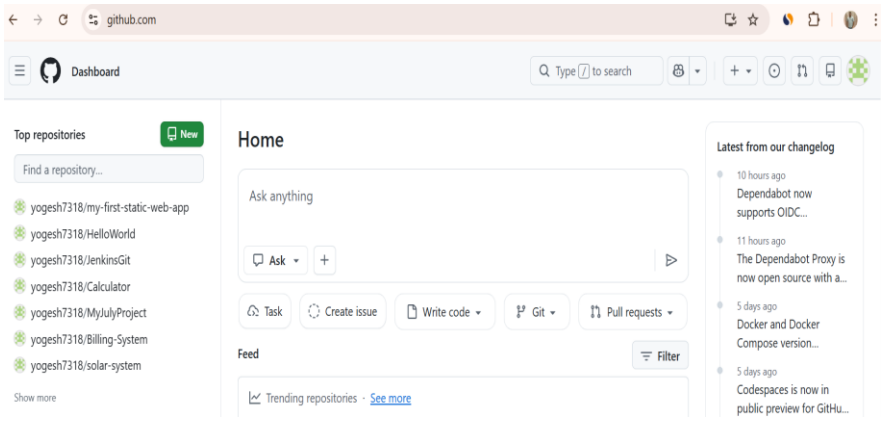
Step 1: Open Web Browser and Open GitHub Website



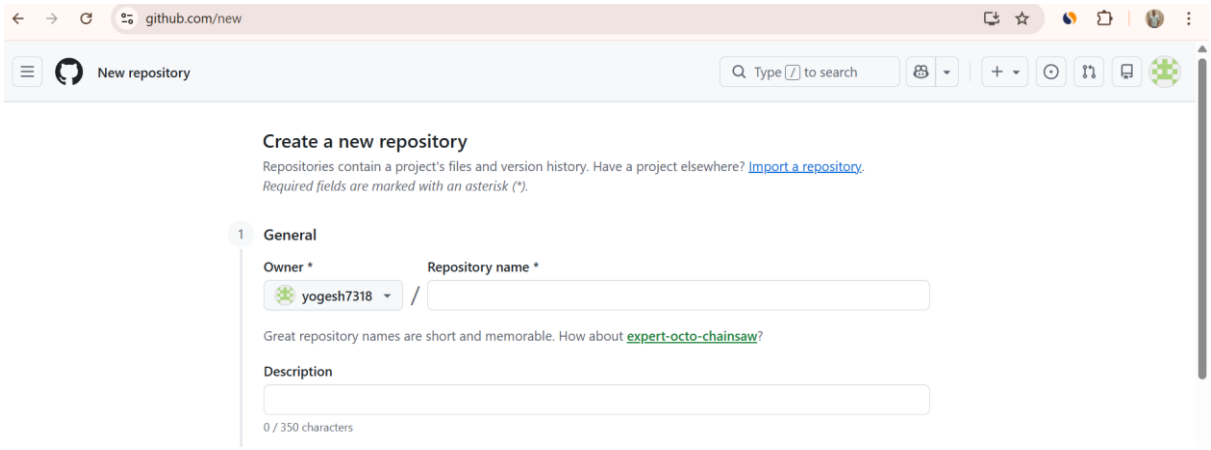
Step 2: Click Sign Up



Step 3: GitHub Dashboard After Login

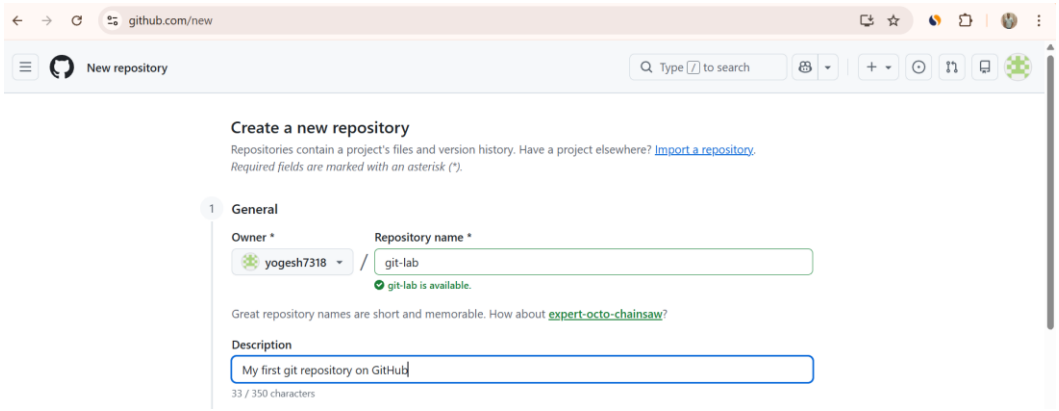


Step 4: Click New Icon → New Repository

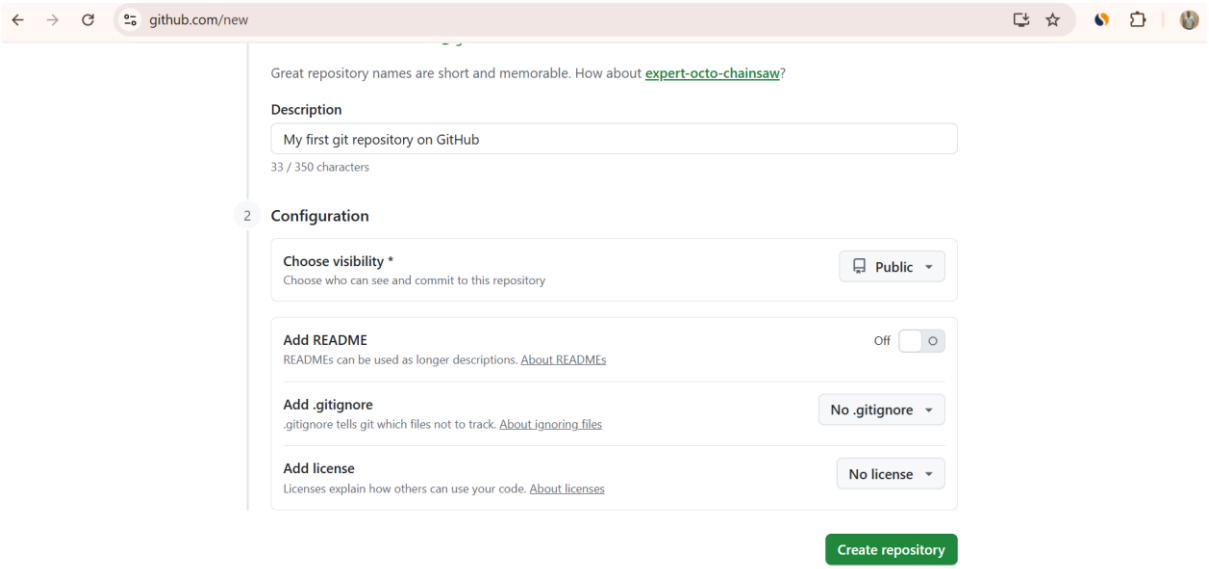


Step 5: Enter Repository Name

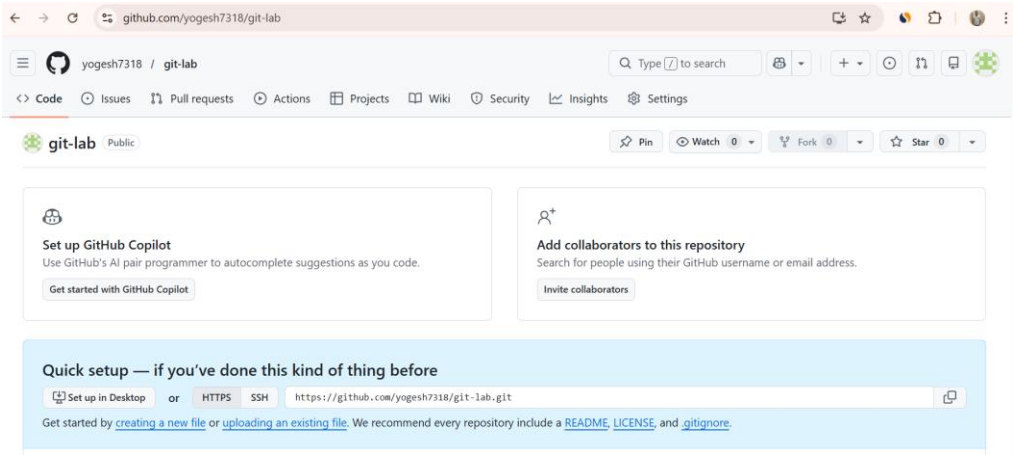
Type git-lab



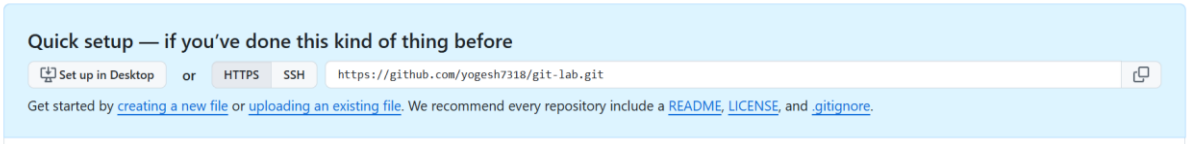
Step 6: Select Public Repository and Do NOT Select README



Step 7: Click Create Repository

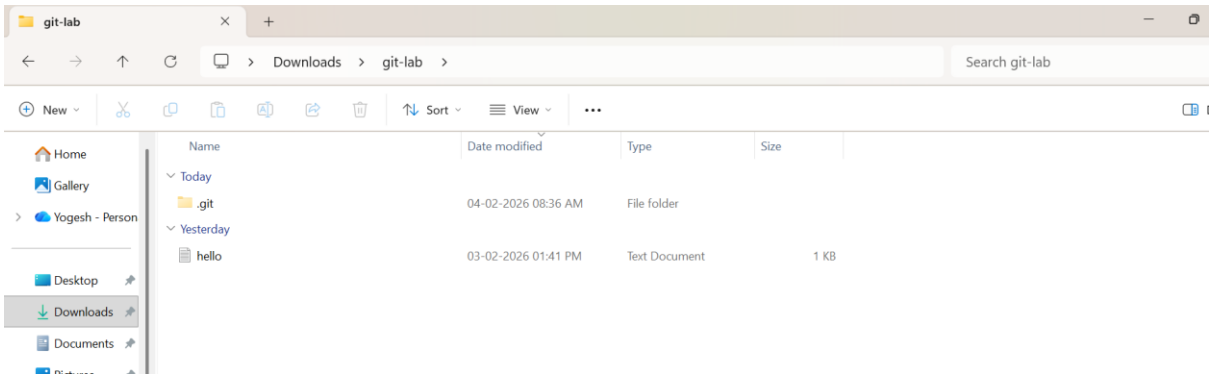


Step 8: Locate HTTPS Link and Copy Repository URL



Local Computer Steps

Step 9: Open git-lab Folder in File Explorer and Right click → Git Bash Here



Step 10: Set Remote Origin in Git Bash

git remote add origin https://github.com/yogesh7318/git-lab.git

- Used when no remote named origin exists.
- It adds a new remote connection to your repository.
- Usually done the first time you connect your local repo to GitHub.

