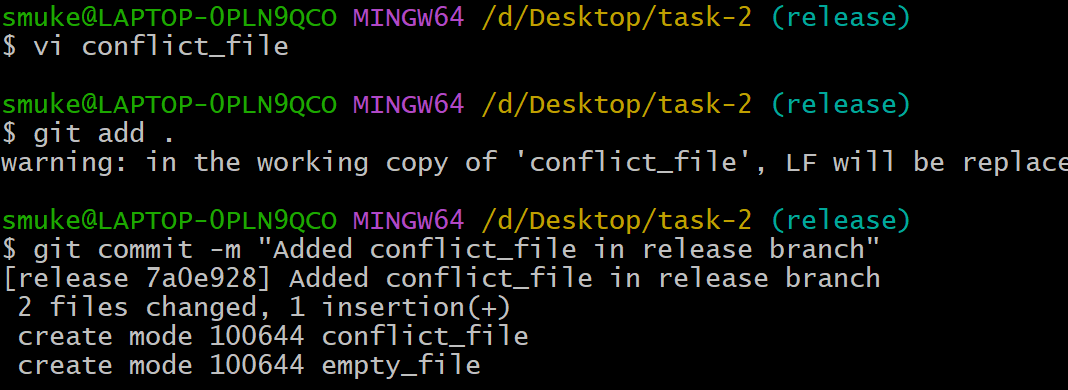
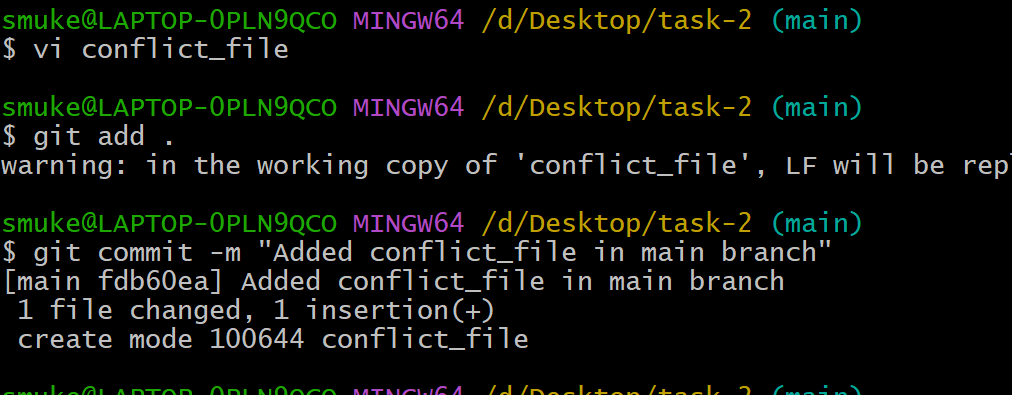
1. Resolve Merge conflict

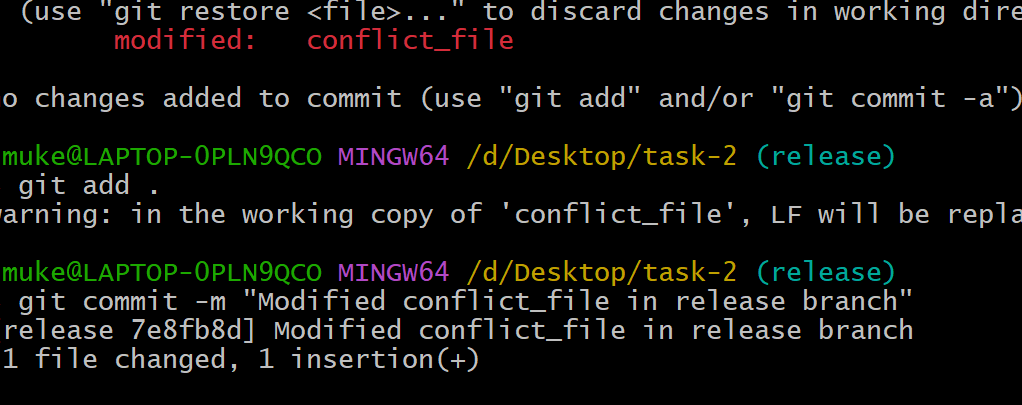
Step 1: create a file in one branch and commit it.



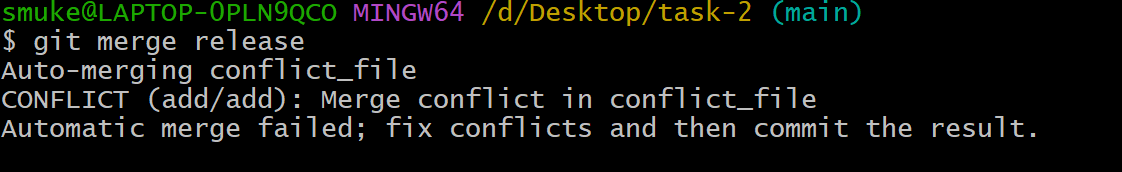
Step 2: switch to another branch and create a file with same name and commit it



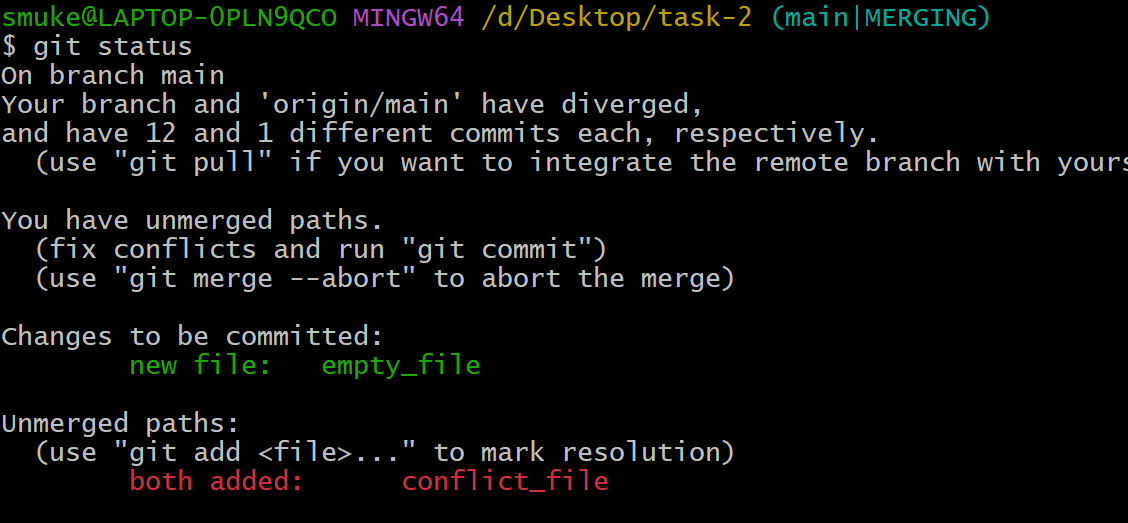
Step 3:Switch to previous branch modify the file and commit it.



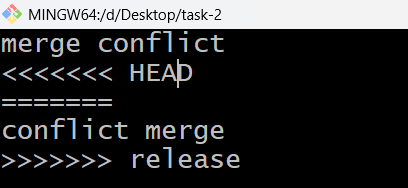
Step 4: Now comeback to other branch and give **git merge branchname c**onflict will occur

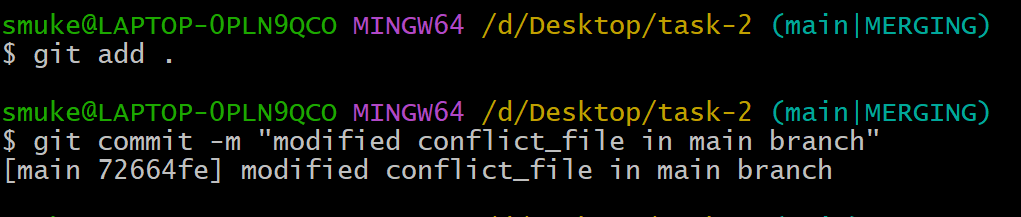


Step 5: Use **git status** to find which file has conflict issue



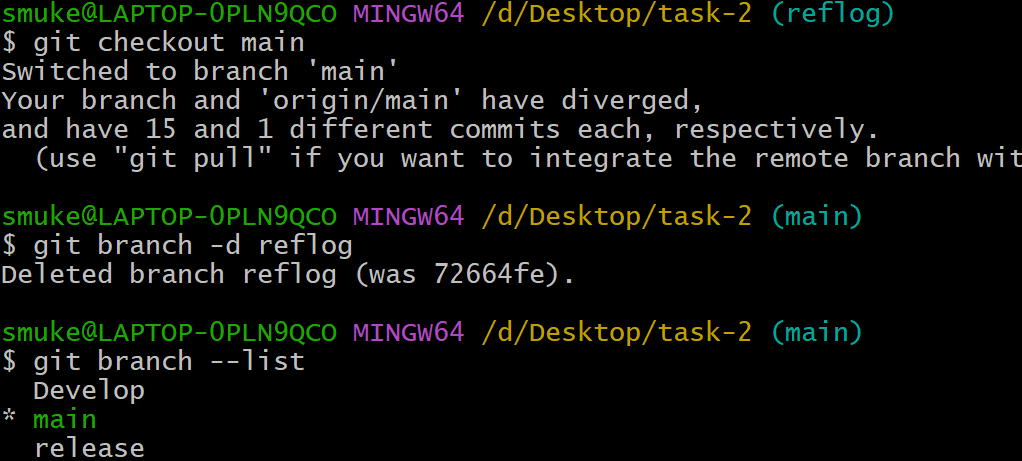
Step 6: Manually change the data of the file and commit again.



