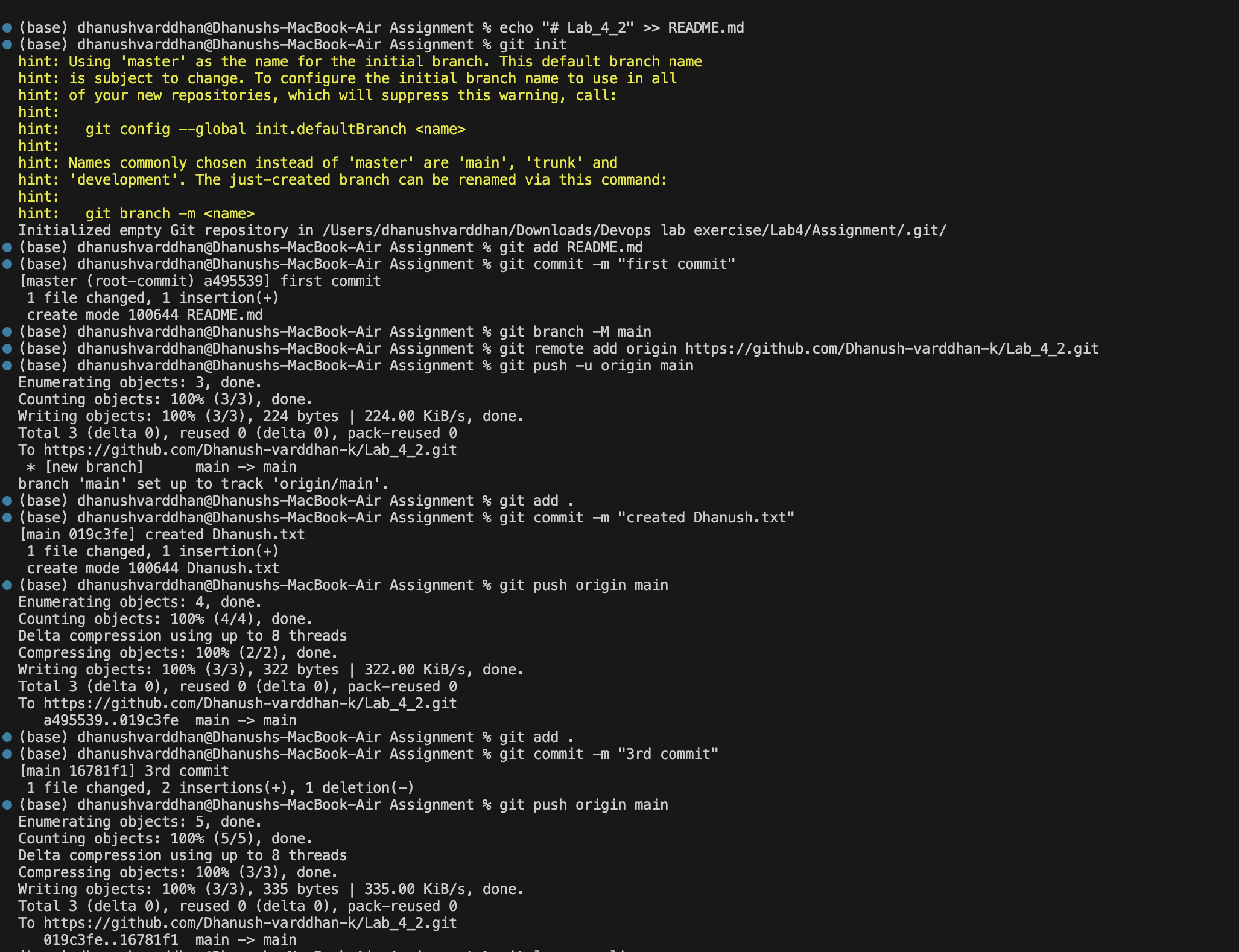
**DevOps Lab**

**Exercise 4**

1. **Exploring Git Commands through Collaborative Coding – Advanced Git commands.**