👉 Think of it as *creating* the remote link.

git remote set-url origin https://github.com/yogesh7318/git-lab.git

- Used when origin already exists.
- It changes the existing remote URL.

👉 Think of it as *updating* the remote link.

git remote -v

git push -u origin main

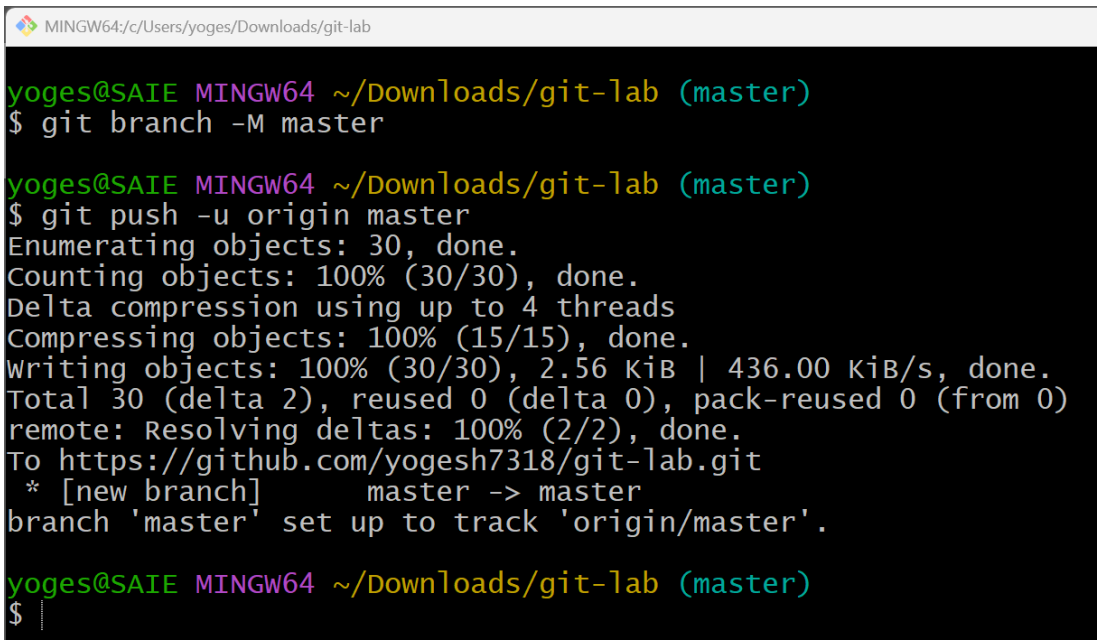
git push -u origin master

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git remote set-url origin https://github.com/yogesh7318/git-lab.git

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git remote -v
origin  https://github.com/yogesh7318/git-lab.git (fetch)
origin  https://github.com/yogesh7318/git-lab.git (push)

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

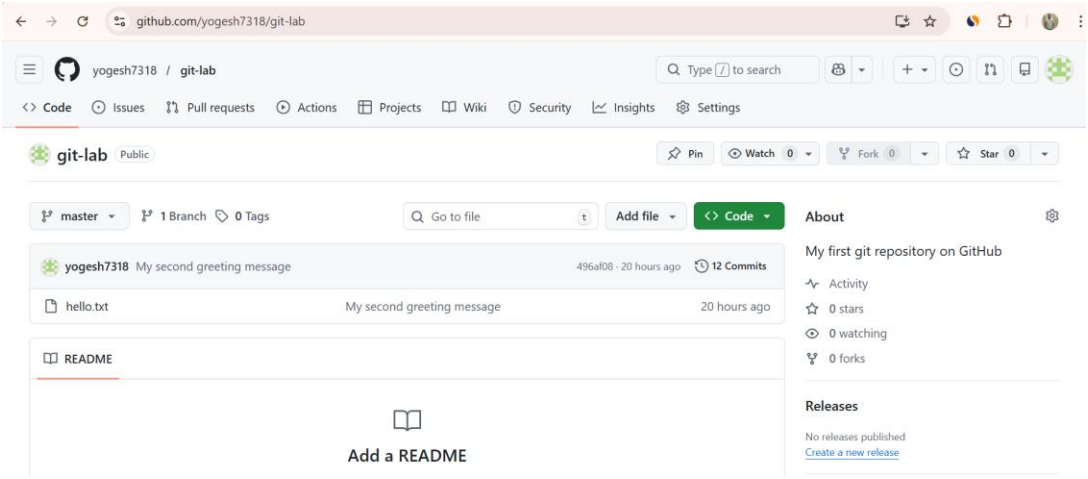
Step 11: Push Files to GitHub

git branch -M master

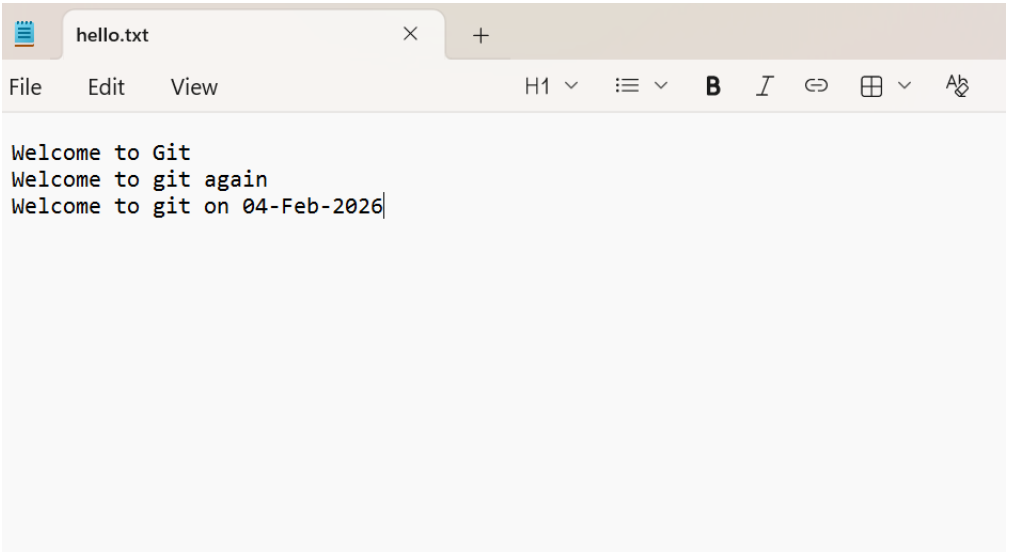
git push -u origin master



Step 12: Verify Files on GitHub



Step 13: Edit hello.txt in File Explorer



Step 14: Commit Change

git add hello.txt

git commit -m "File changed on 04-Feb-2026"

