**What is need?**

**1. Track Changes Over Time**

* Keeps a history of every change made to the codebase.
* Helps understand who changed what and why.

**2. Collaborate with Others**

* Multiple developers can work on the **same project simultaneously**.
* Tools like Git manage merging and handle conflicts between contributors.

**3. Experiment Safely**

* Create **branches** to test new features or ideas without affecting the main project.
* Once tested, merge them safely into the main branch.

**4. Debugging and Bug Tracking**

* Easily identify and revert the commit that introduced a bug.
* Compare different versions of files to find what went wrong.

Basic Setup

1 git config --global user.name "name"

* What it does: Sets the global Git username to "name".
* Use: Git uses this name to label your commits across all repositories on your system.

2 git config --global user.email email@gmail.com

* What it does: Sets the global Git email address.
* Use: Git associates this email with your commits. This should match the email linked to your GitHub/GitLab account if you want contributions to show up.

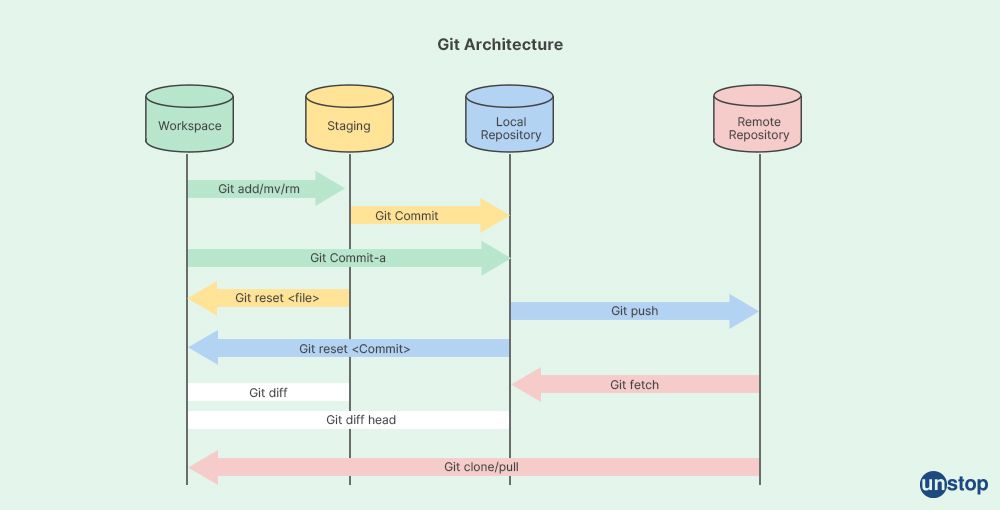
3 git config --global user.name

* What it does: Displays the current global Git username.
* Use: To verify which name is currently configured.

4 git config --global user.email

* What it does: Displays the current global Git email.
* Use: To verify which email is currently configured.

Git Architecture



Git Basic Commands

**1 git init**

PS D:\git\_github>git init

Initializes a new empty Git repository in the current directory.

Creates a hidden folder .git/ where Git stores version history, configuration, and tracking data.

**2 git status**

PS D:\git\_github>git status

On branch main/master

You're currently on the main/master branch.

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

first.py

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

Explanation:

Here's what the git status output means in your case:

git status Output Explained

On branch main

* You're currently on the main branch.

No commits yet

* You haven't committed anything to the repository yet.

Untracked files:

(use "git add <file>..." to include in what will be committed)

first.py

* The file first.py is present in your directory but is **untracked**, meaning Git isn't tracking its changes yet.
* To track it, you need to add it using:
* git add first.py

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

* Git is letting you know that while there are files in the folder, none have been added to the staging area, so a commit would have nothing to include yet.

**3 git add first.py / git add .**

PS D:\git\_github>git add first.py

PS D:\git\_github>git status

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

**new file:** first.py

Explanation

git add first.py Result:

On branch main

No commits yet

* You’re still on the main branch with no commits made yet.

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: first.py

* first.py is now **staged**, meaning it's ready to be committed.
* If you commit now, Git will include this file in the commit.

**Create delete.py file**

PS D:\git\_github>git status

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

**new file:** first.py

Untracked files:

(use "git add <file>..." to include in what will be committed)

delete.py

This means your file first.py **exists** in your project folder but Git is **not tracking** it yet.