****

**Till here I have created a repo initialised it and then added a text file. I did changes for 3 times and have 3 commits. I have pushed the changes to remote origin also.**

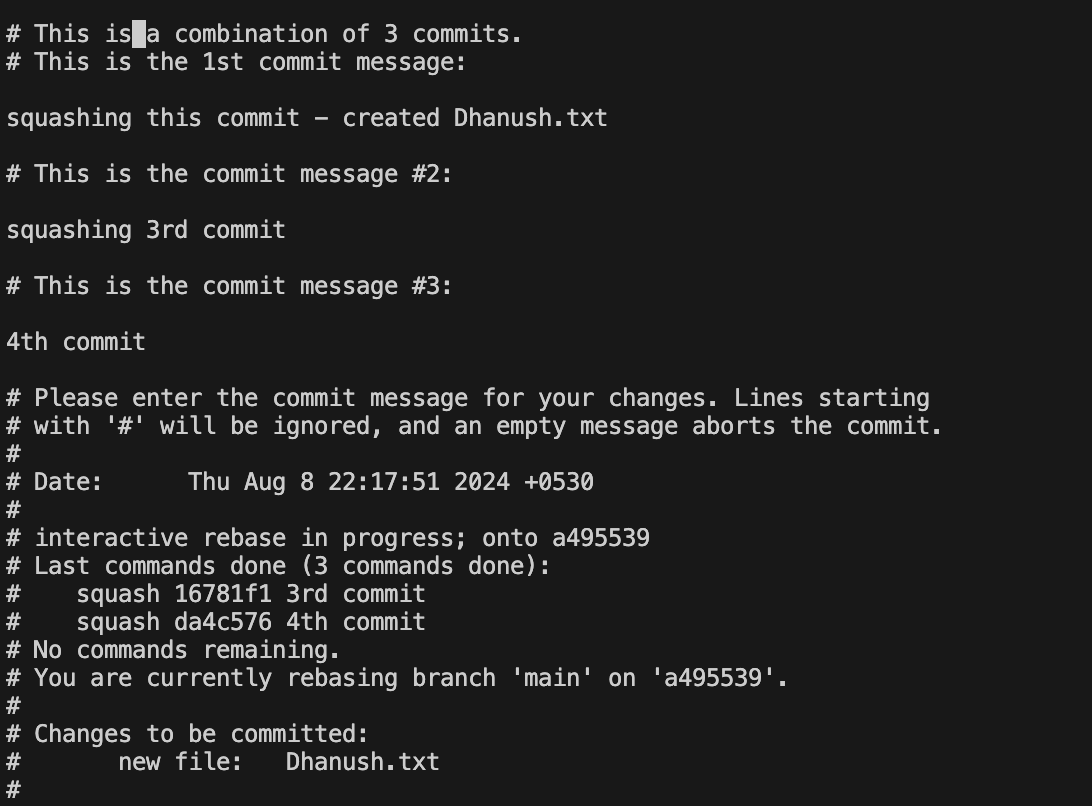
**A screen shot of a computer

Description automatically generated**

**Available 4 commits.**

1. **Use interactive rebase** to combine to multiple commits into one

* Use git rebase -i to modify the commit history.
* Interactive Rebase to Edit Multiple Commits
* In the interactive rebase interface, change pick to edit or squash as needed to edit messages or combine commits.