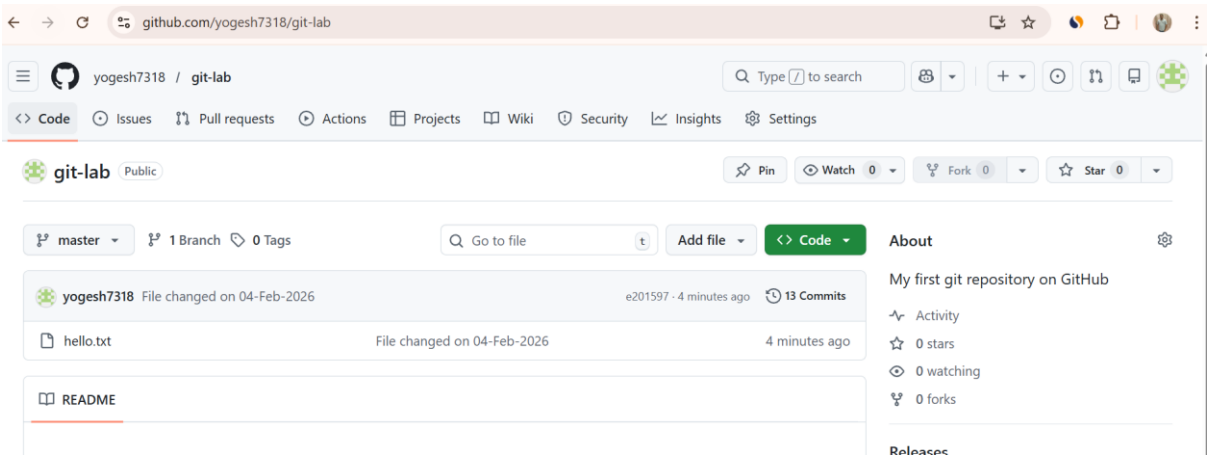
```
MINGW64:/c/Users/yoges/Downloads/git-lab

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git add hello.txt
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the
it

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git commit -m "File changed on 04-Feb-2026"
[master e201597] File changed on 04-Feb-2026
 1 file changed, 1 insertion(+)

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

Step 15: Push to GitHub

git push

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/yogesh7318/git-lab.git
   496af08..e201597  master -> master

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

Step 16: Verify your Commit on GitHub



Validation Checklist

- Repository visible on GitHub
- hello.txt visible
- Multiple commits visible

Industry Mapping

GitHub is used as a **central code storage** by companies so teams can work together.

End-of-Day Learning Outcome

Students can upload and update code on GitHub.
Students understand online code storage.

Lab Day 4 – Branching in GitHub

Objective

Create and use a branch so changes do not affect the main code.

Basic Theory

- The **main branch** is the original work.
- A **branch** is a copy of the work.
- We use branches to practice safely.

Real-life example:

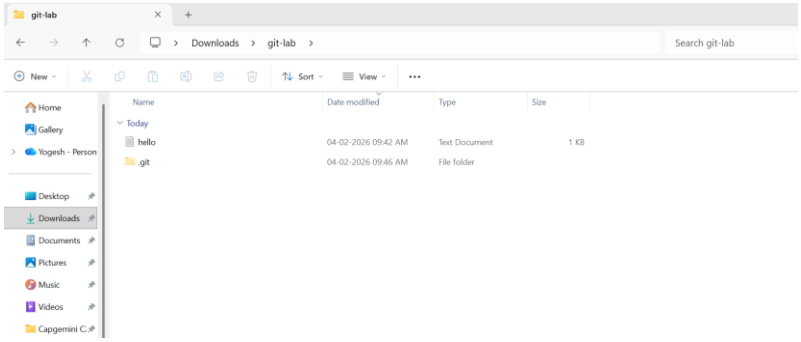
Write rough work on a separate page. Do not spoil the final notebook.

Before We Start (Confirm)

- GitHub repository already exists
- **git-lab** folder exists on computer
- **hello.txt** exists
- Code is already pushed to GitHub

Step-by-Step Procedure (With Screenshots)

Step 1: Open git-lab Folder in File Explorer and then Open Git Bash



Step 2: Check Current Branch in Git Bash

git branch

Step 3: Create a new branch and view all branches

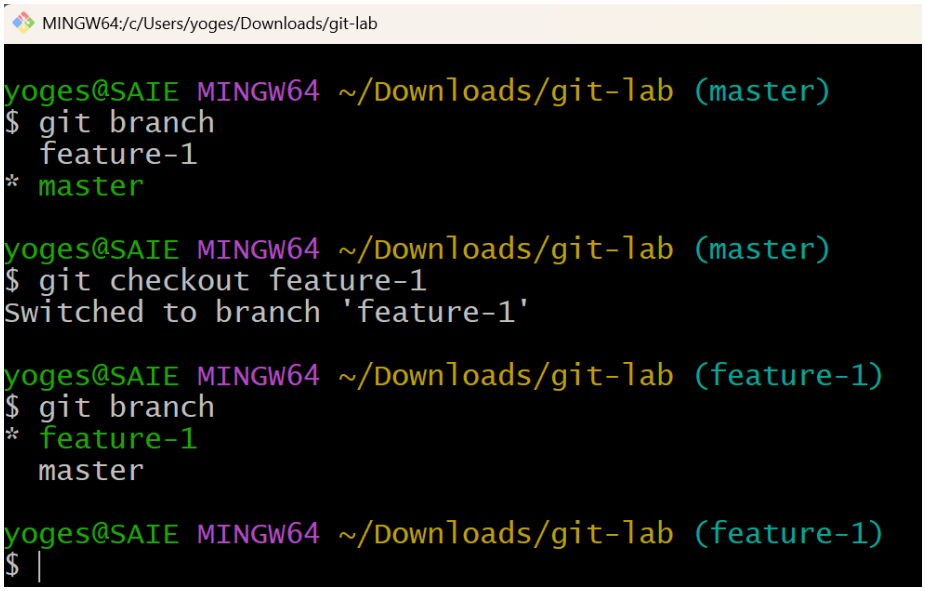
git branch feature-1

git branch

Step 4: Switch to feature-1 Branch and confirm branch is switched

git checkout feature-1

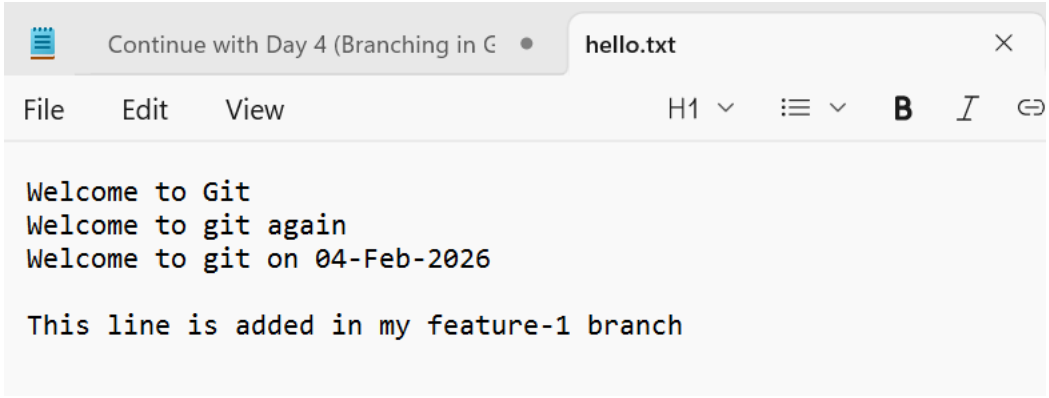
git branch



Step 5: Edit hello.txt File in File Explorer

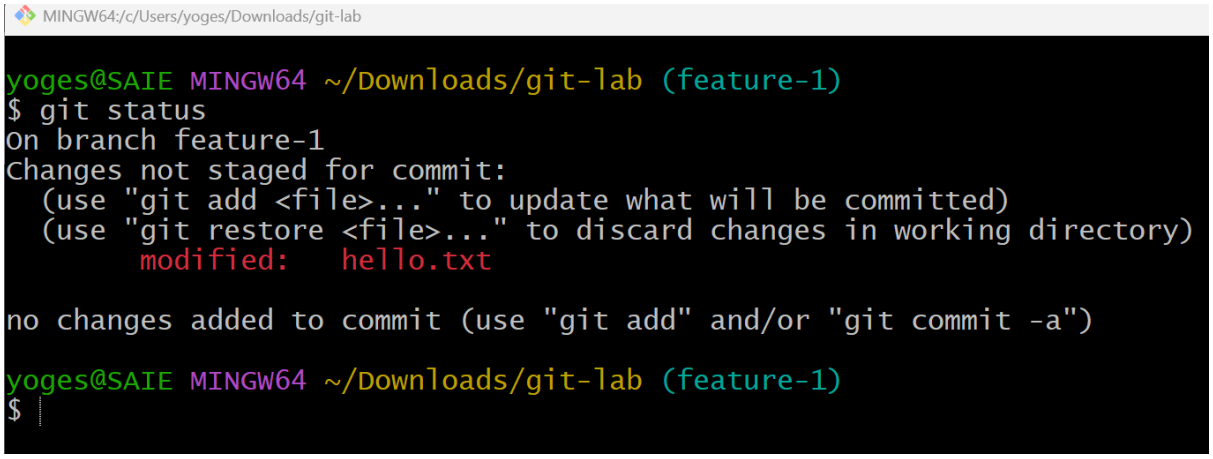
Steps:

1. Open **hello.txt**
2. Add one new line: "**This line is added in my feature-1 branch**"
3. Save the file



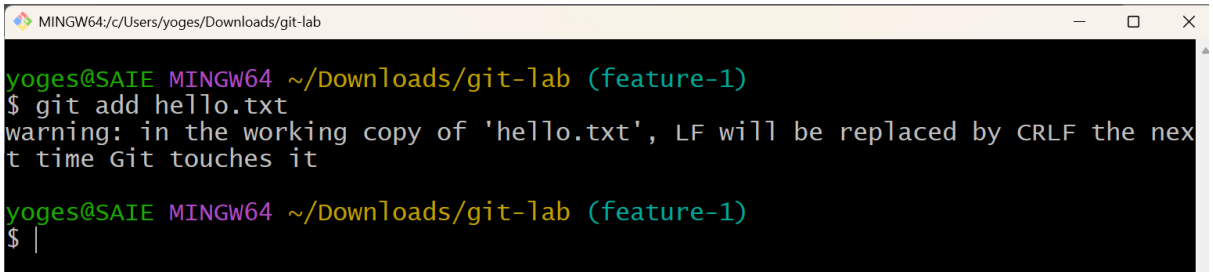
Step 6: Check File Status

git status



Step 7: Add File to Git

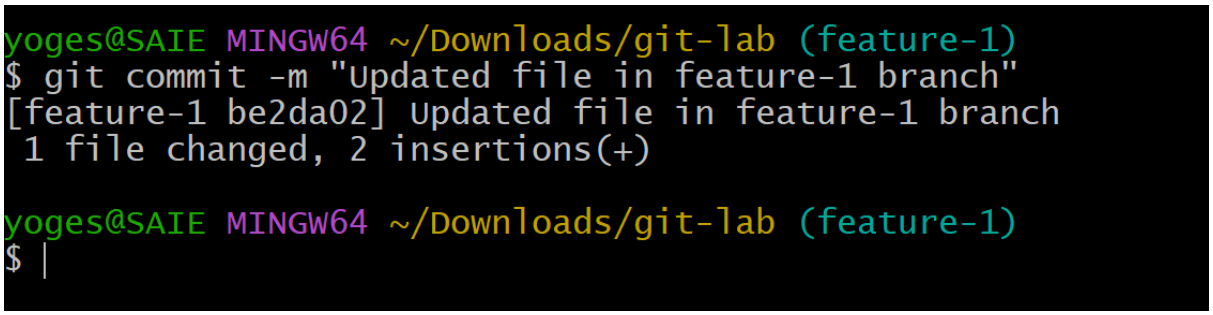
git add hello.txt



Step 11: Commit Change on Feature Branch

git commit -m "Updated file in feature-1 branch"