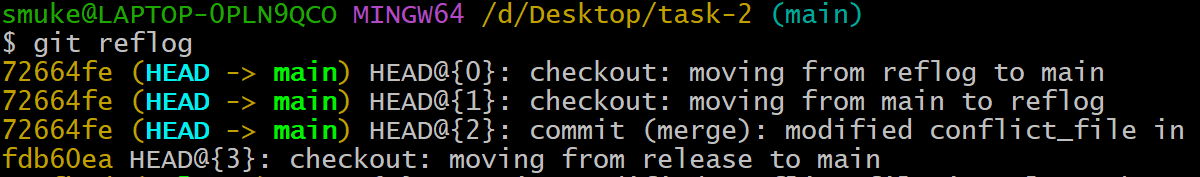


1. Recover deleted branch.

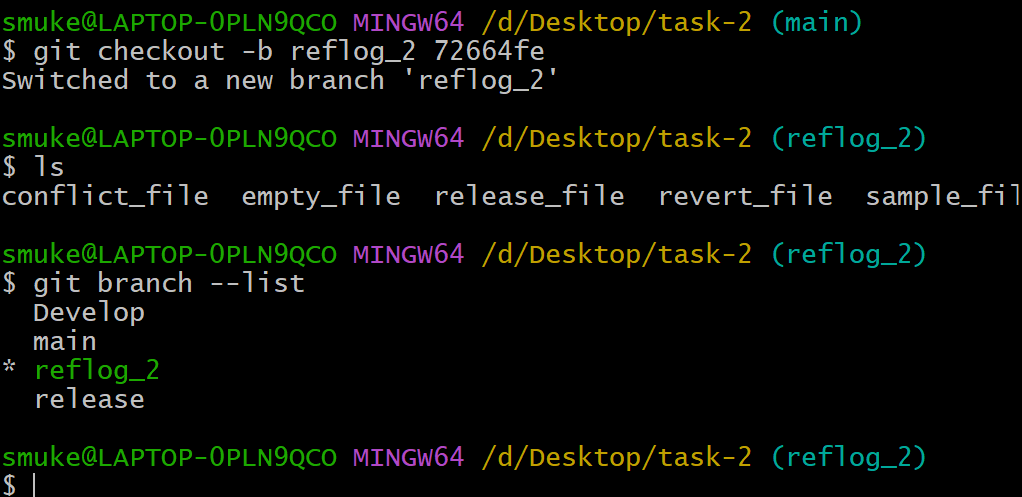
Step 1: switch to other branch and use **git branch -d branch\_name** to delete a branch.



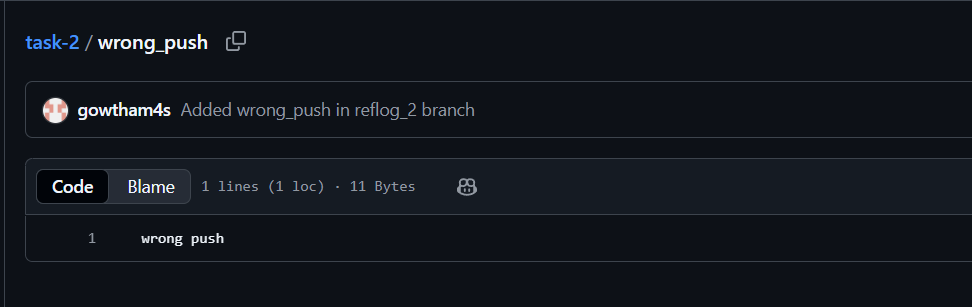
Step 2: Use **git reflog** to get commit it and copy latest commit id.

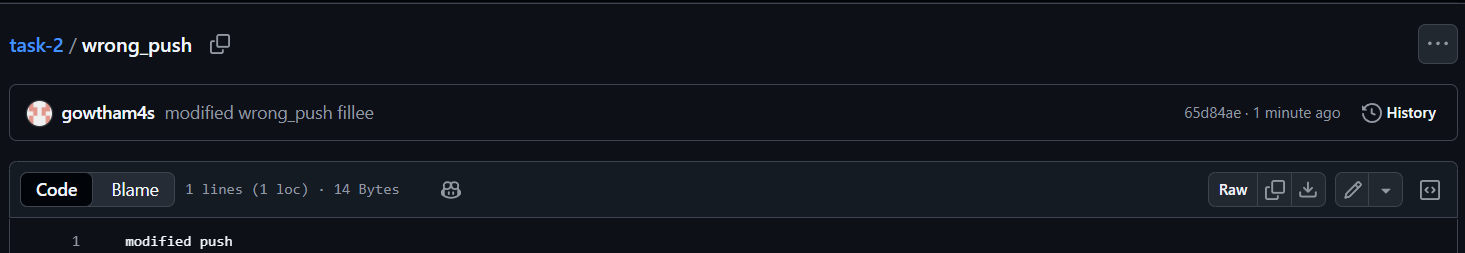


Step 3: Use **git checkout -b branch\_name commit\_id** to recover branch

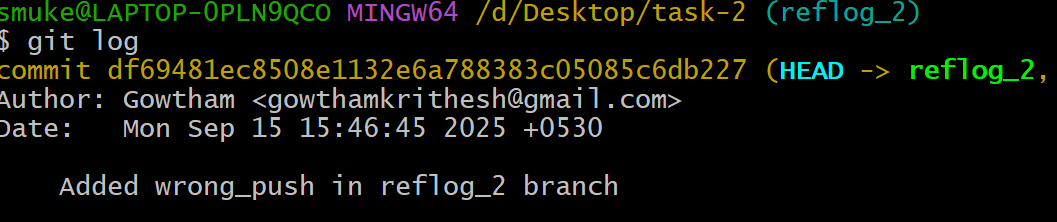


1. Undo wrong push

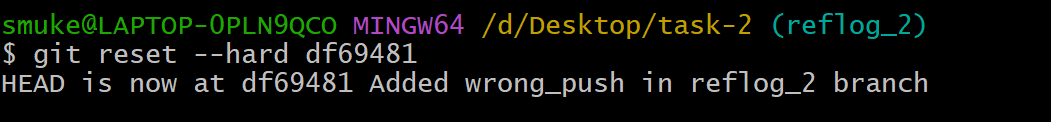




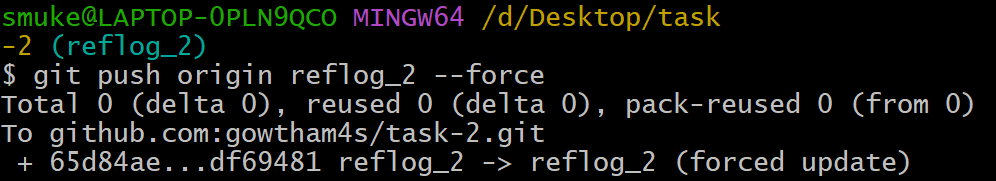
Step 1: Use **git log** to find commit id and copy commit id