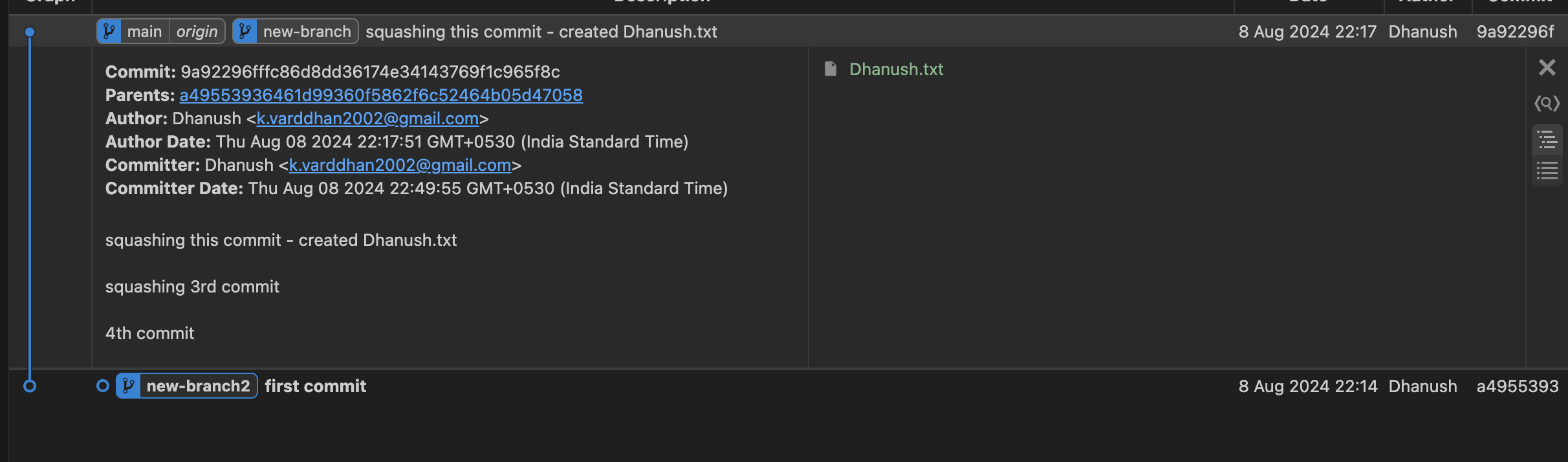
A screenshot of a computer

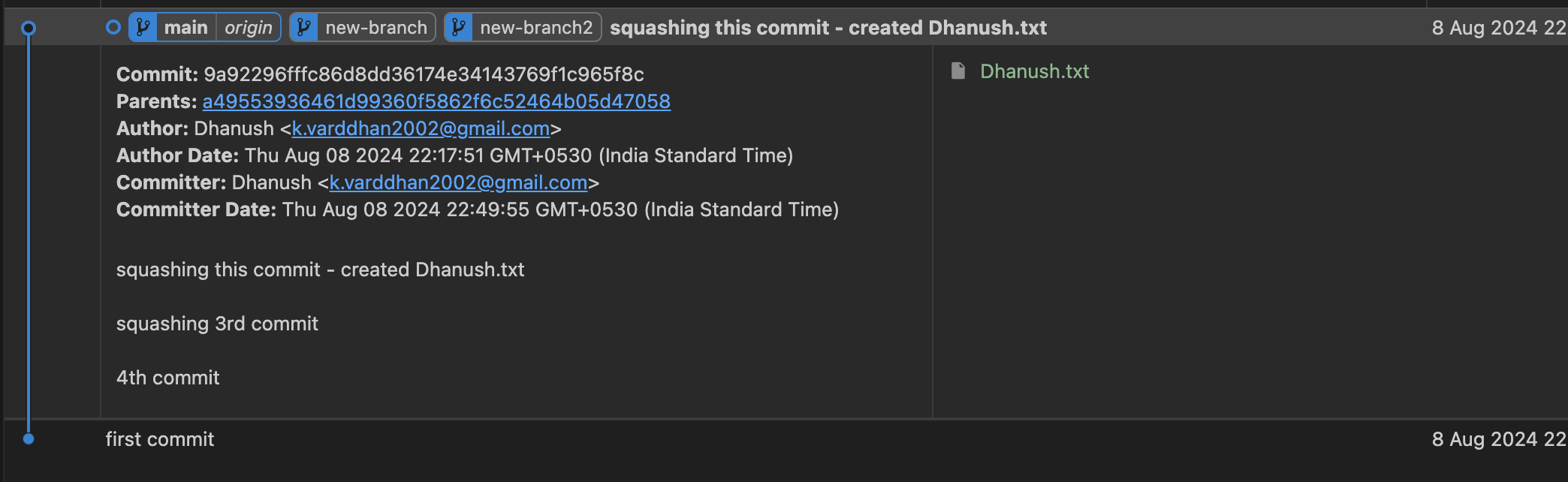
Description automatically generated

* Rebase onto Another Branch.
* Create a new branch from an earlier point in your commit history.
* Use rebase to apply the commits from main onto this new branch.

A screen shot of a computer

Description automatically generated