Explanation of the Output:

* "No commits yet"  
  You haven’t created any commit so far.
* "Untracked files: first.py"  
  Git sees the file but it’s not being tracked (not staged).

Note: Delete delete.py file.

**4 git commit -m “initial commit”**

PS D:\git\_github>git commit -m "initial commit"

[main (root-commit) 6a79ecb] initial commit

1 file changed, 1 insertion(+)

create mode 100644 first.py

Explanation

* git commit -m "initial commit"  
  You created a commit with the message **"initial commit"**.
* 1 file changed, 1 insertion(+)  
  Git detected **1 line added** in first.py.
* create mode 100644 first.py  
   Git created the file with standard permissions (read/write for owner, read for others).

git status

PS D:\git\_github>git status

On branch main

nothing to commit, working tree clean

modify first.py

PS D:\git\_github>git status

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

**modified:** first.py

no changes added to commit (use "git add" and/or "git commit -a")

Explanation

You've modified the file first.py after your last commit.

However, you **haven’t staged** those changes yet — so Git is just tracking that a modification has occurred.

git add first.py

PS D:\git\_github>git add first.py

PS D:\git\_github>git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

**modified:** first.py

Explanation

* You are on the main branch.
* You **have staged** the changes to first.py.
* It's ready to be committed.

git commit -m “add variable b”

PS D:\git\_github>git commit -m "add variable b"

[main 478b94e] add variable b

1 file changed, 2 insertions(+), 1 deletion(-)

.gitigonre

Note: If you've already added delete.txt to Git before adding it to .gitignore, Git will continue to track it. Adding a file to .gitignore does not untrack files that are already being tracked.

.gitignore file tells Git which files or folders to ignore (i.e., not track) in your repository.

Create .gitigonre file, create delete.txt file, enter file name which you want to untrack with extension in `.gitignore` file, in our case it is delete.txt

git status

PS D:\git\_github>git status

On branch main

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

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

git add .gitignore

PS D:\git\_github>git add .gitignore

PS D:\git\_github>git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: .gitignore

git commit -m “.gitignore added”

PS D:\git\_github>git commit -m ".gitignore added"

[main 8cab47e] .gitignore added

1 file changed, 1 insertion(+)

create mode 100644 .gitignore

PS D:\git\_github>git status

On branch main

nothing to commit, working tree clean

Seeing Commits

**1 git log**

Author: AuthorName <email@gmail.com>

Date: Sun Jul 6 16:20:25 2025 +0530

.gitignore added

commit 478b94eaf5d31bb1262ddcf9732a56ac8957a06f

Author: AuthorName<email@gmail.com>

Date: Sun Jul 6 15:56:33 2025 +0530

add variable b

commit 6a79ecb9b38eaf38115fe22ee1214c98b2e71605

Author: AuthorName<email@gmail.com>

Date: Sat Jul 5 22:52:29 2025 +0530

initial commit

Explanation

* HEAD -> main means you're currently on the main branch and pointing to the latest commit.
* It is a **unique identifier** automatically generated by Git for each commit.
* Git uses a hashing algorithm called **SHA-1** to produce this 40-character string.

**2 git log --oneline**

PS D:\git\_github>git log --oneline

8cab47e (HEAD -> main) .gitignore added

478b94e add variable b

6a79ecb initial commit

**3 git log –-stat**

PS D:\git\_github>git log –-stat

commit 8cab47eeb205029a6106702e29716945179b0666 (HEAD -> main)

Author: name <email@gmail.com>

Date: Sun Jul 6 16:20:25 2025 +0530

.gitignore added

.gitignore | 1 +

1 file changed, 1 insertion(+)

commit 478b94eaf5d31bb1262ddcf9732a56ac8957a06f

Author: name <email@gmail.com>

Date: Sun Jul 6 15:56:33 2025 +0530

add variable b

first.py | 3 ++-

1 file changed, 2 insertions(+), 1 deletion(-)

commit 6a79ecb9b38eaf38115fe22ee1214c98b2e71605

Author: name <email@gmail.com>

Date: Sat Jul 5 22:52:29 2025 +0530

initial commit

first.py | 1 +

1 file changed, 1 insertion(+)