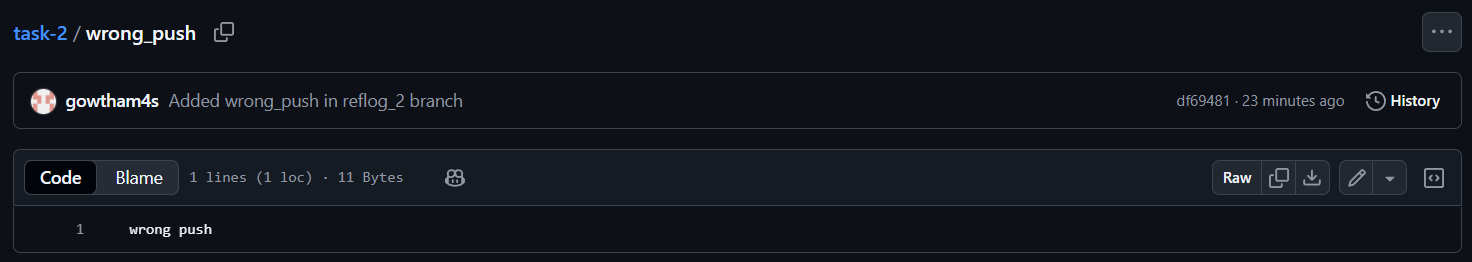


Step 2: use **git reset --hard commit\_id**

****

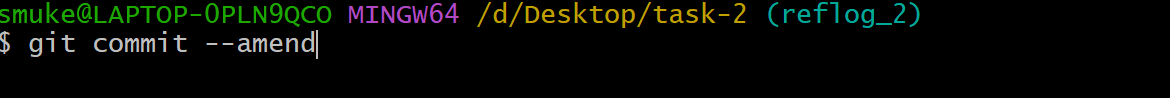
Step 3: Use **git push origin branch\_name --force** to revert changes

****

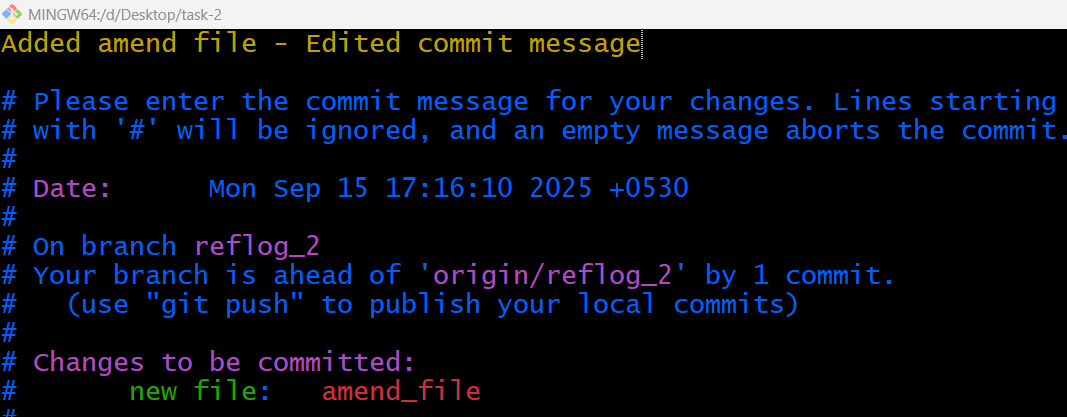
****

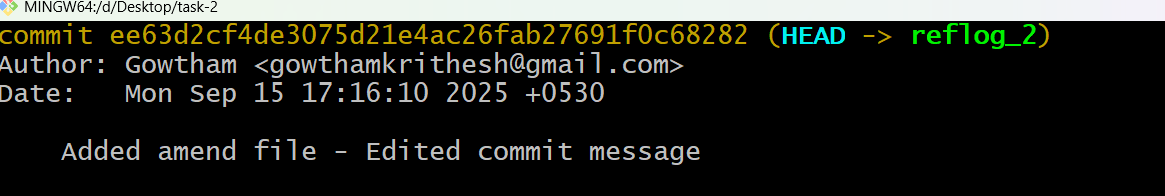
1. Amend a commit.

Step 1: Use **git commit –amend** to edit commit message



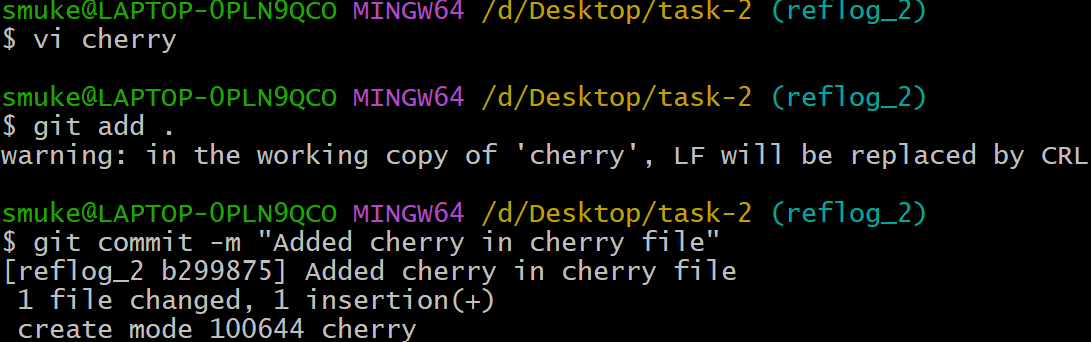
Step 2: Edit the message



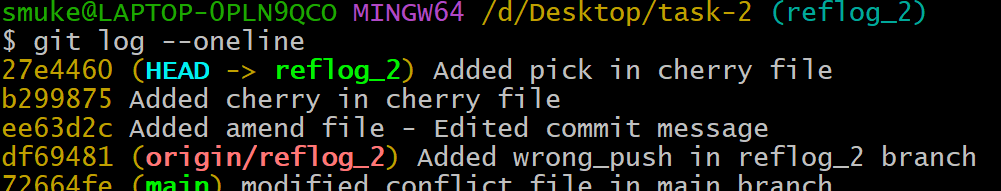


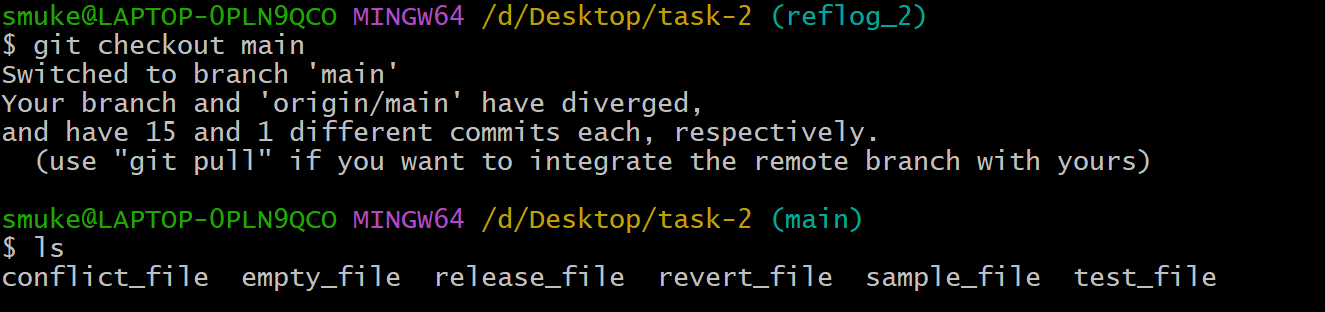
1. Cherry-Pick a commit.

Step 1:Create a file and commit it

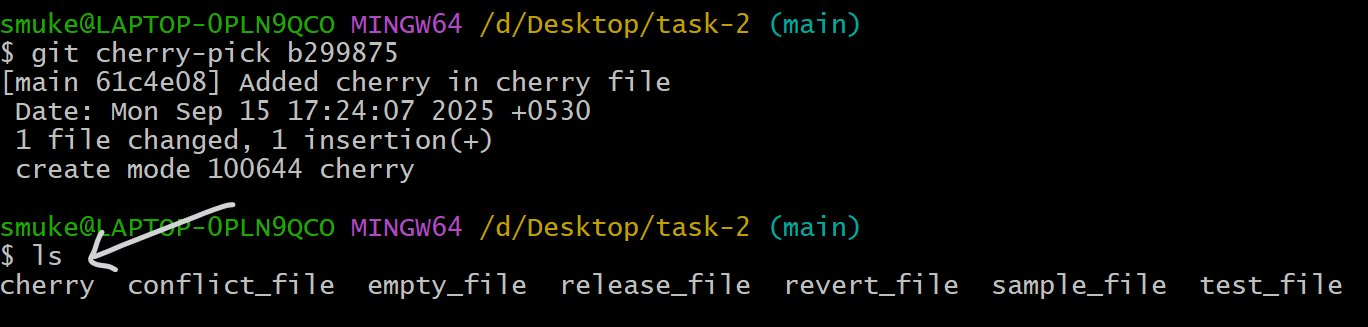
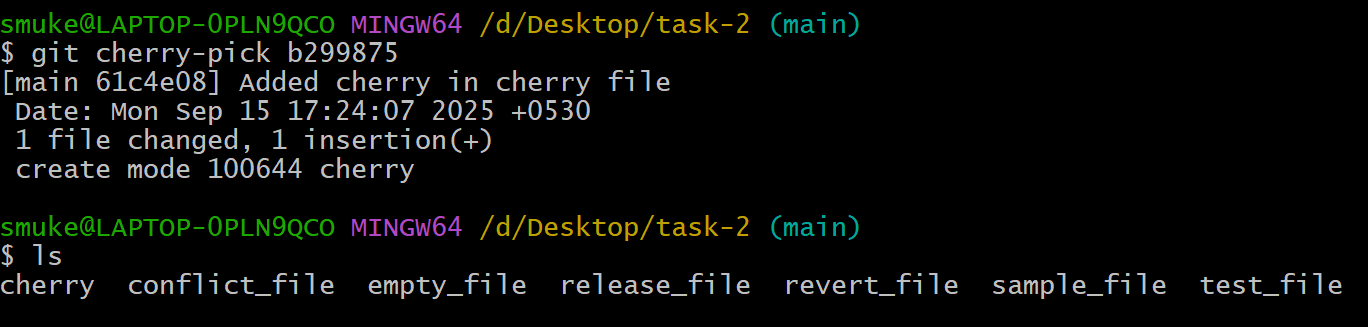