Change is saved only in feature branch



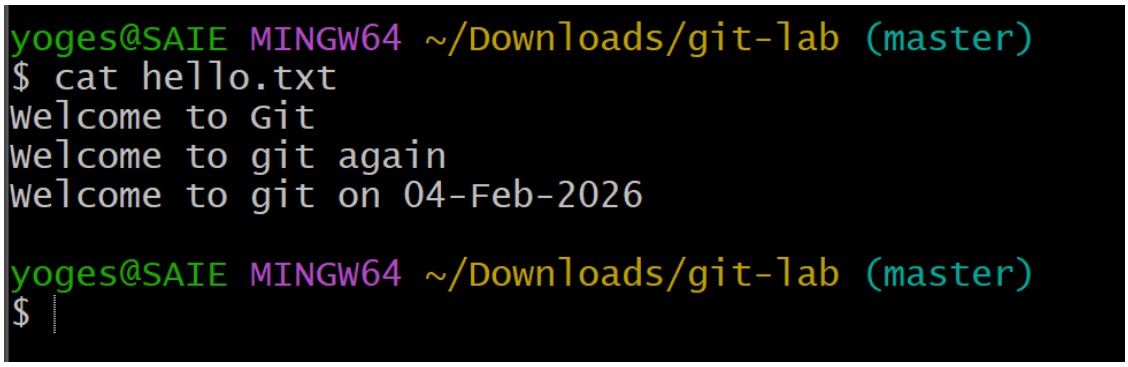
Step 12: Switch back to master branch

git checkout master

Step 13: Open hello.txt in master branch

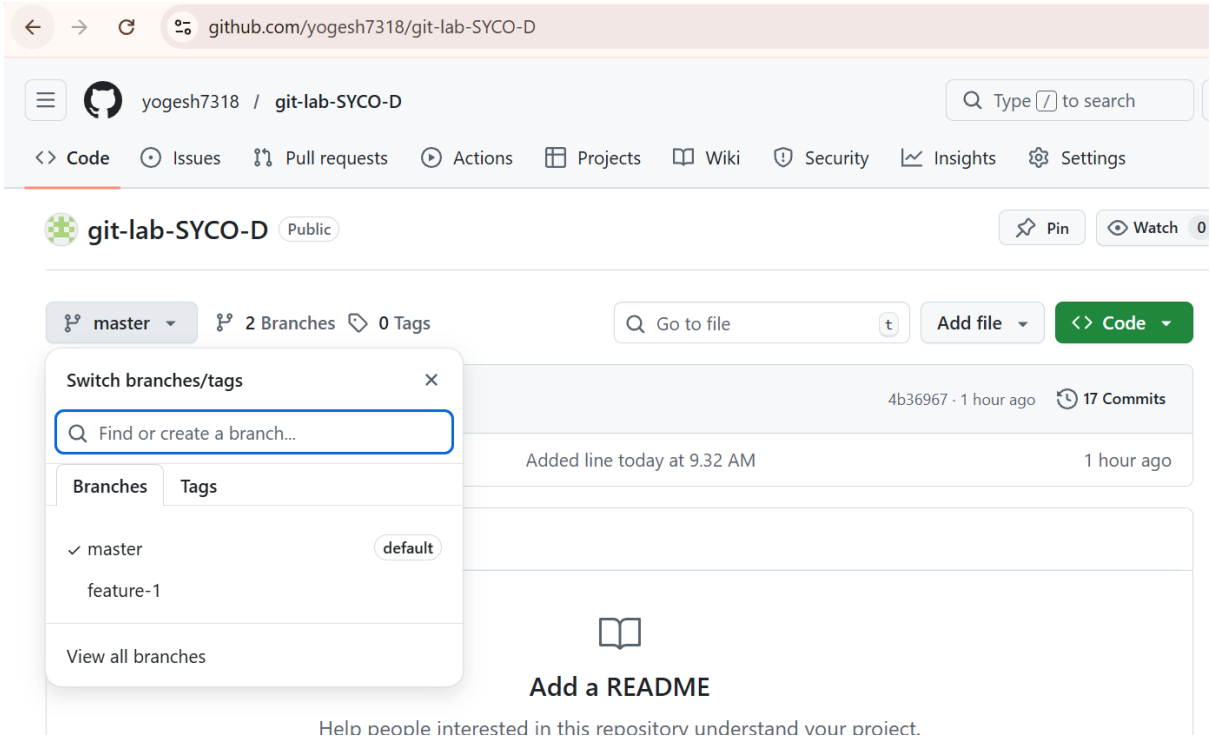
cat hello.txt

- Feature branch line is **NOT visible**
- Master branch is unchanged



Step 14: Open GitHub Repository and View Branch List on GitHub

git push --set-upstream origin feature-1



Validation Checklist

- feature-1 branch created
- Branch visible using git branch
- Feature branch has extra line
- Master branch file unchanged

Lab Day 5 – Merging & Conflict Resolution

Objective

Understand **why merge conflicts happen** and learn **how to resolve them safely**, the same way real development teams do.

Prerequisites

Before starting, confirm:

- Git is installed and working
- Repository already exists
- Main/master branch exists
- feature-1 branch exists
- File hello.txt exists

Step-by-Step Tasks

Step 1: Switch to main branch

git checkout master

You are now working on the **master branch**.

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (feature-1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git branch
  feature-1
* master

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

Step 2: Modify file on main

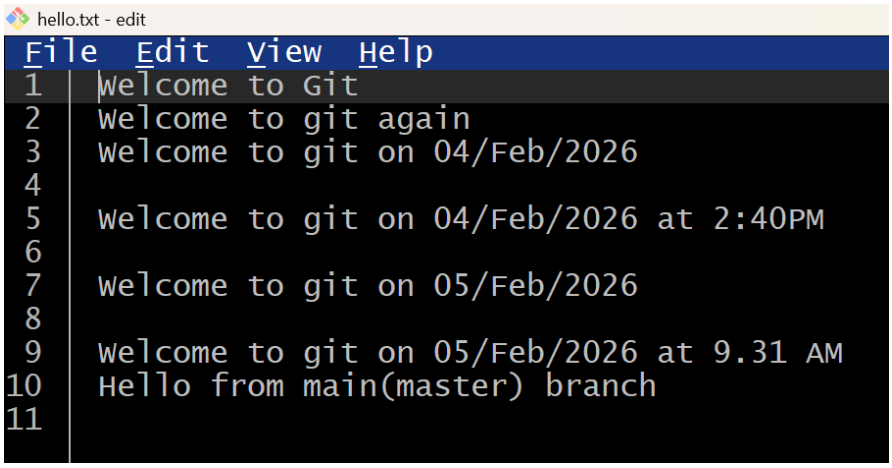
- Open hello.txt

- Change **one existing line**

Example:

Hello from main (master) branch

Save the file.



Step 3: Commit change on main

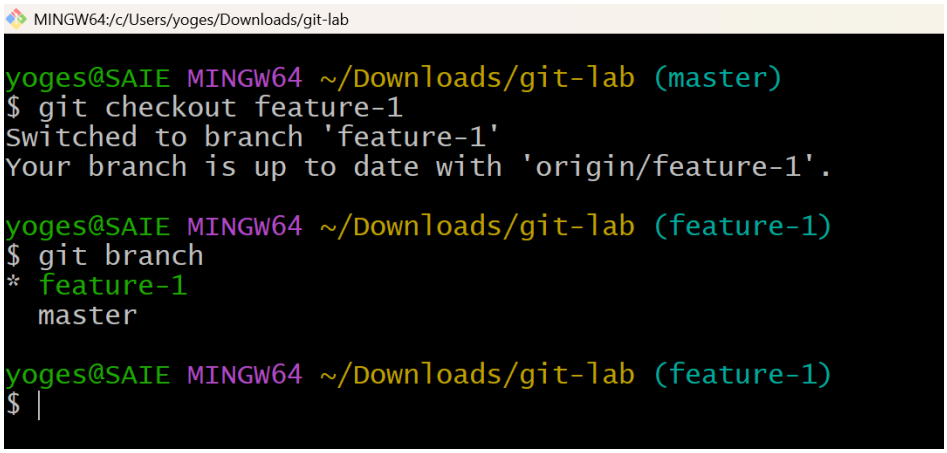
git add hello.txt

git commit -m "Update greeting from main branch"

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git add hello.txt

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git commit -m "Update greeting from main branch"
[master c123059] Update greeting from main branch
 1 file changed, 1 insertion(+)

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

Step 4: Switch to feature-1 branch

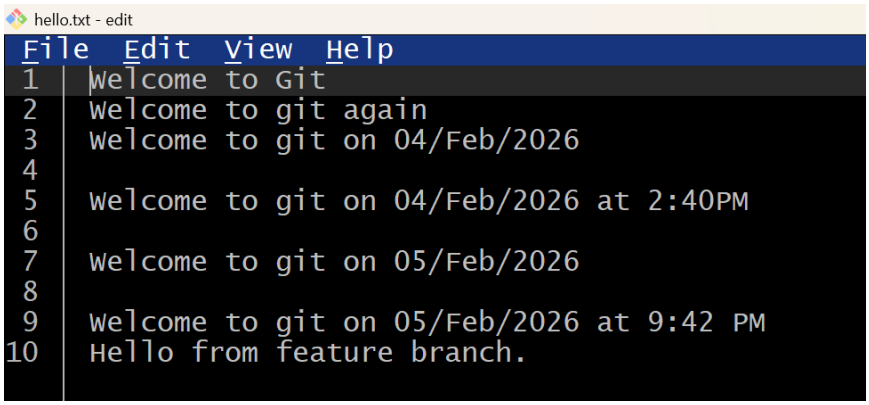
git checkout feature-1



Step 5: Modify SAME line differently

- Open hello.txt
- Change **the same line** to something else

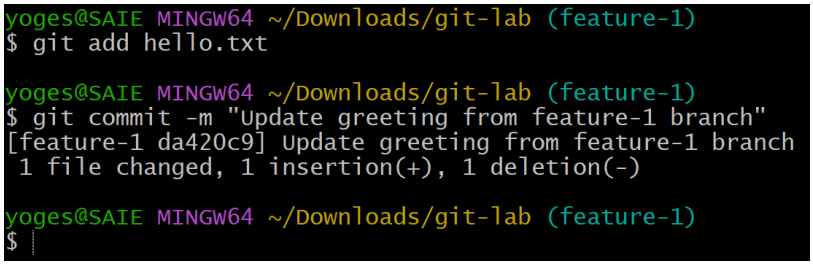
Example: Hello from feature branch. Save the file.



Step 6: Commit change on feature-1

git add hello.txt

git commit -m "Update greeting from feature-1 branch"



Step 7: Switch back to main

git checkout master

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (feature-1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$
```