**3 git log -p**

It displays:

1. Commit metadata:
   * Commit hash
   * Author
   * Date
   * Commit message
2. Code changes (patch) for each commit:
   * Which lines were added (+)
   * Which lines were removed (-)
   * For each modified file

**4 git show <SHA id>**

To review changes made in a specific commit

PS D:\git\_github>git show 6a79ecb

commit 6a79ecb9b38eaf38115fe22ee1214c98b2e71605

Author: name <email@gmail.com>

Date: Sat Jul 5 22:52:29 2025 +0530

initial commit

diff --git a/first.py b/first.py

new file mode 100644

index 0000000..304f821

--- /dev/null

+++ b/first.py

@@ -0,0 +1 @@

+a = "welcome to Mlops"

\ No newline at end of file

**5 git diff / git diff filename**

Changes made within staging area before commit

git diff <commit1> <commit2>: compare any two commits

PS D:\git\_github>git diff first.py

diff --git a/first.py b/first.py

index e74e495..f8343f2 100644

--- a/first.py

+++ b/first.py

@@ -1,2 +1,4 @@

a = "welcome to Mlops"

-b = "welcome world"

\ No newline at end of file

+b = "welcome world"

+

+c = "new variable c"

\ No newline at end of file

**6 git checkout <commit id>**

Lets you view or experiment from any commit without changing branch history.

Does not move your branch.

You are in a detached HEAD state, any commits made now won’t belong to a branch unless you create one.

PS D:\git\_github>git log --oneline

8cab47e (HEAD -> main) .gitignore added

478b94e add variable b

6a79ecb initial commit

PS D:\git\_github>git checkout 478b94e

M first.py

Note: switching to '478b94e'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 478b94e add variable b

PS D:\git\_github>git checkout 8cab47e

M first.py

Previous HEAD position was 478b94e add variable b

HEAD is now at 8cab47e .gitignore added

PS D:\git\_github>git checkout main

M first.py

Switched to branch 'main'

Summary of above steps:

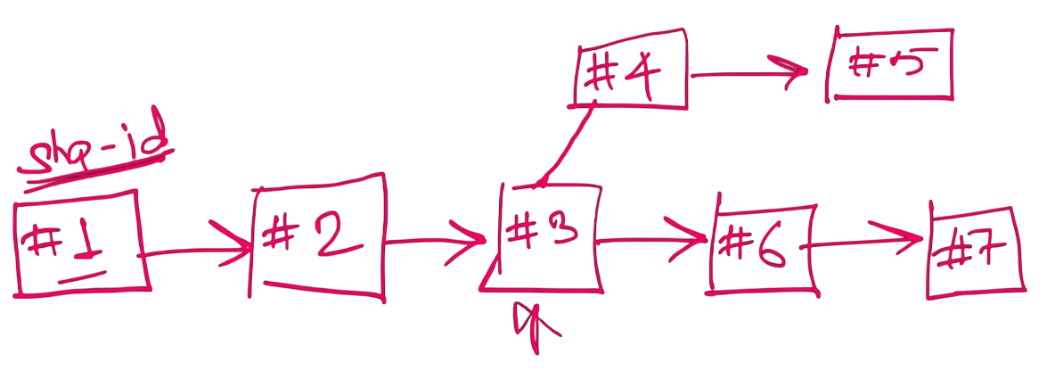
| Step | Action | Necessary? | Comment |
| --- | --- | --- | --- |
| checkout 478b94e | Checked out older commit | ✅ Yes | If exploring or creating a new branch |
| checkout 8cab47e | Checked out latest commit directly | ❌ No | Could’ve skipped; main already points here |
| checkout main | Returned to your working branch | ✅ Yes | Correct to resume normal work |