Created a new branch in first commit spot. Then rebase it with main branch.

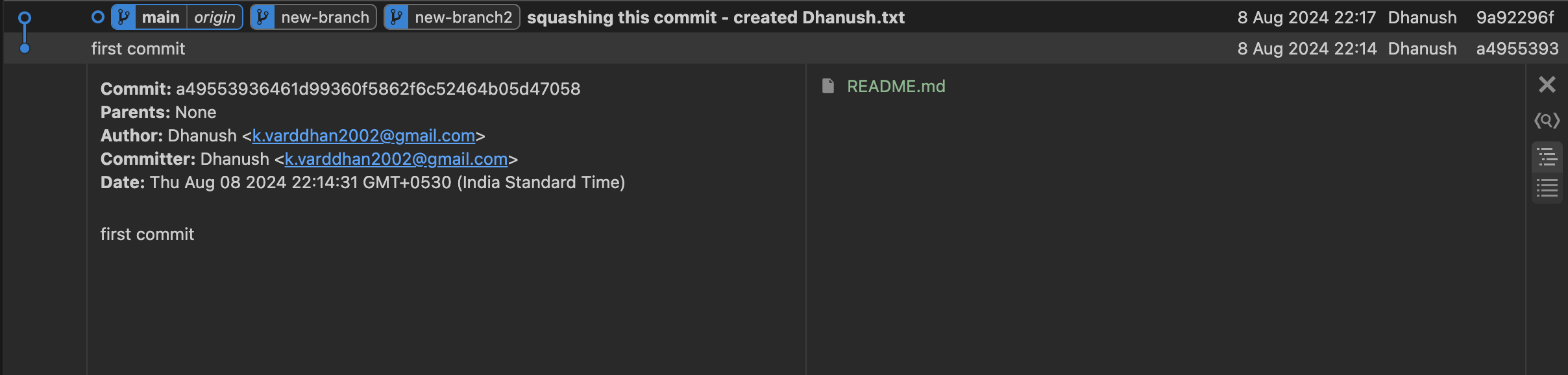
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

To make working tree clean.



Instead of 4 commits only two commits are there. This is effect of squashing.

1. **Stash Changes**

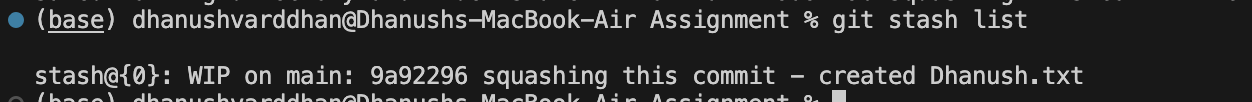
* Make some changes in your working directory.
* Stash those changes.