Step 2: Use **git log --oneline** to get commit id and switch to other branch

****

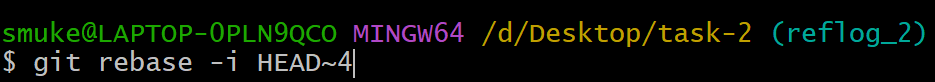
****

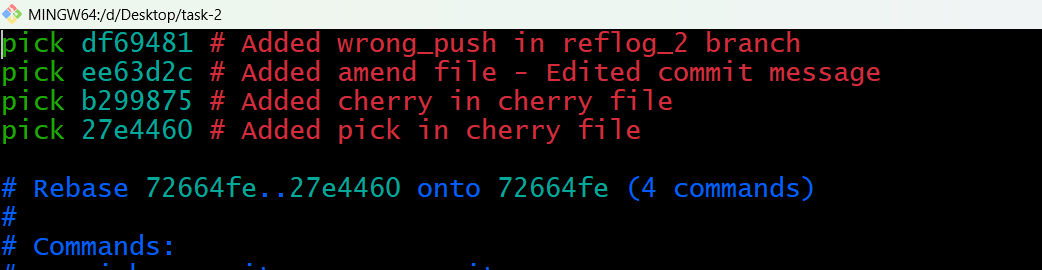
Step 3: Use **git cherry-pick commit\_id** to merge specific commit



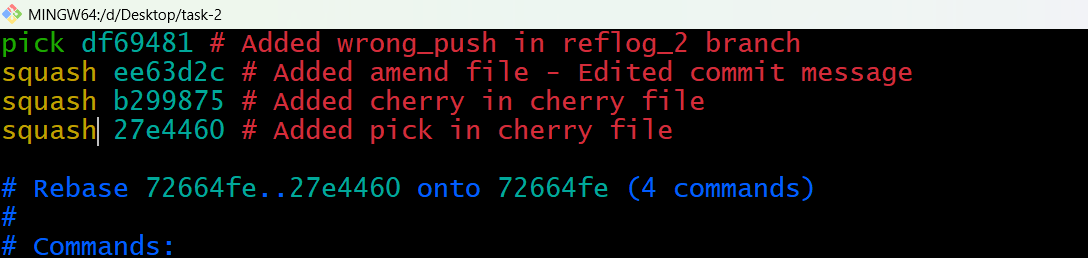
1. Interactive Rebase

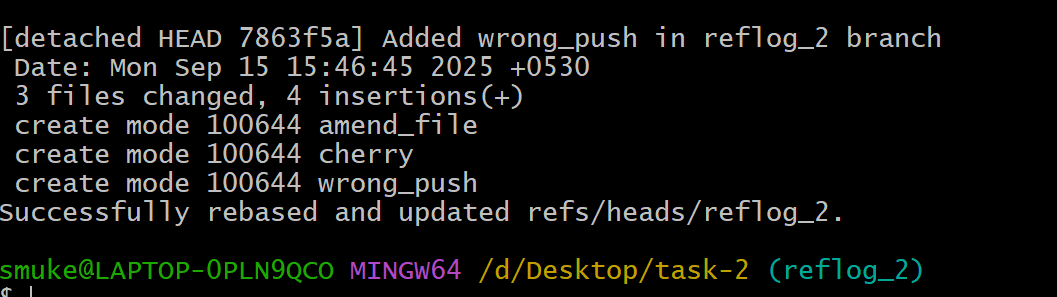
Step 1: Use **git rebase -i HEAD~4** to squash last 4 commits



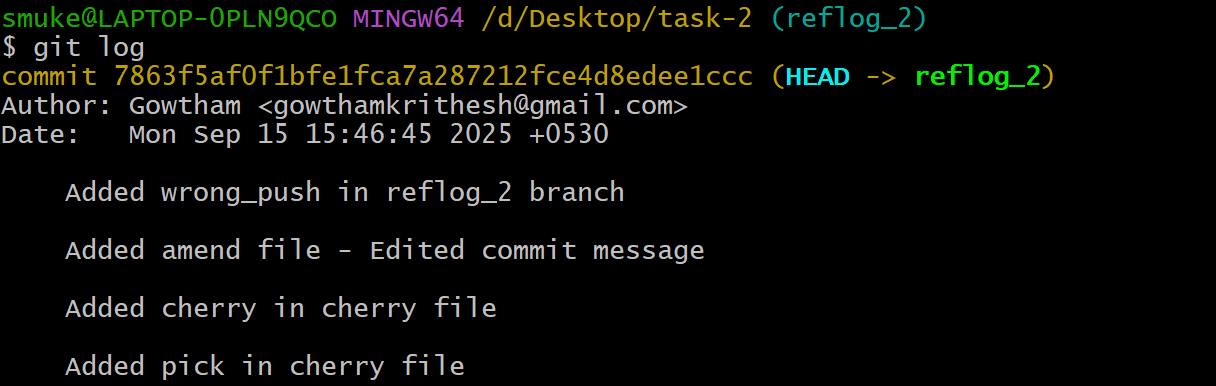


Step 2: Change the commits to be squashed from **pick** to **squash** and save it.