Git Branching and Merging

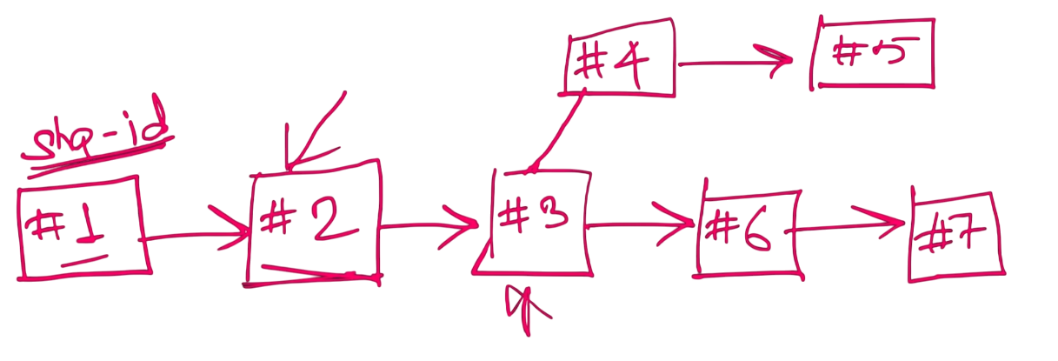
Okay, we’ve made three commits so far.

Now, what are we going to do?  
We’re going to create a new branch from commit three. Okay? We'll create a new branch and make the fourth commit and fifth commit on it.

Then, on the master/main branch, the next commit we make will be the **sixth commit, seventh commit**,  
because the fourth and fifth commits were made on the new branch.  
Currently, we are here commit number 3, and from here, we are going to create our first branch.



**Keep in mind**, it’s not necessary to create a branch from where you currently are.  
If you want, you can go back in history and create a branch from an earlier commit.



And how do you go back?  
As we just saw, you can use git checkout  
and provide the SHA ID of the desired commit to navigate there.

Once you’re there, you can create a branch from that point.  
But for now, the key is to understand how to create a branch.

So let’s now create a new branch and name it **helper-branch**.  
We will call it **helper-branch**.

**1 git branch <branch name> <commit id>**

Creates a new branch. The commit ID (optional) specifies the commit *from which* the new branch is created.

PS D:\git\_github>git branch helper

**2 git branch**

To list of all branches

PS D:\git\_github>git branch

helper

\* main

Currently on main branch

We don’t make any commit so currently head is pointing to junction point.

PS D:\git\_github> git log --oneline

8cab47e (HEAD -> **main, helper**) .gitignore added

478b94e add variable b

6a79ecb initial commit

**3 git checkout <branch name>**

Checkout to newly created helper branch

PS D:\git\_github>git checkout helper

Switched to branch 'helper'

PS D:\git\_github>git branch

\* helper

Main

Now create helper.py file and add one line of code to it and commit.

PS D:\git\_github>git add .

PS D:\git\_github>git commit -m "4 - helper work started"

[helper 03c8c40] 4 - helper work started

1 file changed, 1 insertion(+)

create mode 100644 helper.py

Now add one line of code again in helper.py and commit

PS D:\git\_github>git add .

PS D:\git\_github>git commit -m "5 - helper work finished"

[helper b7209fb] 5 - helper work finished

1 file changed, 2 insertions(+), 1 deletion(-)

main/master branch has no new commits since the source branch was created so,

Logs (commits) of helper branch

git log --oneline

b7209fb (HEAD -> helper) 5 - helper work finished

03c8c40 4 - helper work started

8cab47e (main) .gitignore added

478b94e add variable b

6a79ecb initial commit

**Checkout to main branch, head is pointing to the commit number 3.**

PS D:\git\_github>git checkout main

Switched to branch 'main'

Logs (commits) of main branch

PS D:\git\_github>git log --oneline

8cab47e (HEAD -> main) .gitignore added

478b94e add variable b

6a79ecb initial commit

Now add one line of code in first.py and commit.

PS D:\git\_github>git add .

PS D:\git\_github>git commit -m "6 - first commit post helper"

[main d492b95] 6 - first commit post helper

1 file changed, 3 insertions(+), 1 deletion(-)

Now add one line of code again in first.py and commit.

PS D:\git\_github>git add .