A screenshot of a computer

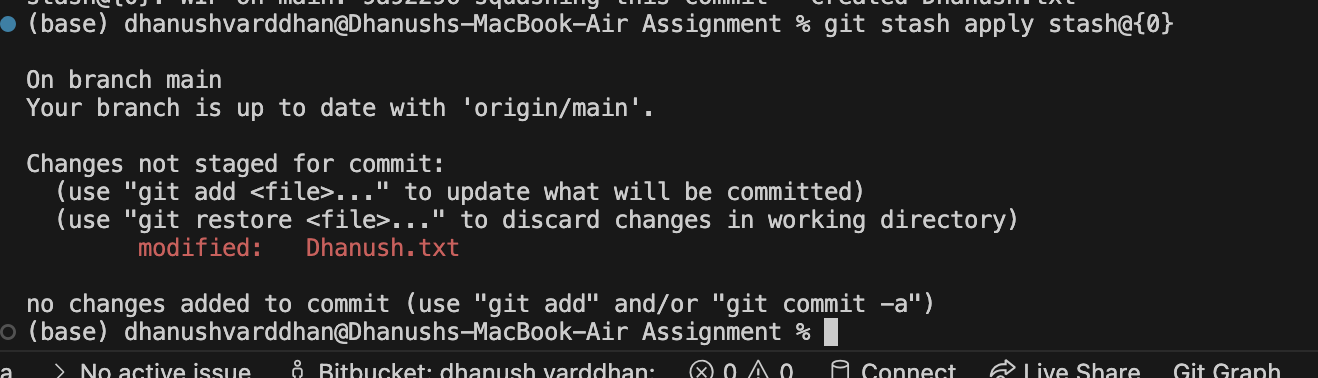
Description automatically generated

Added some text in Dhanush.txt



List of stashes present.