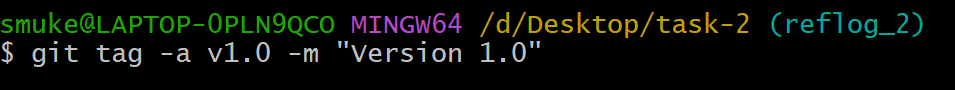


Step 3:Use **git log** to check the change

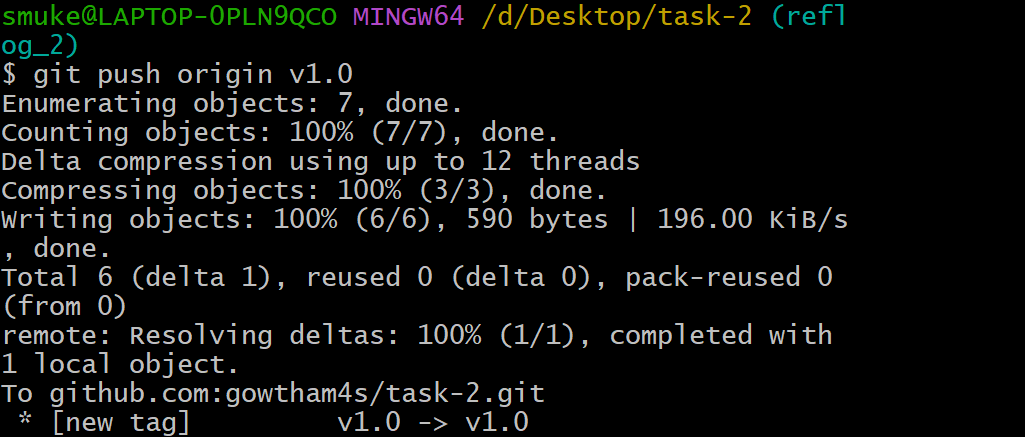


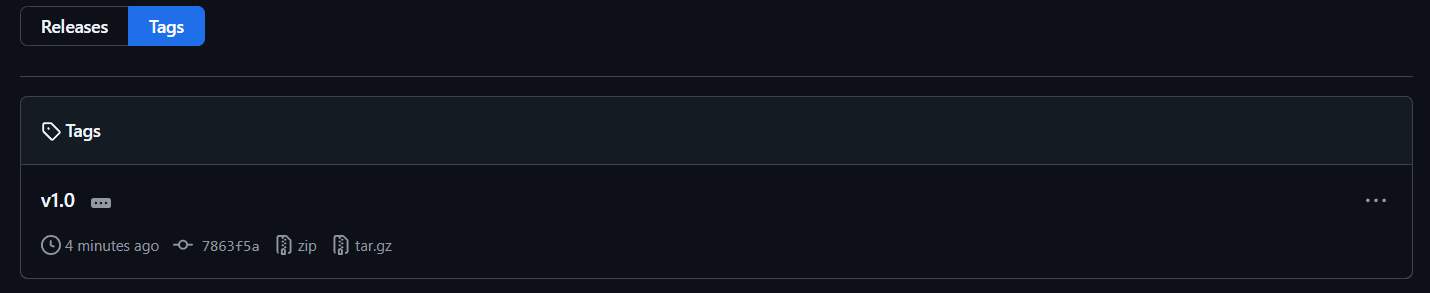
1. Tagging & Release

Step 1: Use **git tag -a tag\_name -m “Tag message”** to create a tag

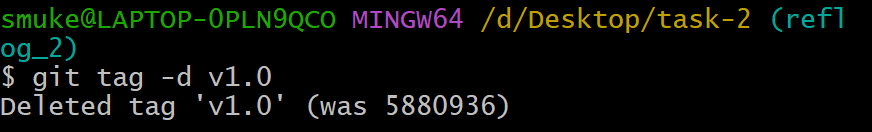
****

Step 2: Use **git push origin tag\_name** to push it to github

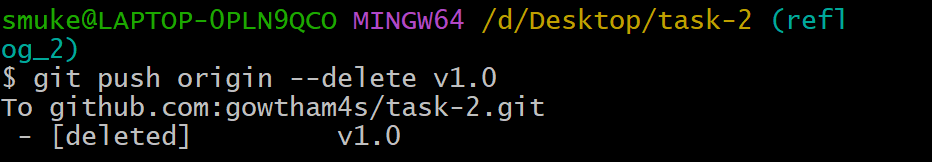


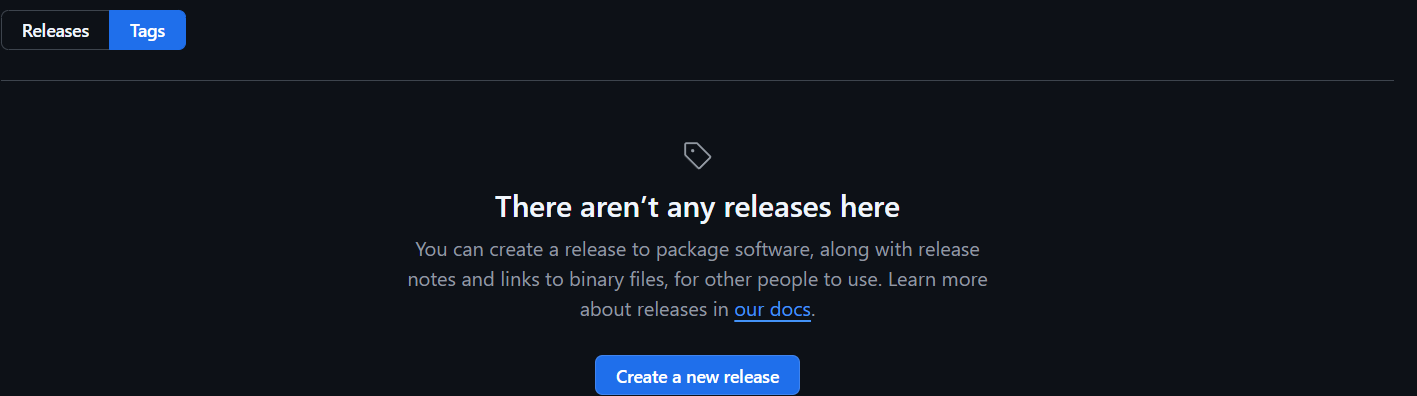


Step 3: Use **git tag -d v1.0** to delete the tag in Git.



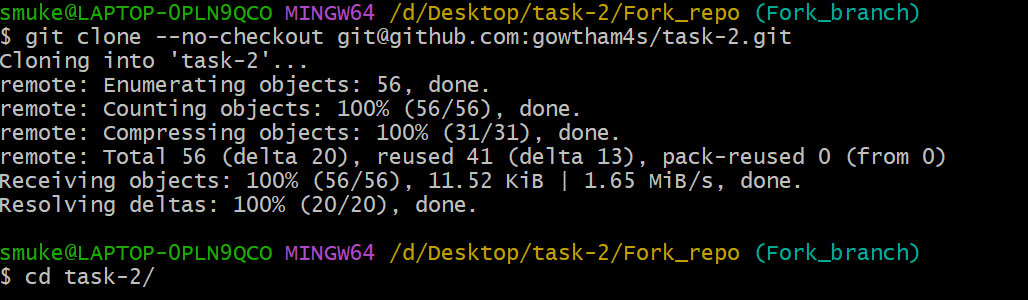
Step 4: Use **git push origin --delete tag\_name** to delete tag in GitHub.



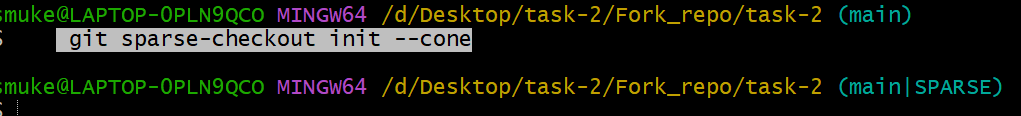


1. Clone with sparse checkout.

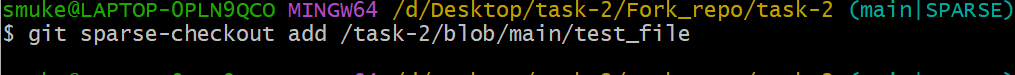
Step 1: Use **git clone --no-checkout repo\_link** to clone and give **cd repo\_name**



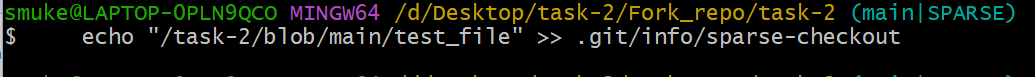
Step 2: Use **git sparse-checkout init --cone** to enable sparse checkout for current directory



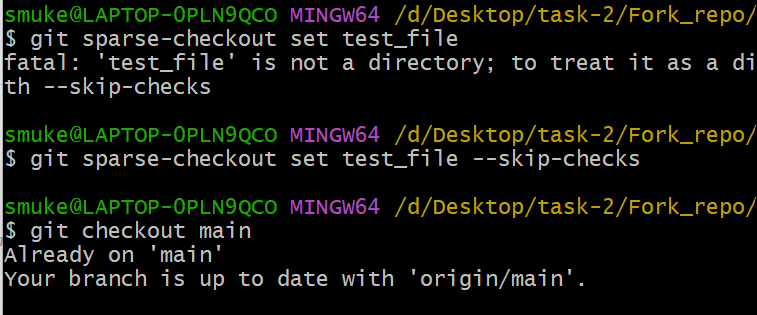
Step 3:Use **git sparse-checkout add directory\_path** to sparse checkout configuration



Step 4: Use **echo “path/to/directory” >> .git/info/sparse-checkout**



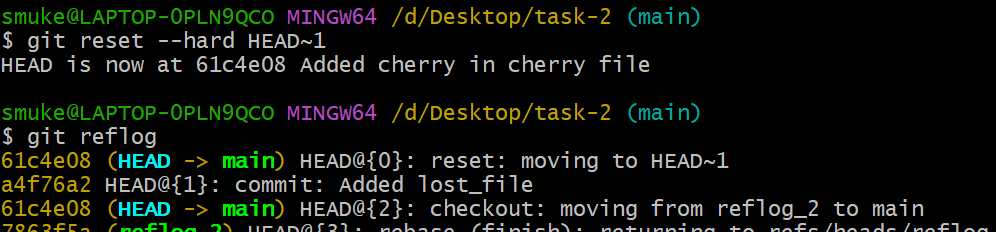
Step 4: Use **git sparse-checkout set file\_name &** **git checkout branch\_name**