PS D:\git\_github>git commit -m "7 - second commit post helper"

[main 787e7b3] 7 - second commit post helper

1 file changed, 2 insertions(+), 1 deletion(-)

Logs (commits) of main branch

PS D:\git\_github>git log --oneline

787e7b3 (HEAD -> main) 7 - second commit post helper

d492b95 6 - first commit post helper

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

Logs (commits) of helper branch

PS D:\git\_github>git checkout helper

Switched to branch 'helper'

PS D:\git\_github>git log --oneline

b7209fb (HEAD -> helper) 5 - helper work finished

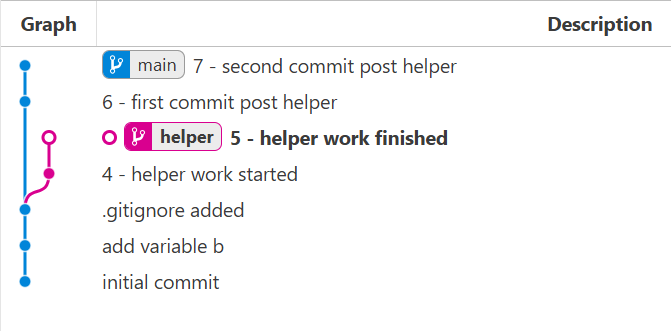
03c8c40 4 - helper work started

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

**Structure of branches in git graph VS code extension.**



**4 git log --oneline --all**

Shows all commits from all local branches in compact one-line format.

PS D:\git\_github>git log --oneline --all

787e7b3 (main) 7 - second commit post helper

d492b95 6 - first commit post helper

b7209fb (HEAD -> helper) 5 - helper work finished

03c8c40 4 - helper work started

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

PS D:\git\_github>git checkout main

Switched to branch 'main'

PS D:\git\_github>git log --oneline --all

787e7b3 (HEAD -> main) 7 - second commit post helper

d492b95 6 - first commit post helper

b7209fb (helper) 5 - helper work finished

03c8c40 4 - helper work started

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

**5 git log --oneline --all –graph**

Visuals of branches with all commits

PS D:\git\_github>git log --oneline --all --graph

\* 787e7b3 (HEAD -> main) 7 - second commit post helper

\* d492b95 6 - first commit post helper

| \* b7209fb (helper) 5 - helper work finished

| \* 03c8c40 4 - helper work started

|/

\* 8cab47e .gitignore added

\* 478b94e add variable b

\* 6a79ecb initial commit

**6 git branch -d/-D<branch name>**

You cannot delete a branch while you are currently on it.

जिस डाली पर बैठे हो, उसे नहीं तोड़ सकते.

git checkout helper

Switched to branch 'helper'

PS D:\git\_github>git branch -d helper

error: cannot delete branch 'helper' used by worktree at 'D:/git\_github'

PS D:\git\_github>git branch -D helper

error: cannot delete branch 'helper' used by worktree at 'D:/git\_github'

**git branch -d <branch name>**

Deletes the branch **only if it has been fully merged** into the current branch or any upstream branch.

PS D:\git\_github>git checkout main

Switched to branch 'main'

PS D:\git\_github>git branch -d helper

error: the branch 'helper' is not fully merged

hint: If you are sure you want to delete it, run 'git branch -D helper'

hint: Disable this message with "git config set advice.forceDeleteBranch false"

**git branch -D <branch name>**

Force deletes the branch, regardless of merge status.

PS D:\git\_github>git branch -D helper

Deleted branch helper (was b7209fb).

PS D:\git\_github> git log --oneline

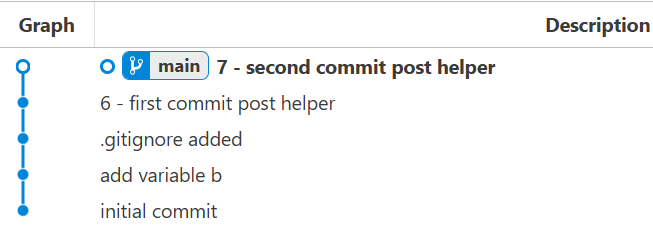
787e7b3 (HEAD -> main) 7 - second commit post helper

d492b95 6 - first commit post helper

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit



**Merging**

**7 git merge <file name>**

Merging Rules

* You must be on the branch *into which* you want to merge (on main, to merge helper into main).
* If no files were changed in both branches at the same place (no merge conflict), merge will be clean.
* If there is no new commit on the target branch (e.g., main), Git just moves the pointer forward and does not create a merge commit.