Step 8: Attempt merge

git merge feature-1

Git **stops the merge** because the same line was changed differently.

MINGW64:/c/Users/yoges/Downloads/git-lab

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master)
$ git merge feature-1
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.

yoges@SAIE MINGW64 ~/Downloads/git-lab (master|MERGING)
$
```

Understanding Conflict Markers

cat hello.txt

```
yoges@SAIE MINGW64 ~/Downloads/git-lab (master|MERGING)
$ cat hello.txt
welcome to Git
Welcome to git again
Welcome to git on 04/Feb/2026

Welcome to git on 04/Feb/2026 at 2:40PM

Welcome to git on 05/Feb/2026

<<<<<<< HEAD
welcome to git on 05/Feb/2026 at 9.31 AM
Hello from main(master) branch
=======
Welcome to git on 05/Feb/2026 at 9:42 PM
Hello from feature branch.
>>>>>>> feature-1

yoges@SAIE MINGW64 ~/Downloads/git-lab (master|MERGING)
$
```

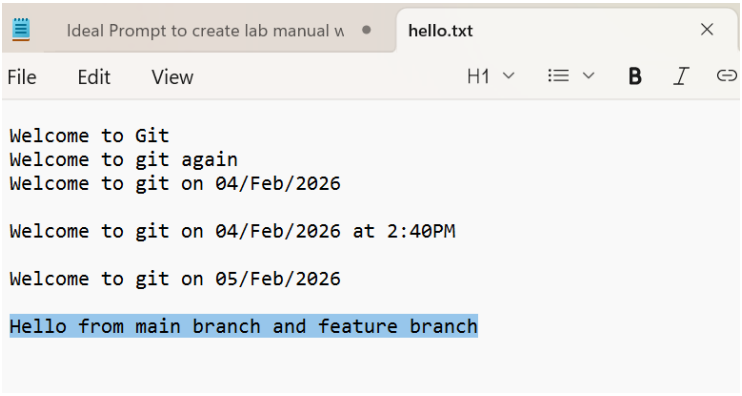
Resolve Conflict Manually

Step 9: Fix the file

- Remove **ALL conflict markers**
- Keep final correct line only

Example final content:

Hello from main branch and feature branch. Save the file.



Step 10: Mark conflict resolved and Complete merge

git add hello.txt

git commit -m “Added lines in main and feature branch”

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master|MERGING)
$ git add hello.txt

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master|MERGING)
$ git commit -m "Added lines in main and feature branch"
[master 89e4f82] Added lines in main and feature branch

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

Step 12: Verify merge history

git log --oneline

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
89e4f82 (HEAD -> master) Added lines in main and feature branch
60ff012 (feature-1) Updated greeting from my feature-1 branch
```

Step 13: Check status

git status

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

Validation Checklist – Day 5

- No conflict markers remain
- Merge commit exists
- git status is clean

Common Mistakes – Day 5

- Leaving conflict markers in file
- Committing without fixing

Lab Day 6 – Push, Pull & Sync

---

1. Lab Overview

- Till now, you worked mostly on one computer.
- In real teams, code moves between many computers.
- Push and pull help everyone stay updated.

---

2. Pre-Lab Setup (very important to complete all steps until Day 5)

Check current branch

git branch (You should see your feature branch selected)

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git branch
* feature-1
  feature-2
  master
```

Check working tree is clean *(If files are modified, commit them before proceeding)*

git status

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git status
On branch feature-1
Your branch is ahead of 'origin/feature-1' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

Verify GitHub remote is linked

git remote -v

```
MINGW64:/c/Users/yoges/Downloads/git-lab-SYCO-B

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git remote -v
origin  https://github.com/yogesh7318/git-lab-SYCO-B1.git (fetch)
origin  https://github.com/yogesh7318/git-lab-SYCO-B1.git (push)

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

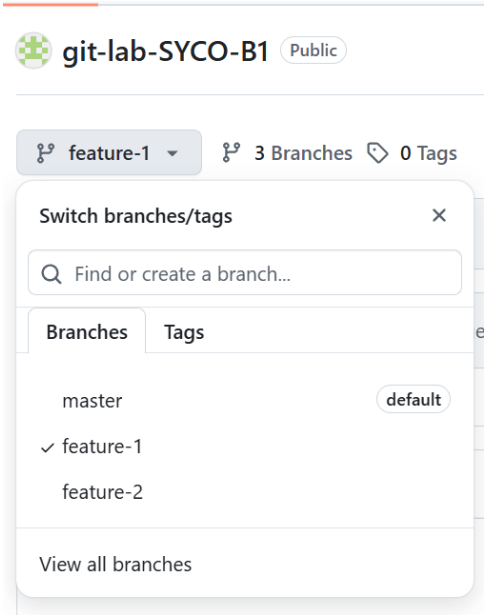
Push Feature Branch to GitHub and Verify Branch on GitHub

Command

git push -u origin feature-1

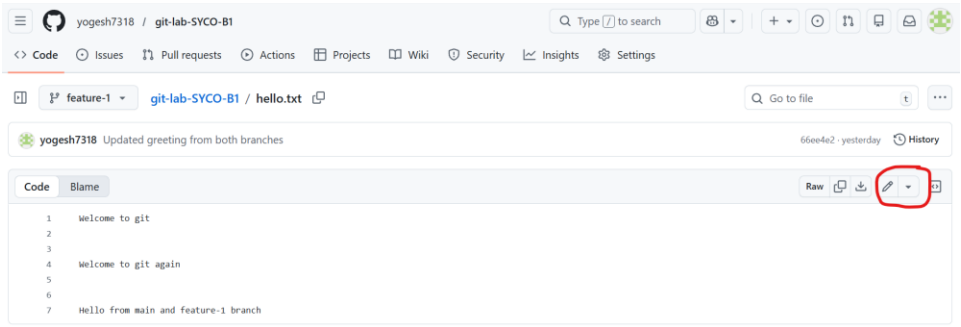
What you should observe:-

- Files upload to GitHub
- Push completes successfully
- Feature branch appears on GitHub



Step 2: Edit File Using GitHub.com

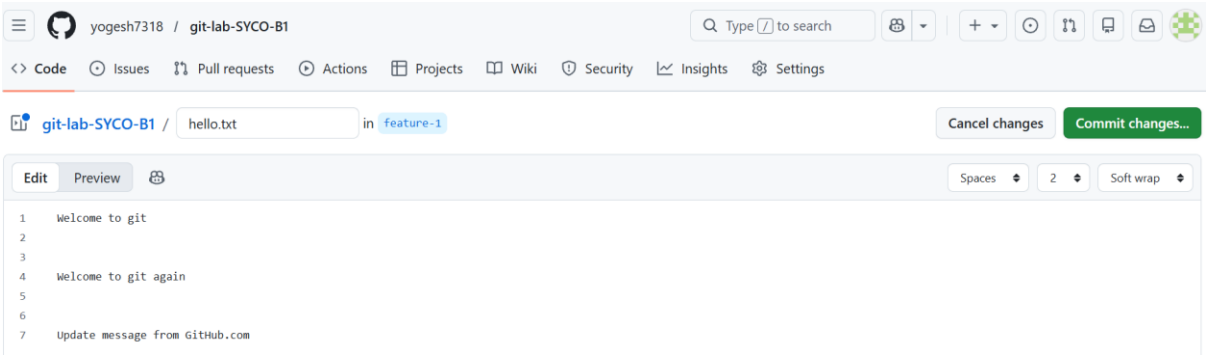
1. Open the same file on GitHub.com
2. Click **Edit (pencil icon)**
3. Change the same line edited locally

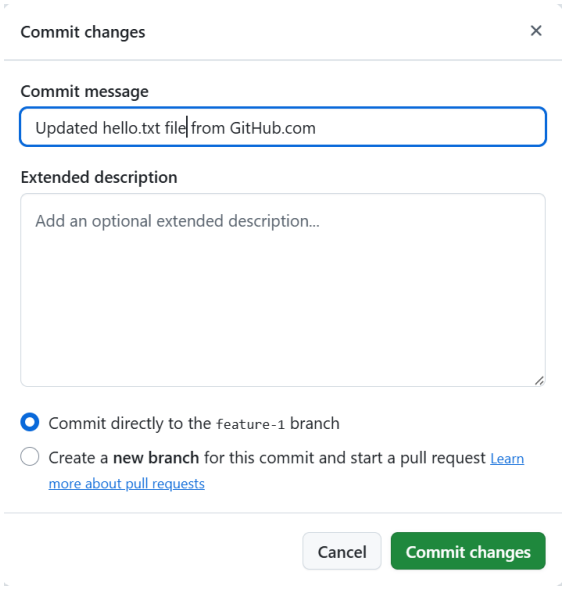


Commit Using Browser

1. Scroll down
2. Enter commit message
3. Click **Commit changes**

Example message: ***Update message from GitHub.com***





GitHub has a new commit. But your local machine does not know this yet.