****

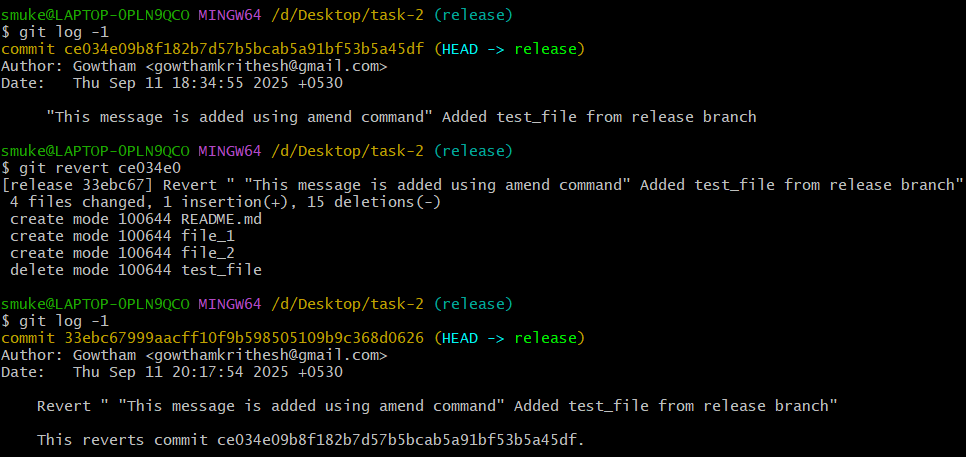
1. Reset vs Revert

**Reset**

**git reset --hard commit\_id** will rewrites the project history. It moves the HEAD pointer and the current branch reference to the specified **commit.**

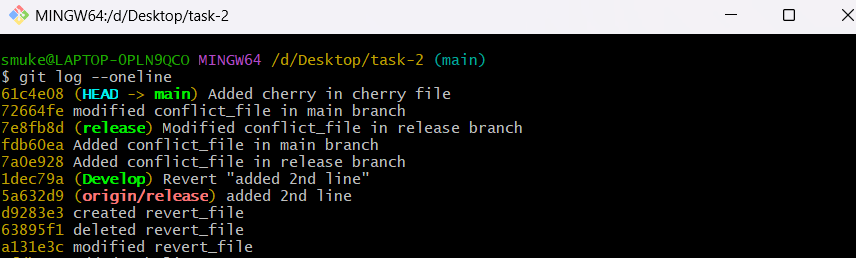


**Revert**

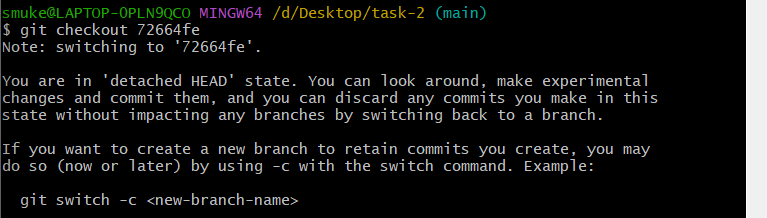
**git revert commit\_id** creates a new commit that undoes the changes introduced by the specified **commit.** 

1. Detached HEAD challenge

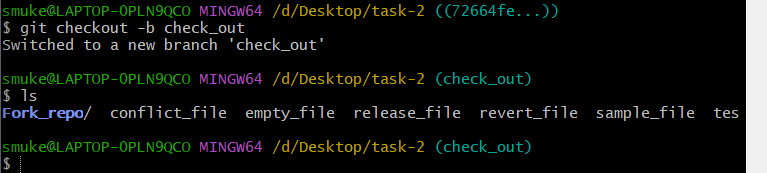
Step 1: Use **git log –online** to get the commit id.



Step 2: Use **git checkout commit\_id** to switch to that commit\_id.

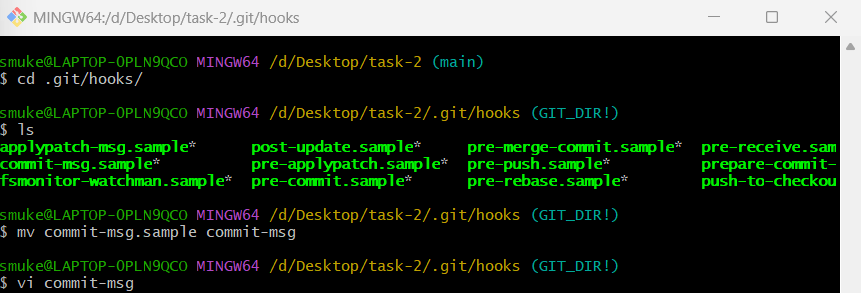


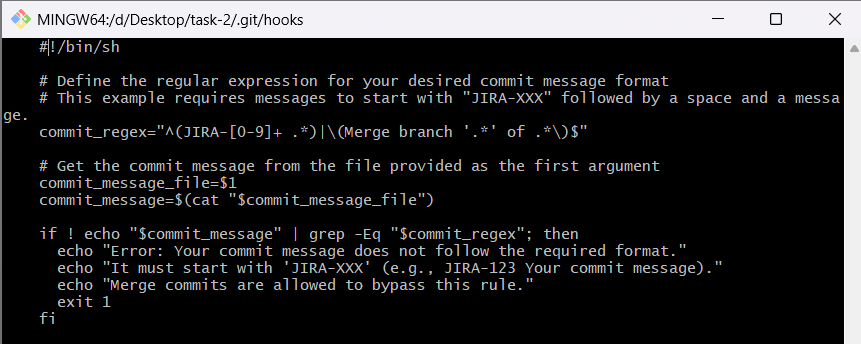
Step 3: Use **git checkout -b new\_branch\_name** to create new branch and retain commit.



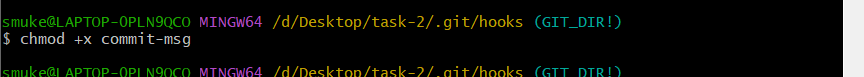
1. Git Hooks Challenge

Step 1: Use **cd .git/hooks/** and enter the scrip in **commit-msg** file

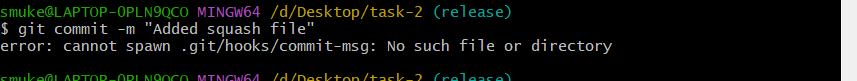




Step 2: Use **chmod +x commit-msg** to give executable permission to the file.



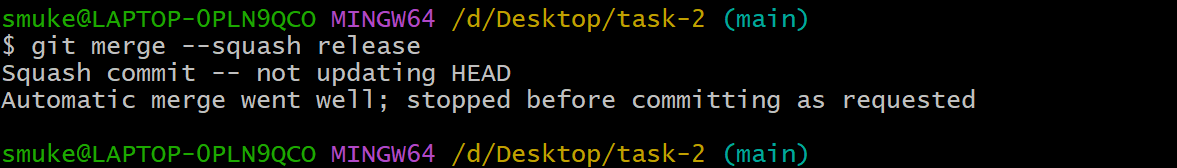
Now, whenever you attempt to commit, this commit-msg hook will automatically check if your commit message adheres to the specified "JIRA-XXX" format or not. If doesn’t commit will be rejected with error message.