Create new branch named helper2 and checkout helper2 branch

PS D:\git\_github> git branch helper2

PS D:\git\_github> git branch

helper2

\* main

Switch to helper2

PS D:\git\_github> git checkout helper2

Switched to branch 'helper2'

Create file helper2.py add one line of code and commit.

git commit -m "8 - first commit of helper2"

[helper2 b8ec3eb] 8 - first commit of helper2

1 file changed, 1 insertion(+)

create mode 100644 helper2.py

PS D:\Study\_Files\MLOps\Day2\_Git\_GitHub\git\_github>

Add one line of code again and commit.

PS D:\git\_github> git add .

PS D:\git\_github> git commit -m "9 - second commit of helper2"

[helper2 e011c01] 9 - second commit of helper2

1 file changed, 2 insertions(+), 1 deletion(-)

main/master branch has no new commits since the source branch was created.

Logs (commits) of helper2 branch looks like this

PS D:\git\_github> git log --oneline

e011c01 (HEAD -> helper2) 9 - second commit of helper2

b8ec3eb 8 - first commit of helper2

787e7b3 (main) 7 - second commit post helper

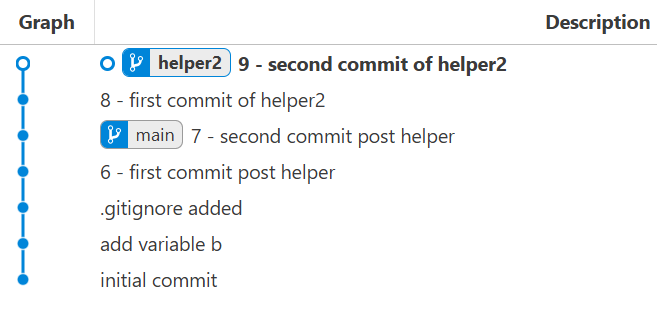
d492b95 6 - first commit post helper

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

Git graph looks like this



**Fast Forward Merging**

Fast-forward merge is a type of Git merge that happens when the target branch (main) has no new commits since the source branch (helper2) was created.

PS D:\git\_github> git checkout main

Switched to branch 'main'

PS D:\git\_github> git merge helper2

Updating 787e7b3..e011c01

Fast-forward

helper2.py | 2 ++

1 file changed, 2 insertions(+)

create mode 100644 helper2.py

PS D:\git\_github> git log --oneline

e011c01 (HEAD -> main, helper2) 9 - second commit of helper2

b8ec3eb 8 - first commit of helper2

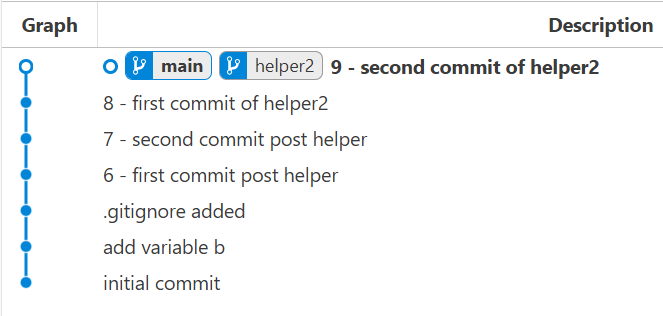
787e7b3 7 - second commit post helper

d492b95 6 - first commit post helper

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit



Delete helper2 branch

PS D:\git\_github git branch -d helper2

Deleted branch helper2 (was e011c01).

PS D:\git\_github> git log --oneline

e011c01 (HEAD -> main) 9 - second commit of helper2

b8ec3eb 8 - first commit of helper2

787e7b3 7 - second commit post helper

d492b95 6 - first commit post helper

8cab47e .gitignore added

478b94e add variable b

6a79ecb initial commit

**Merge Conflict**

Merge conflict occurs when files were changed in both branches at the same place.

Create branch final and create new commit

Add one line of code at line number 7 in first.py

PS D:\git\_github> git branch final

PS D:> git branch

final

\* main

PS D:\Study\_Files\MLOps\Day2\_Git\_GitHub\git\_github> git checkout final

Switched to branch 'final'

PS D:\git\_github> git add .