---

5. Pull Changes from GitHub.com to your Local Machine

Step 4: Pull Latest Changes

git pull origin feature-1

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git pull origin feature-1
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (5/5), 2.73 KiB | 155.00 KiB/s, done.
From https://github.com/yogesh7318/git-lab-SYCO-B1
 * branch            feature-1  -> FETCH_HEAD
   13797d0..c0eb718  feature-1  -> origin/feature-1
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master|MERGING)
$ git add hello.txt

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master|MERGING)
$ git commit -m "Resolve conflict after merging feature-1 to master"
[master b9cb3cc] Resolve conflict after merging feature-1 to master

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git pull origin feature-1
From https://github.com/yogesh7318/git-lab-SYCO-B1
 * branch            feature-1  -> FETCH_HEAD
Already up to date.
```

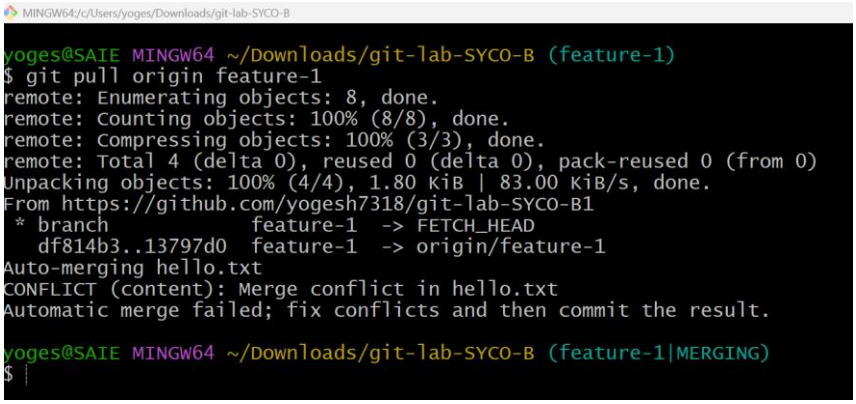
Case A: No Conflict

- Files update automatically
- Pull completes successfully

You may proceed to validation.

Case B: Conflict Occurs

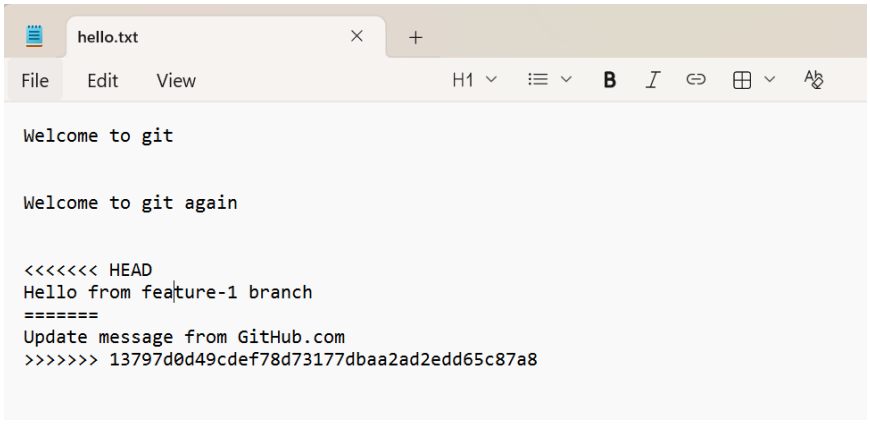
Git stops and shows a conflict message.

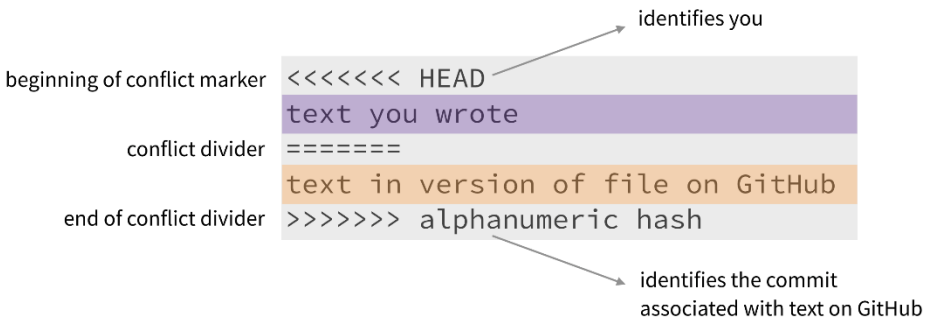


---

6. Conflict Scenario (If it happens)

Open the conflicted file





How to decide what to keep

- Read both versions
- Choose the correct one
- Remove all conflict markers

Fix and Complete the Merge

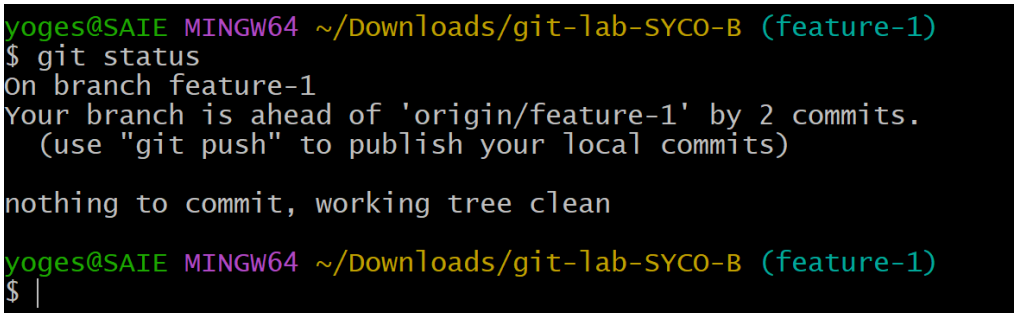
git add hello.txt

git commit -m "Resolve conflict after GitHub pull"

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1|MERGING)
$ git add hello.txt

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1|MERGING)
$ git commit -m "Resolve conflict after GitHub pull"
[feature-1 486bf65] Resolve conflict after GitHub pull

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

Final Status Check

git status



---

7. Validation Checklist

✓ Feature branch pushed to GitHub
✓ Browser edits pulled locally
✓ No conflict markers left
✓ git status is clean
✓ Local and remote code match

End of Lab Day 6

Students can now:

- Push work safely
- Pull others' changes
- Handle merge conflicts calmly

Lab Day 7 – Pull Requests & CI Introduction

---

1. Lab Overview

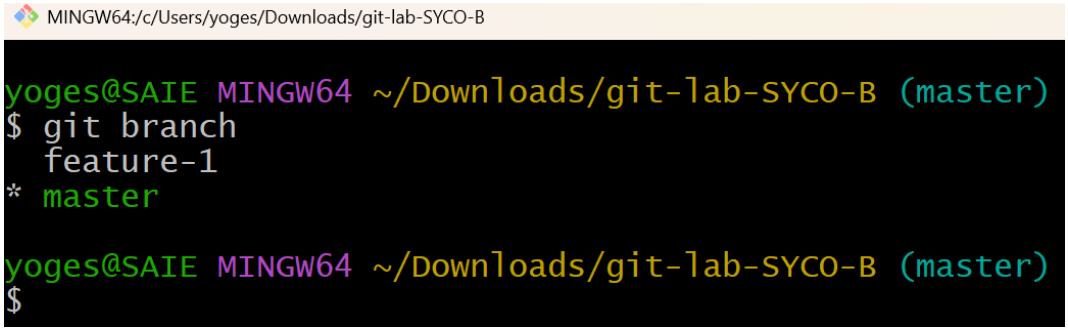
- Till now, you pushed code directly between systems.
- In real teams, code is merged only after review.
- Pull Requests and CI help teams avoid mistakes.

---

2. Pre-Lab Setup (very important to complete all steps until Day 6)

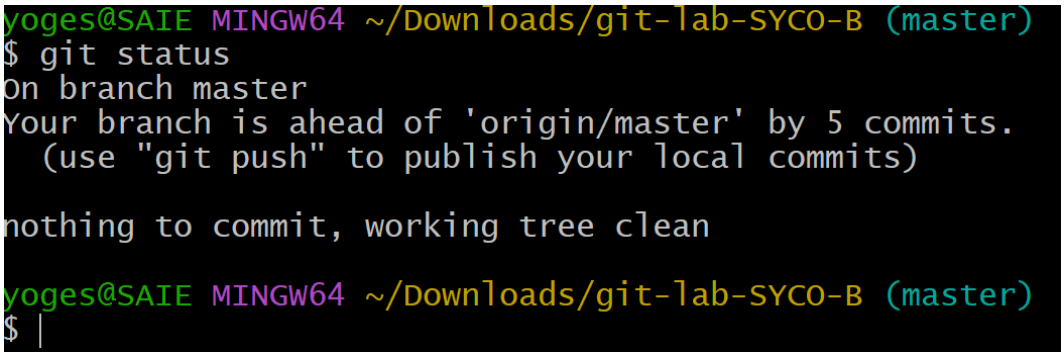
Step 1: Check current branch

git branch



Step 2: Check working tree is clean

git status



Step 3: Confirm feature branch is pushed

git push

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git push
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.45 KiB | 370.00 KiB/s, done.
Total 15 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/yogesh7318/git-lab-SYCO-B1.git
   b9cb3cc..89e4f82  master -> master

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

---

3. Step-by-Step Hands-on Instructions

Step 1: Push Latest Feature Branch

git push -u origin feature-1

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git push -u origin feature-1
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/yogesh7318/git-lab-SYCO-B1.git
   c0eb718..60ff012  feature-1 -> feature-1
branch 'feature-1' set up to track 'origin/feature-1'.

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

---

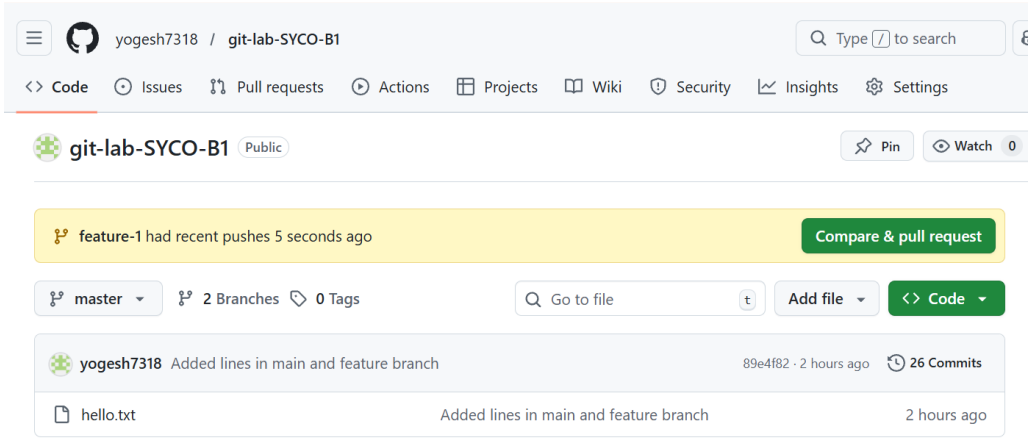
4. Pull Request Section (Very Important)

What is a Pull Request?

A Pull Request means: **“I want permission to merge my code into master/main branch.”**

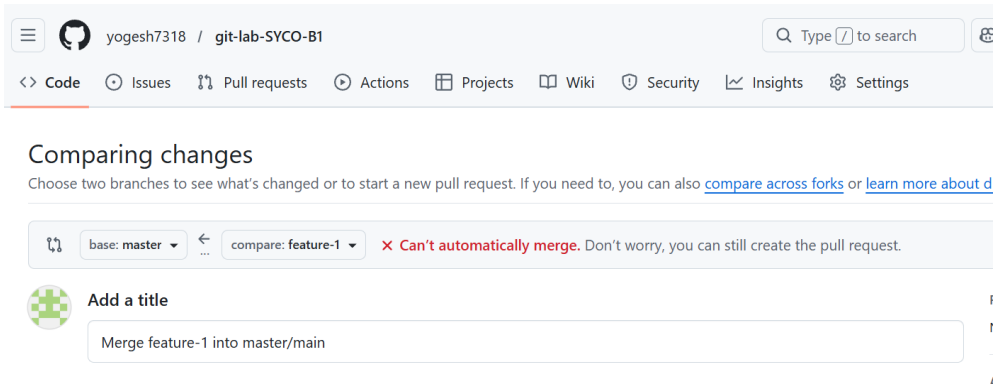
Step 1: Create Pull Request on GitHub

1. Open your GitHub repository
2. Click **Compare & pull request**



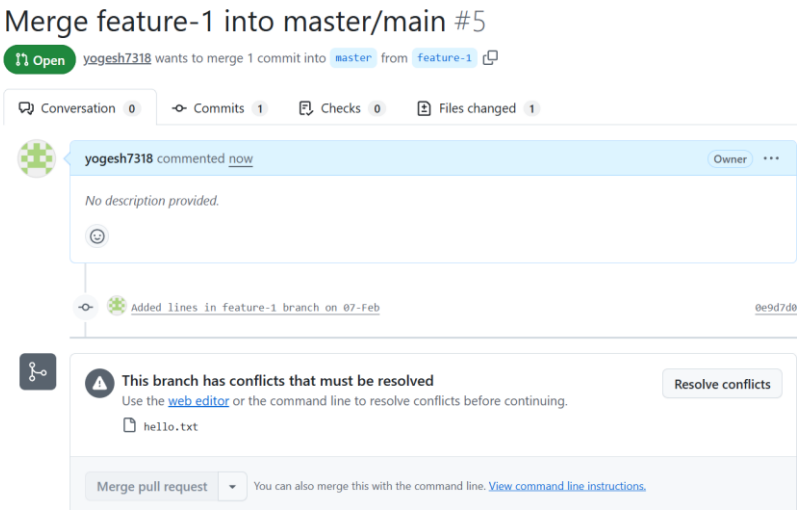
Select branches carefully

- **Base branch:** master/main
- **Compare branch:** feature-1



Create Pull Request

- Title: **Merge feature-1 into master/main**
- Click **Create pull request**



Common mistakes to avoid:

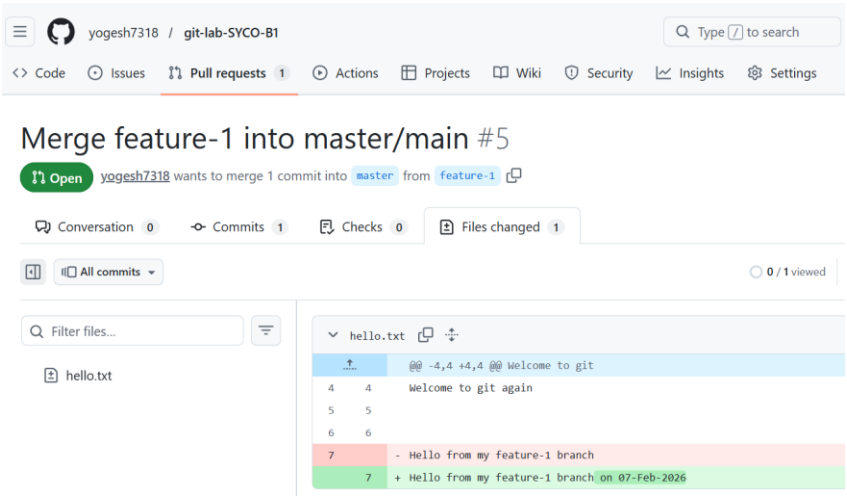
- Selecting wrong base branch
- Creating PR from main to main

---

5. Review & Merge Section

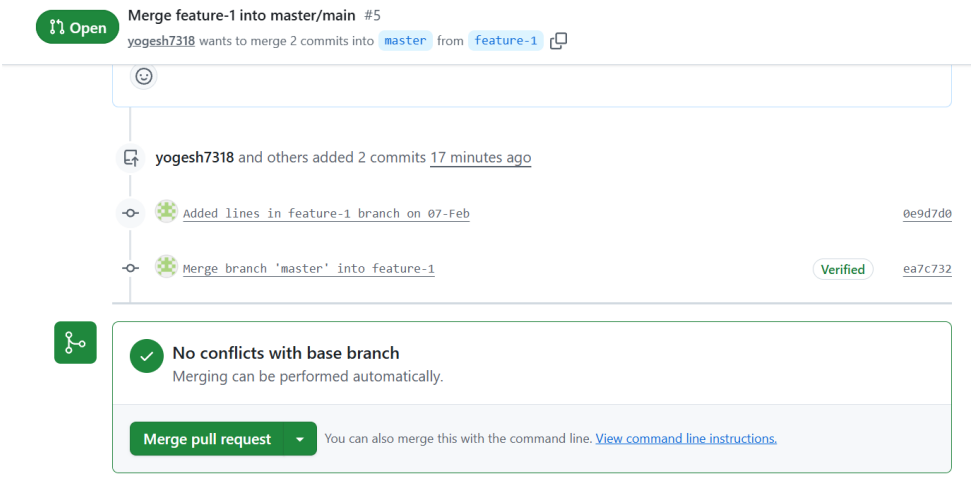
Step 1: Review the Pull Request

- Scroll through changed files
- Read changes carefully

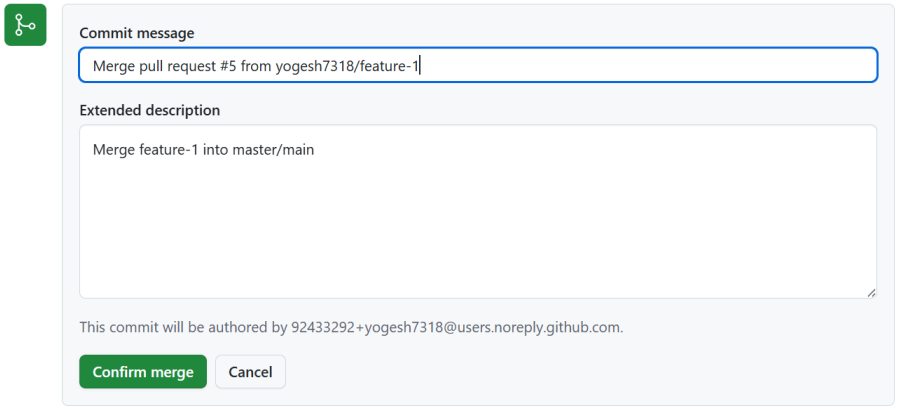


Step 2: Merge Pull Request (GitHub Only)

1. Click **Merge pull request**



2. Click **Confirm merge**



What happens after merge

- Code from feature-1 moves into main
- Feature branch can be deleted
- Main branch is updated safely

---

6. CI Introduction Section

What is CI (Continuous Integration) ?

CI means: **“Automatic check after every push.”**

No human runs it. GitHub runs it.

Step 1: Create CI Workflow Folder and File

Create folder:

mkdir -p .github/workflows

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ pwd
/c/Users/yoges/Downloads/git-lab-SYCO-B

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ mkdir -p .github/workflows

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ ls -a
./  ../  .git/  .github/  hello.txt

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

Create file:

touch .github/workflows/ci.yml

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ touch .github/workflows/ci.yml

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

Add the below content in ci.yml