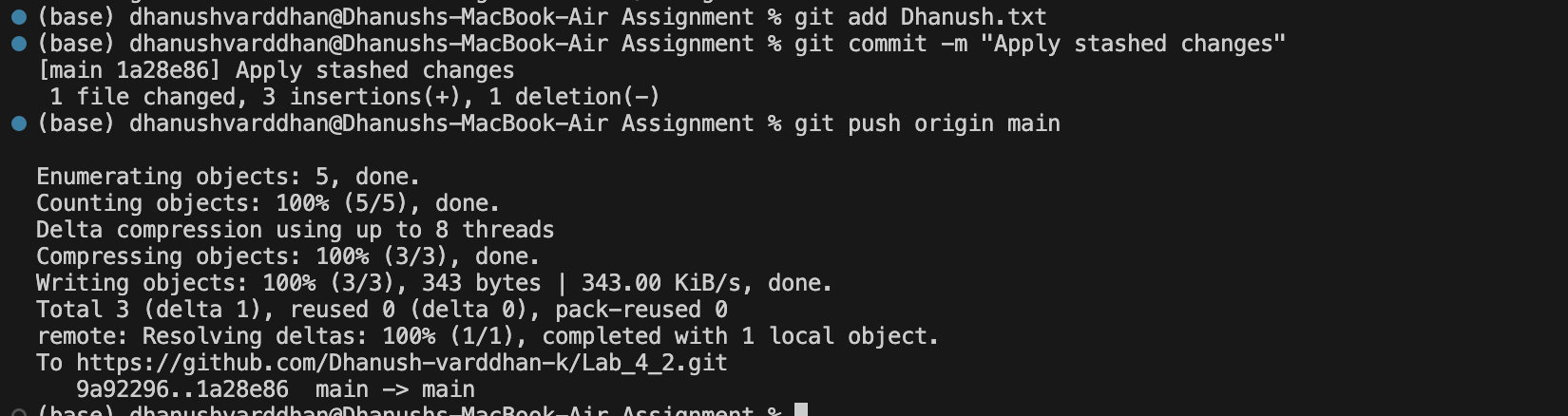
* then apply the stash

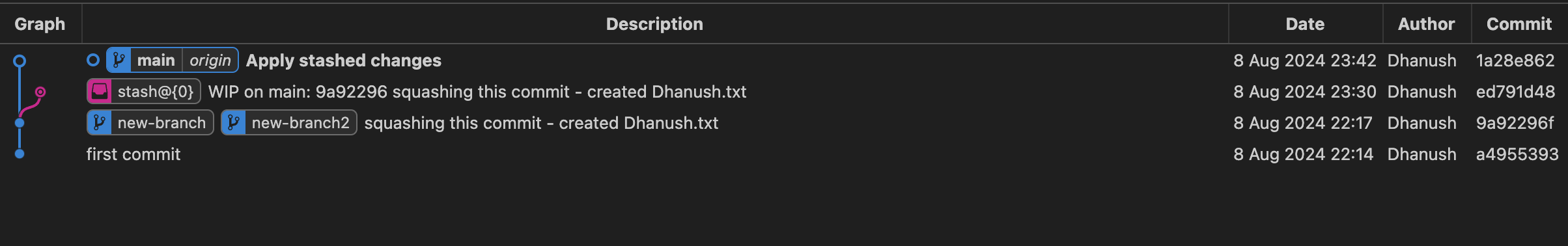


A black screen with white text

Description automatically generated

Now after this committing stashed changes in main and origin main branches.





1. **Revert and Reset**

* Create a new commit, and then use git revert to undo it.

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

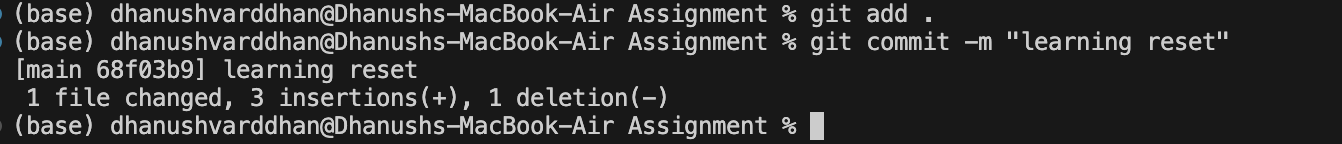
Description automatically generated

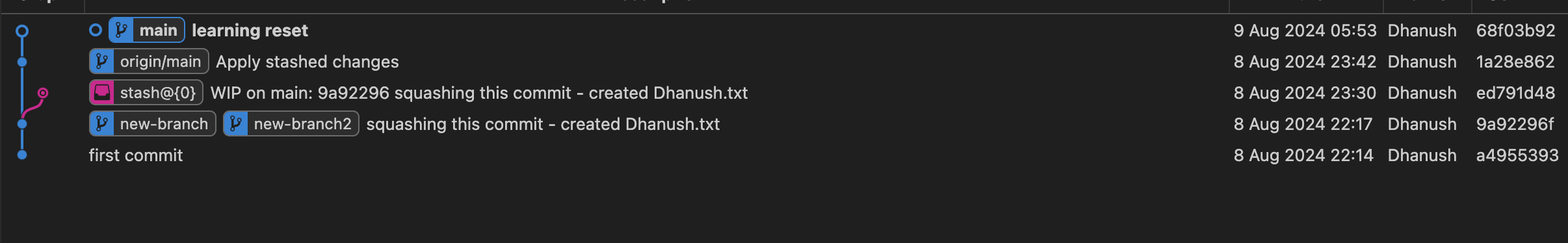
In revert commit is retained in history. Commit is still there but on top of it new commit is made and changes are discarded. The changes are reflected in local only. Not in remote.



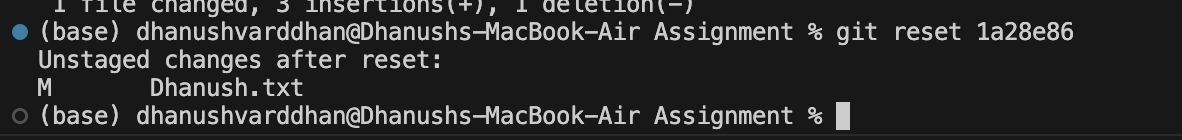
So we do “git push origin main”.

* Experiment with git reset to understand the difference between --soft, --mixed, and --hard.