PS D:\git\_github> git commit -m "10 final branch commit"

[final 8f8b363] 10 final branch commit

1 file changed, 3 insertions(+), 1 deletion(-)

Checkout to main/master branch and create new commit

Add one line of code at line number 7 in first.py

PS D:\git\_github> git checkout main

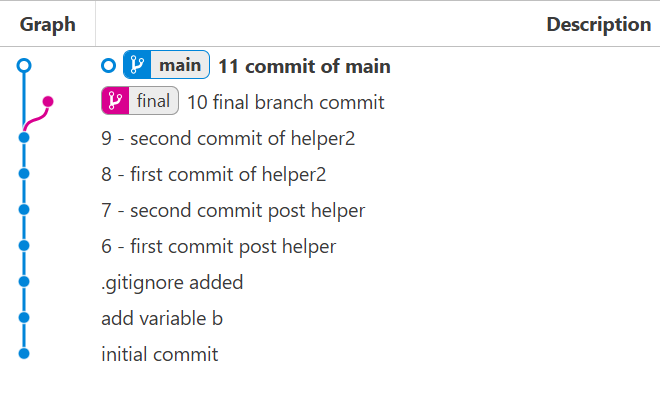
Switched to branch 'main'

PS D:\git\_github> git add .

PS D:\git\_github> git commit -m "11 commit of main"

[main ee91917] 11 commit of main

1 file changed, 3 insertions(+), 1 deletion(-)



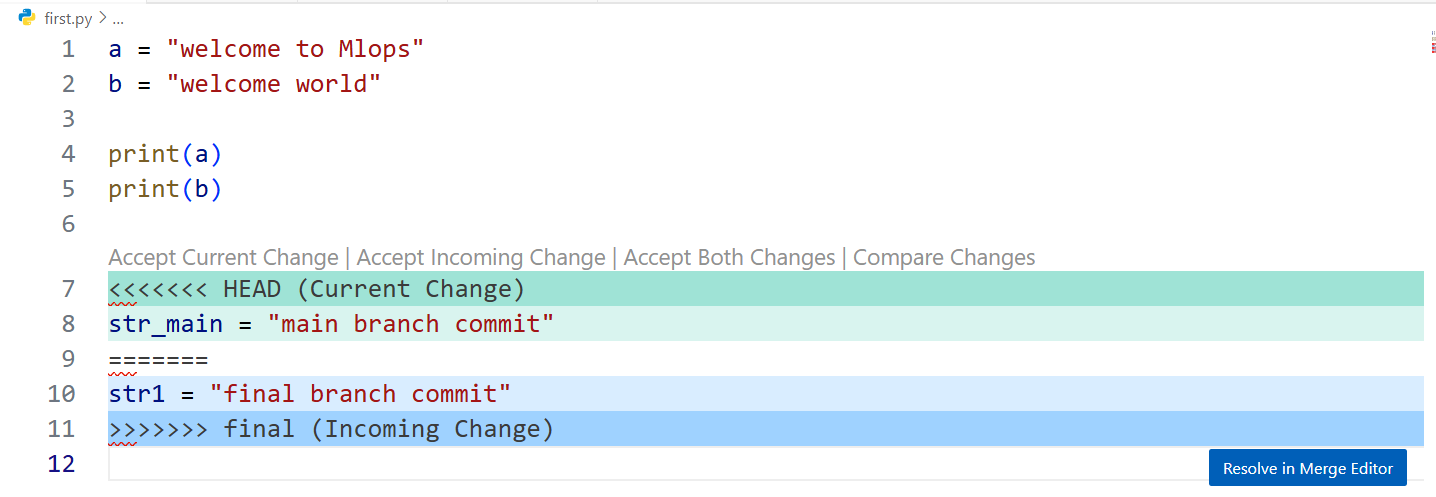
Merge final branch into main/master

PS D:\git\_github> git merge final

Auto-merging first.py

CONFLICT (content): Merge conflict in first.py

Automatic merge failed; fix conflicts and then commit the result.



**Merge Conflict Resolution**

* **Accept Current Change**: Keep the version from your current branch (main/master) and discard the incoming branch's (final) change.

git checkout --ours <filename>

* **Accept Incoming Change**: Keep the version from the branch you're merging in and discard your current branch's change.

git checkout --theirs <filename>

* **Accept Both Changes**: Keep both versions (current and incoming), placing them together in the file.