```
name: First CI Job


on: [push]


jobs:
  simple-job:
    runs-on: ubuntu-latest
    steps:
      - name: Print message
        run: echo "CI is running successfully"
```

ci.yml

File Edit View

```yaml
name: First CI Job

on: [push]

jobs:
  simple-job:
    runs-on: ubuntu-latest
    steps:
      - name: Print message
        run: echo "CI is running successfully"
```

---

7. Trigger CI

Step 1: Push CI Workflow

git status

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git status
On branch feature-1
Your branch is up to date with 'origin/feature-1'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .github/

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

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

git add .github/workflows/ci.yml

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git add .github/workflows/ci.yml

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

git commit -m "Add first CI workflow"

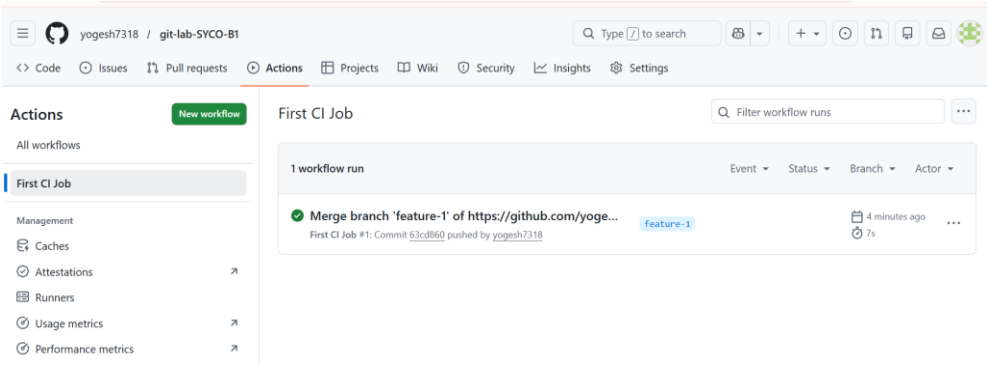
```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git commit -m "Add first CI workflow"
[feature-1 0faf622] Add first CI workflow
 1 file changed, 10 insertions(+)
 create mode 100644 .github/workflows/ci.yml

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

git push

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 812 bytes | 270.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/yogesh7318/git-lab-SYCO-B1.git
   62143bf..63cd860  feature-1 -> feature-1