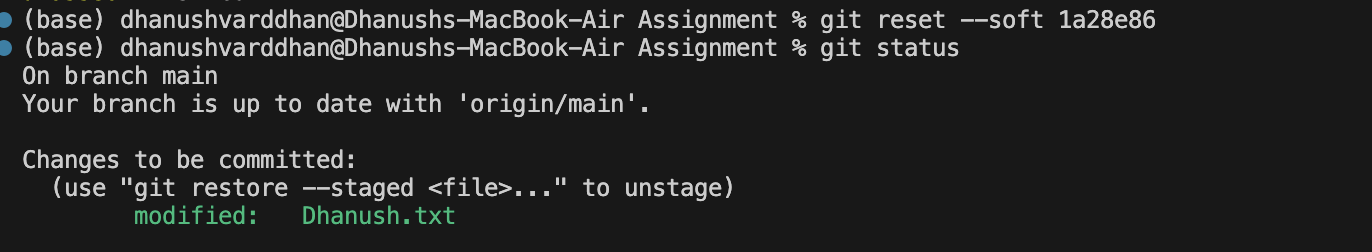




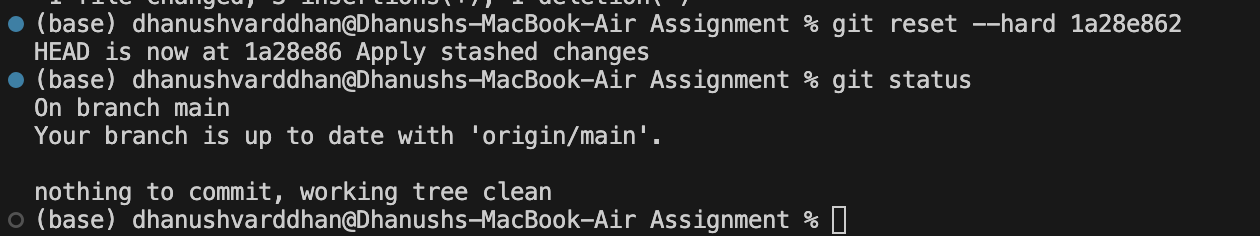
Dummy branch for working out reset is made.



Use “git reset \_\_\_\_\_\_”. The id for which you specify, above git commits would be gone. But changes are there in working area. Here we didn’t mention what type. So it takes mixed reset as default. Mixed reset removes commit and unstages it. Means changes you made would be, but they would not be commited and added. Changes would be in working area and not in staging area.



In soft reset changes would be in staging area and only commit would be removed.





In above image hard reset has been performed. This command forces head to get back to the commit and deletes everything after that.

Difference between reset and revert:

In reset commit itself would be removed from history but in revert the commit is retained in history.

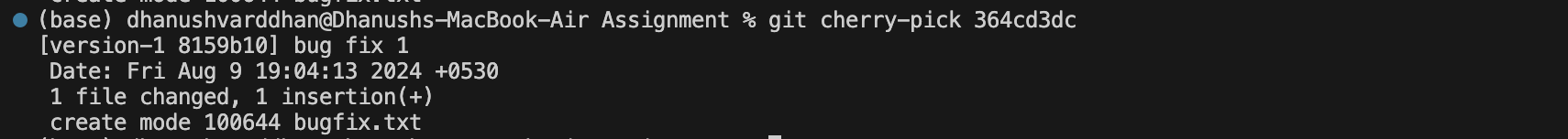
1. **Cherry-Pick a Commit**

**Mainly we use cherry-pick in bug fixes where you want to place the bugfix commit in all the versions of the branches.**