Manually select option in vs code Accept Both Changes

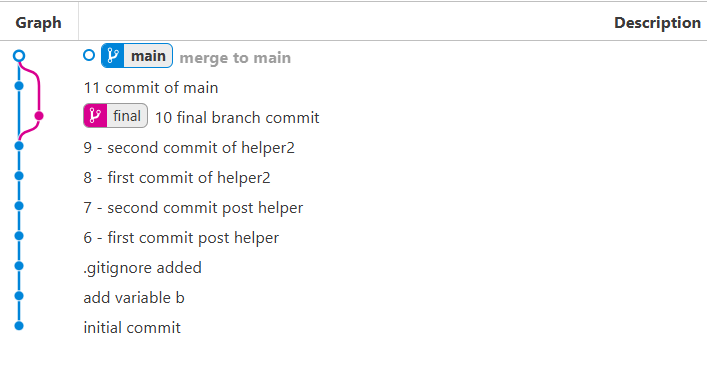
PS D:\git\_github> git checkout --ours first.py

Updated 1 path from the index

PS D:\git\_github> git add .

PS D:\git\_github> git commit -m "merge to main"

[main b6d2565] merge to main

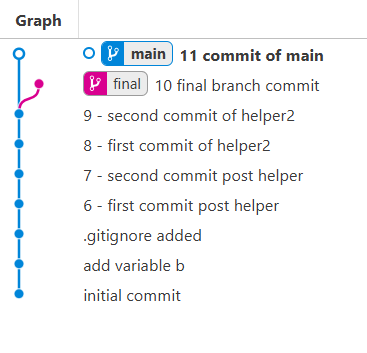
We accept commit of main branch not final branch

**8 git reset --hard HEAD~1 / git reset --hard <commit-hash>**

Forcefully resets your repository to a previous commit or specified commit, discarding all later commits and changes in the working directory.

PS D: \git\_github> git reset --hard HEAD~1

HEAD is now at ee91917 11 commit of main



Merge to main by accepting both changes

PS D:\git\_github> git merge final

Auto-merging first.py

CONFLICT (content): Merge conflict in first.py

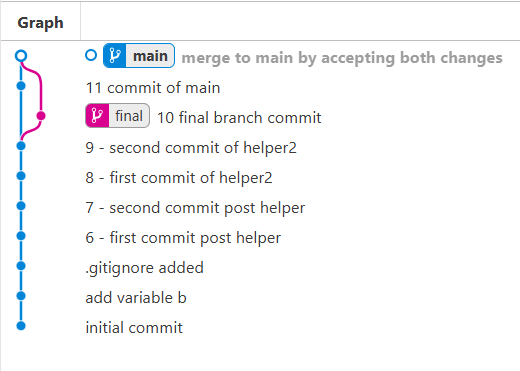
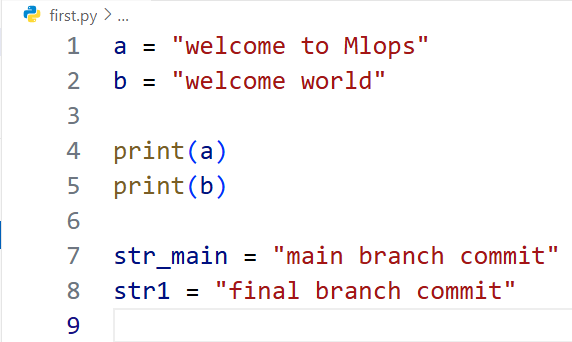
Automatic merge failed; fix conflicts and then commit the result.

Resolve conflict by select Accepting Both Changes in VsCode.

PS D:\git\_github> git add .

PS D:\git\_github> git commit -m "merge to main by accepting both changes"

[main 66a7ef8] merge to main by accepting both changes



GitHub

To push your **local Git repository** (with full history) to an **empty remote GitHub repository**, follow these steps:

**Step-by-step: Push Local Git Repo to GitHub Empty Repo**

Assume your GitHub repo URL is:

https://github.com/your-username/your-repo-name.git

**Step 1: Set the Remote URL**

In your local project directory:

git remote add origin https://github.com/your-username/your-repo-name.git

You can confirm it’s set with:

git remote -v

**Step 2: Push All Local History to GitHub**

git push -u origin main

-u sets the upstream, so future git push will default to origin main.

**If your local branch is not called main**

Check with:

git branch

If it's called master, push like:

git push -u origin master