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (feature-1)
$
```

What you should observe

1. Open **Actions** tab on GitHub
2. CI job starts automatically
3. Green check appears after success



Meaning of green check

✓ Code passed basic checks
✓ Safe to continue work

✕ If red cross appears:
Read logs. Fix mistake. Push again.

---

8. Validation Checklist

✓ Pull Request created
✓ Pull Request merged via GitHub
✓ Feature branch merged into main

✓ CI workflow file present
✓ CI ran automatically
✓ Green check visible in Actions tab

---

9. Industry Mapping (Very Practical)

- **Code reviews**
  Pull Requests allow teams to check code.
- **Preventing bad code in main**
  No direct push to main.
- **Release safety**
  CI catches mistakes early.
- **Why main is protected**
  Main goes to production.

End of Lab Day 7

You have completed the **full Git workflow**:

Feature → Push → Pull Request → Merge → CI

This is how real teams work.

Lab Day 8 – Commit Recovery & Undo Techniques

---

1. Lab Overview

In real coding, mistakes happen every day.
Files are changed by mistake. Wrong commits are made.
Git is powerful because **it remembers everything** and allows safe recovery.

Today's lab will teach you **how to undo mistakes without fear**.

---

2. Pre-Lab Setup

Before starting, confirm these conditions.

Step 1: Check current branch

git branch ***You should see main (or master)***

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git branch
  feature-1
* master

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

Step 2: Ensure clean working tree

git status ***You should see working tree clean***

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 8 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

Step 3: Ensure multiple commits exist

git log --oneline ***You should see at least 3 commits***

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
da2a676 (HEAD -> master) added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

---

3. Important Reminder Before We Begin

- Making mistakes is normal
- Git **never deletes history immediately**
- Recovery is possible **if you stay calm**

---

4. Viewing Commit History

Step 1: View full commit history

git log

What you observe

- Commit ID (long code)
- Author name
- Date
- Commit message

What Git does internally

Git stores every commit like a **snapshot in time**.

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log
commit da2a676f61774ef1966f693947e03fb421a586e6 (HEAD -> master)
Author: Yogesh K <yogesh7318@gmail.com>
Date:   Mon Feb 9 11:25:33 2026 +0530

    added file

commit 202b7e654f59956b24934b347a6da385171488c2
Merge: 2b01615 26c34bd
Author: Yogesh K <yogesh7318@gmail.com>
Date:   Mon Feb 9 09:29:49 2026 +0530

    Added file to git on 09-Feb-2026

commit 26c34bdeb21c605a9a59ecfda488cfc6a72f7542
Author: Yogesh K <yogesh7318@gmail.com>
Date:   Mon Feb 9 09:27:53 2026 +0530

    Added file to git on 09-Feb-2026
```

Step 4.2: View short commit history

git log --oneline

What you observe

- Short commit ID
- One-line message

Simple explanation

Commit ID is like a **roll number** for each saved change.

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
da2a676 (HEAD -> master) added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

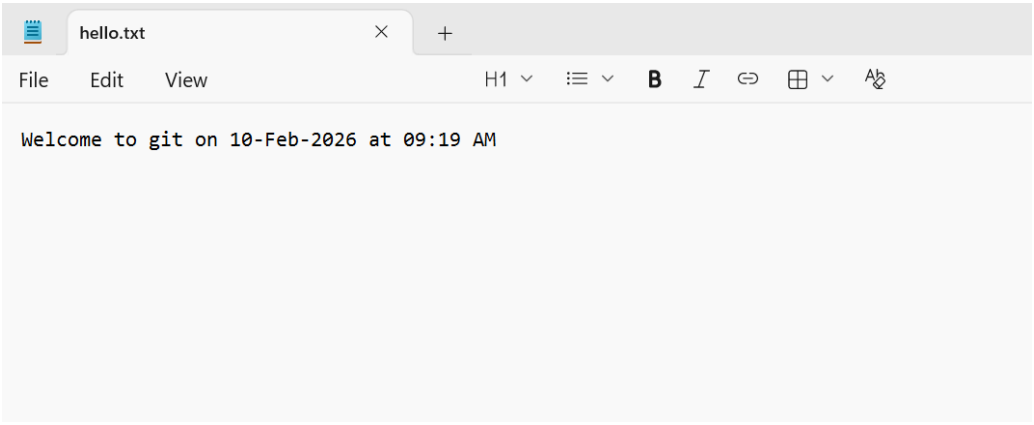
---

5. Undo Changes Before Commit

Situation

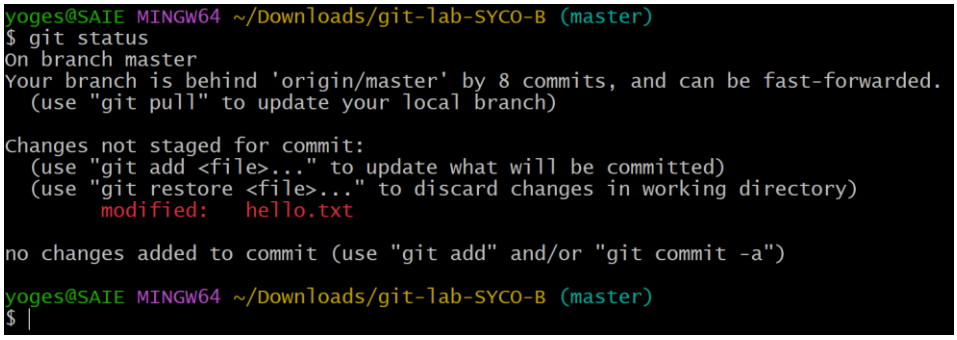
You edited a file but **did not commit** yet.

Step 1: Modify a file and save the file.



Step 5.2: Check status

git status



Step 5.3: Undo file changes

git restore hello.txt

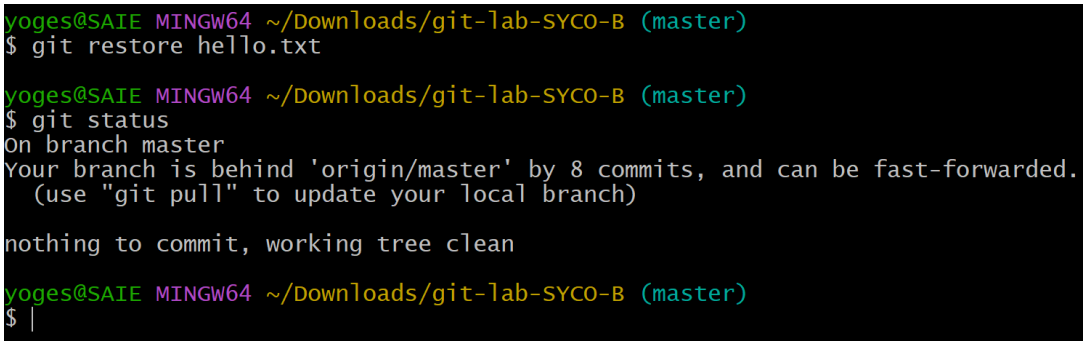
git status

What you observe

- File returns to last committed version
- No changes remain

What Git does internally

Git replaces the file using the **last saved snapshot**.



---

6. Undo Commits Without Losing Work (Soft Reset)

Situation

You committed too early.
You want the files back but **not lose changes**.

Step 1: Make a new commit and then check commit history

git commit -m "Wrong commit"

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git commit -m "Wrong commit"
[master c41f99a] Wrong commit
 1 file changed, 1 insertion(+), 1 deletion(-)

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

git log –-oneline

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
c41f99a (HEAD -> master) Wrong commit
da2a676 added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

Step 2: Soft reset

git reset --soft HEAD~1

git log --oneline

git status

What you observe

- Commit disappears from log
- Files are still staged

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git reset --soft HEAD~1

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
da2a676 (HEAD -> master) added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 8 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.txt


yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
```

---

7. Hard Reset

Hard reset **permanently removes changes**.

Step 1: Create a commit

git commit -m "Temporary commit"

git log --oneline

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git commit -m "Temporary commit"
[master e5b60a7] Temporary commit
 1 file changed, 1 insertion(+), 1 deletion(-)

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
e5b60a7 (HEAD -> master) Temporary commit
da2a676 added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

Step 2: Hard reset

git reset --hard HEAD~1

git log --oneline

git status

What you observe

- Commit removed
- File changes gone

What is deleted

- Files
- Staging
- Commit

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git reset --hard HEAD~1
HEAD is now at da2a676 added file

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
da2a676 (HEAD -> master) added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 8 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$
```

When NOT to use hard reset

- On shared branch
- On pushed commits
- When unsure

---

8. Recovering Commits Using git reflog

Explanation

git reflog is **Git's hidden diary**. It remembers everything, even deleted commits.

Step 1: View reflog

git reflog

What you observe

- List of actions
- Old commit IDs

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git reflog
da2a676 (HEAD -> master) HEAD@{0}: reset: moving to HEAD~1
e5b60a7 HEAD@{1}: commit: Temporary commit
da2a676 (HEAD -> master) HEAD@{2}: reset: moving to HEAD~1
c41f99a HEAD@{3}: commit: Wrong commit
da2a676 (HEAD -> master) HEAD@{4}: checkout: moving from feature-1 to master
c5990f4 (origin/feature-1, feature-1) HEAD@{5}: commit: file updated on 10-Feb-2026 at 08:31 AM
```

Step 2: Recover deleted commit

git reset --hard e5b60a7

git log --oneline

Result

Deleted commit is back

What Git does

Git moves HEAD back to that saved point.

```
yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git reset --hard e5b60a7
HEAD is now at e5b60a7 Temporary commit

yoges@SAIE MINGW64 ~/Downloads/git-lab-SYCO-B (master)
$ git log --oneline
e5b60a7 (HEAD -> master) Temporary commit
da2a676 added file
202b7e6 Added file to git on 09-Feb-2026
26c34bd Added file to git on 09-Feb-2026
```

---

9. Understanding HEAD Movement

Simple meaning

HEAD means **where Git is currently pointing.**

HEAD movement examples

Action	**HEAD moves**
Commit	Forward
Reset	Backward
Reflog recovery	To selected commit

---

10. Validation Checklist

Student can confirm:

- ✅ Deleted commit recovered
- ✅ Difference between soft and hard reset explained
- ✅ Meaning of HEAD explained clearly

---

11. Industry Mapping

In real projects:

- Developers commit wrong files
- Production fixes need rollback
- Panic makes things worse

Good developers are **not fast**. They are **calm and recover safely**.