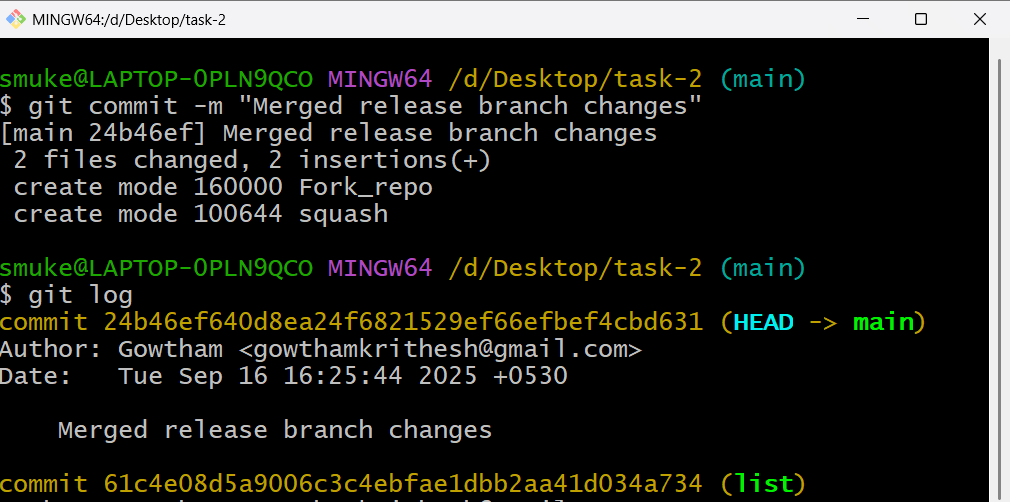


1. Squash Merge vs Rebase Merge.

**Squash Merge**

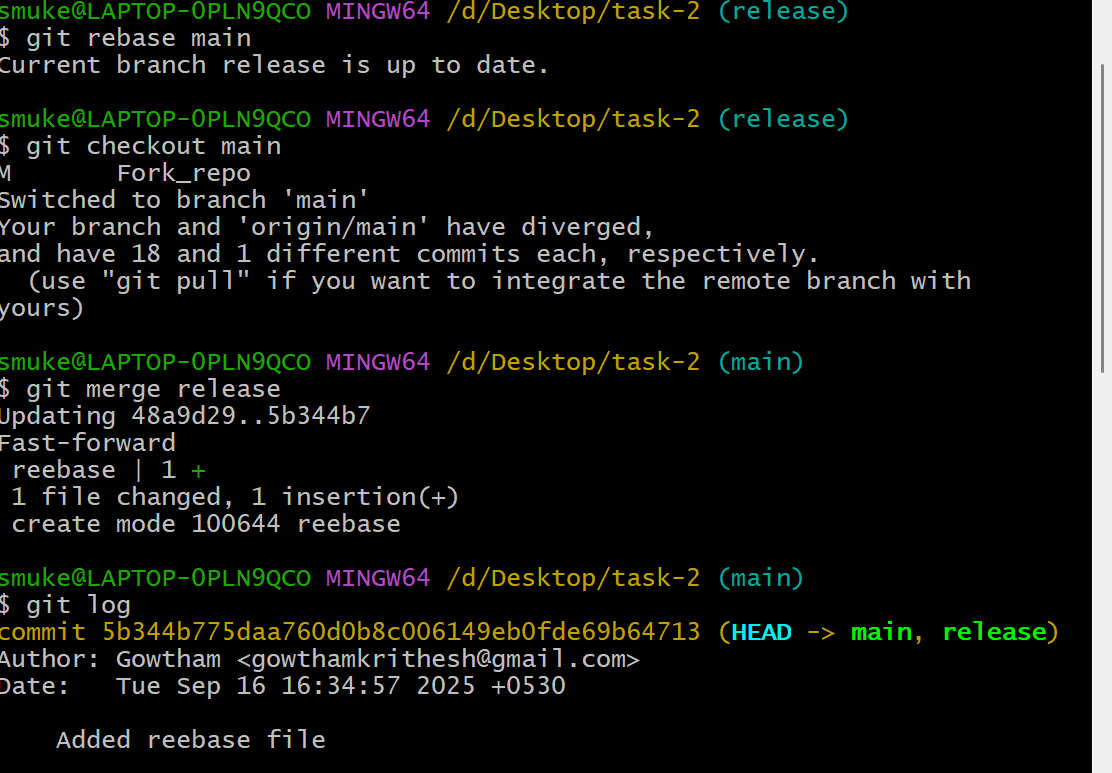
A squash merge will consolidate all commits from other branch into a single new commit on the target branch.





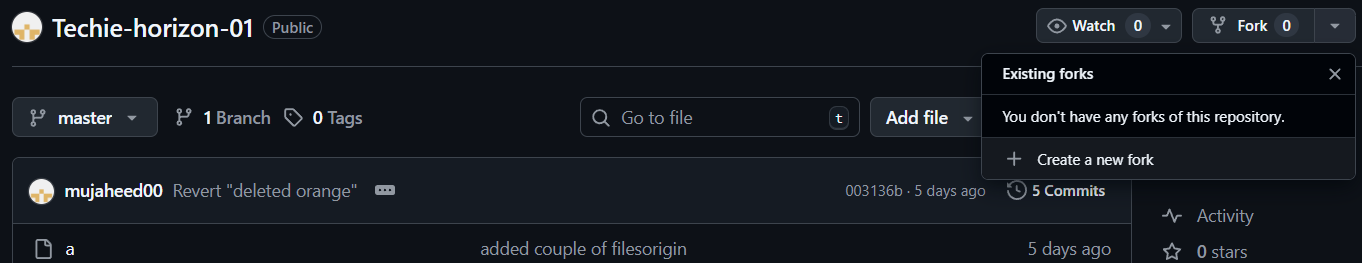
**Rebase Merge**

A rebase merge rewrites the commit history of the other branch by moving its base to the tip of the target branch.

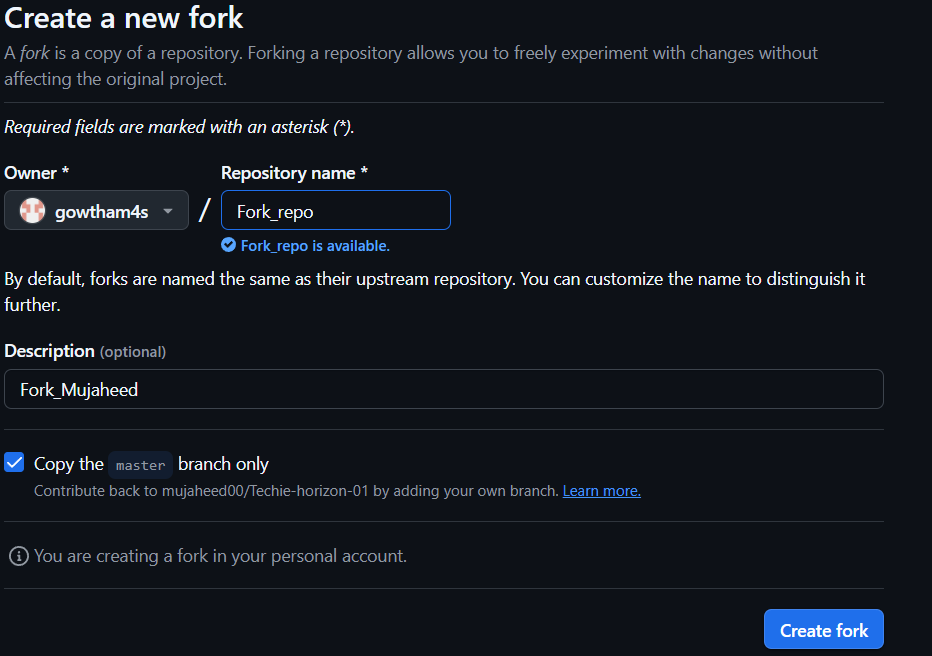


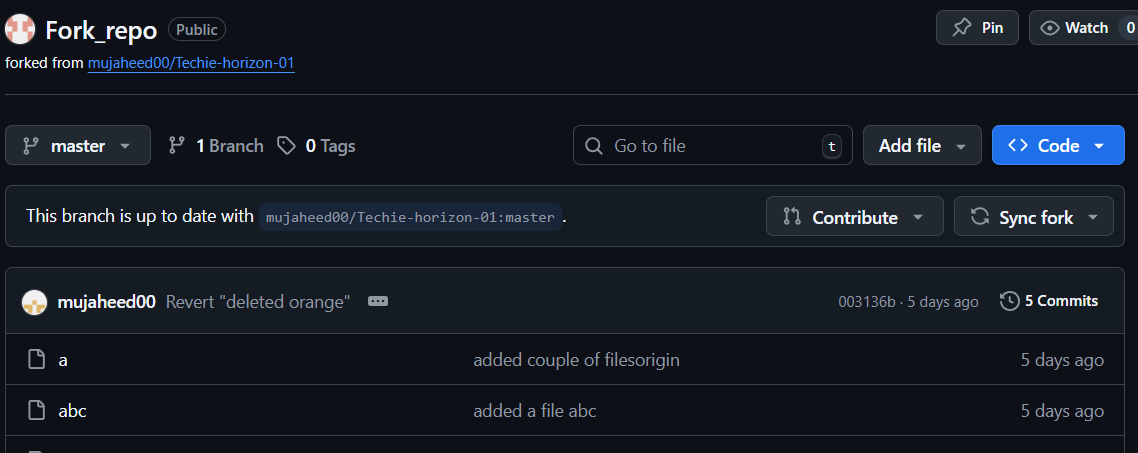
13. Fork & Pull request workflow

Step 1: open other repo click Fork 🡪create a new fork

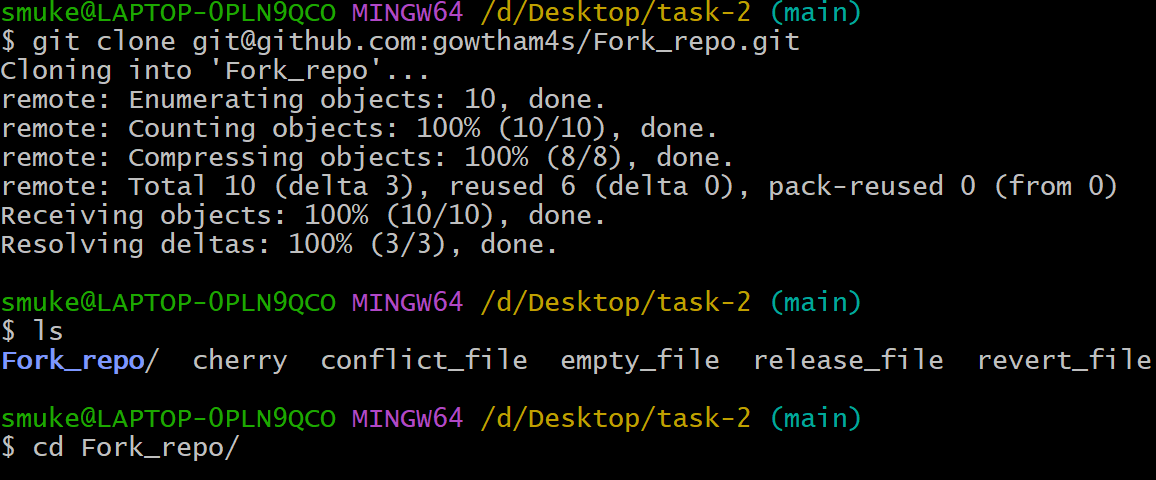


Step 2: Give repo name and click create fork





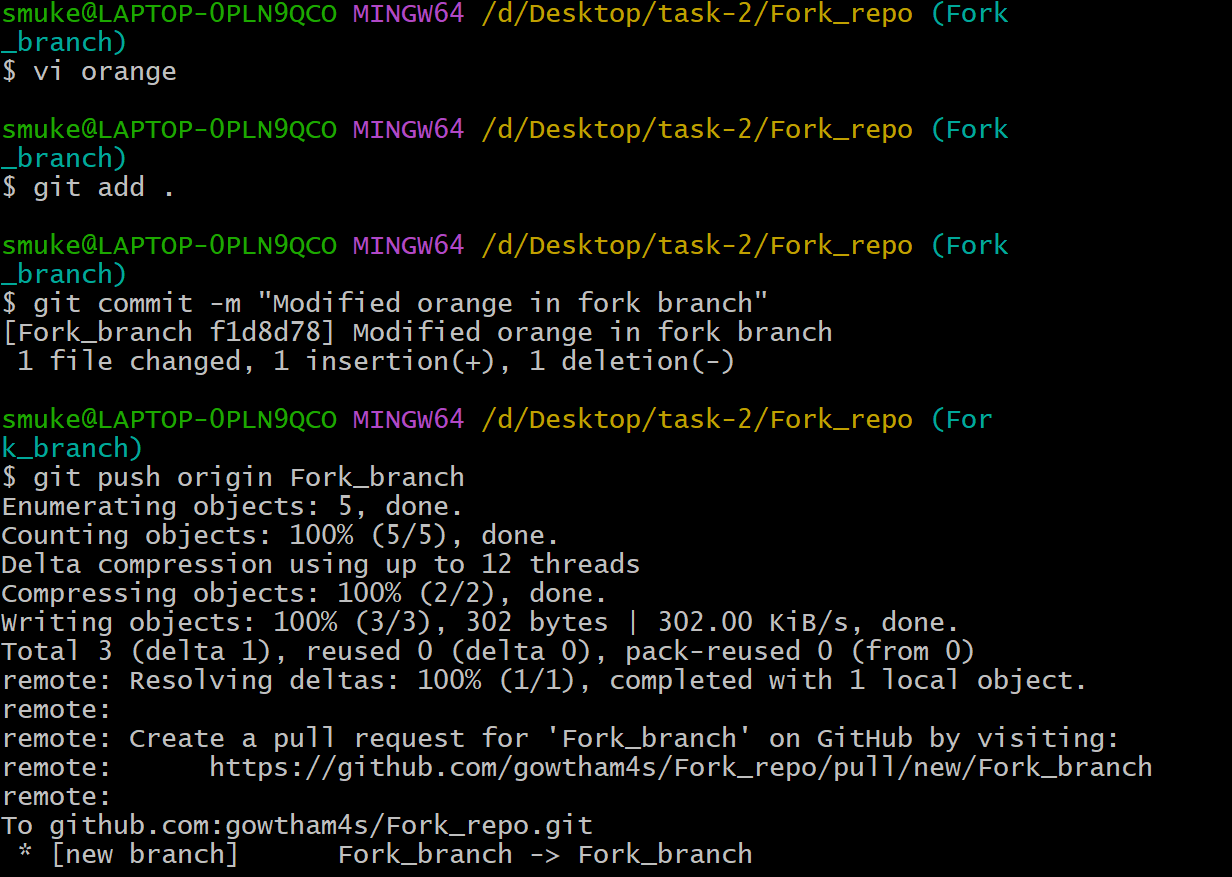
Step 3: Now in Git Enter **git clone fork\_repo\_url** and **cd Fork\_repo\_name**



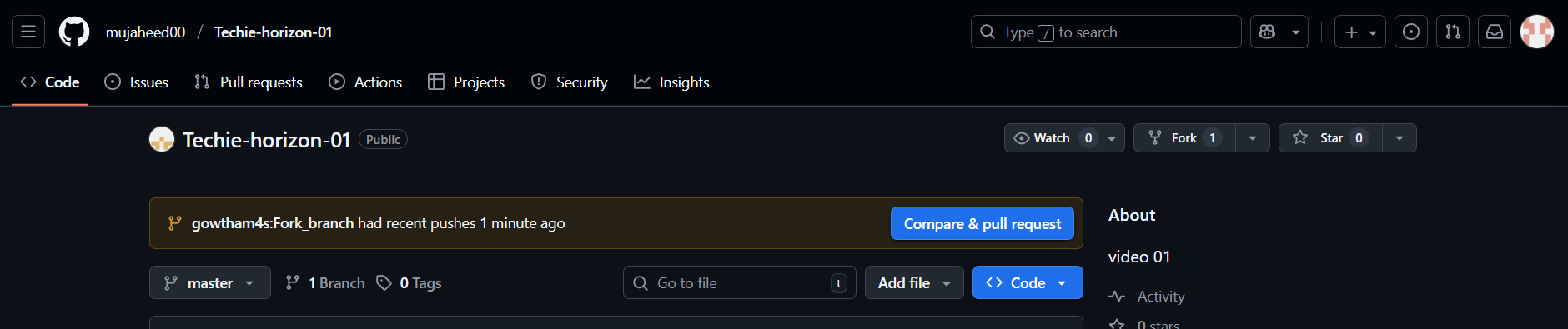
Step 4: Edit any file and push it.





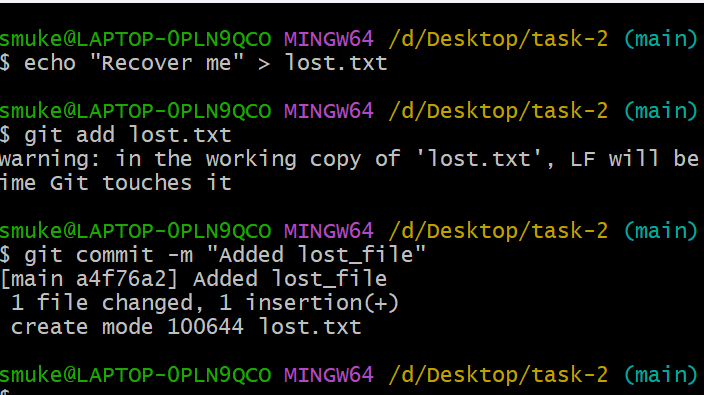


Step 5: Now go to original repo there you can see **compare & pull request**

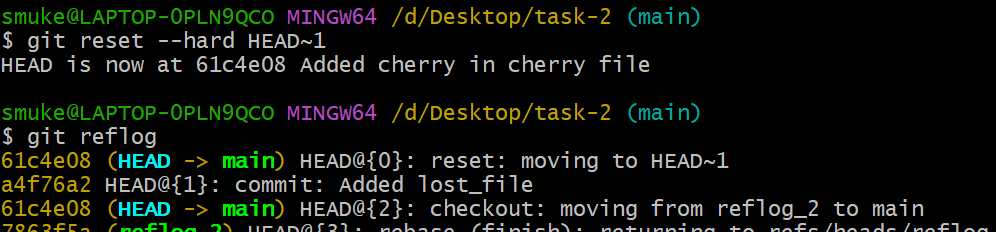
****

14. Recover lost commit.

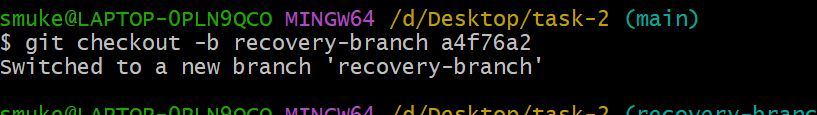
Step 1: Create a file and commit it.



Step 2: Use **git reset --hard HEAD~1** to revert one commit back and **git reflog** to list commit.



Step 3: Use **git checkout -b recovery-branch commit\_id** to create new branch at that commit



Step 4: Use **git reset --hard commit\_id** to move head back to that commit