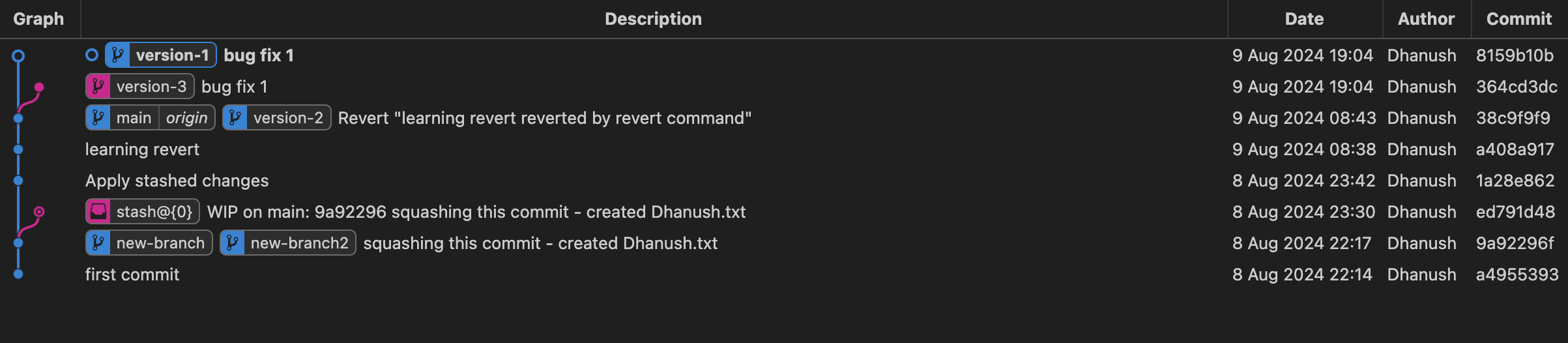
**It is also used when we accidently made commits in wrong branch and we want the commit to be in another branch etc.**

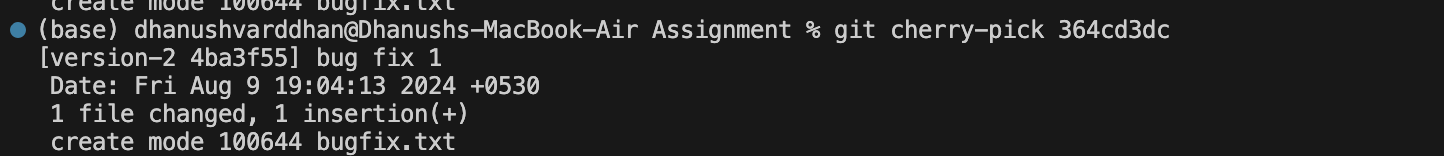


**Before cherry-pick.**

****

**Fixing the bug in version 1 using cherry-pick.**

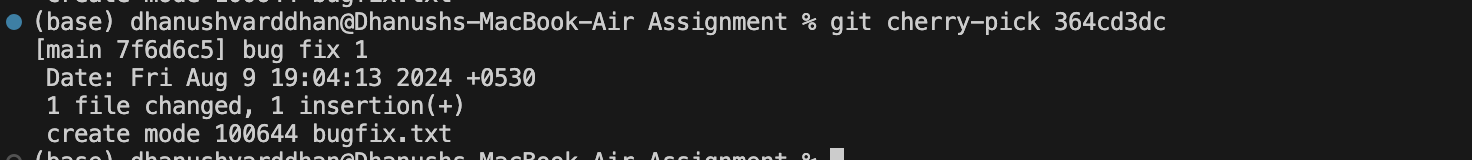
****

****

**Fixing the bug in version 2 using cherry-pick.**

**A screenshot of a computer

Description automatically generated**

****

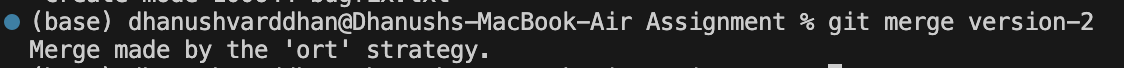
**Fixing bug in main branch.**

**A screenshot of a computer

Description automatically generated**

**Note: Cherry pick creates new commit always.**

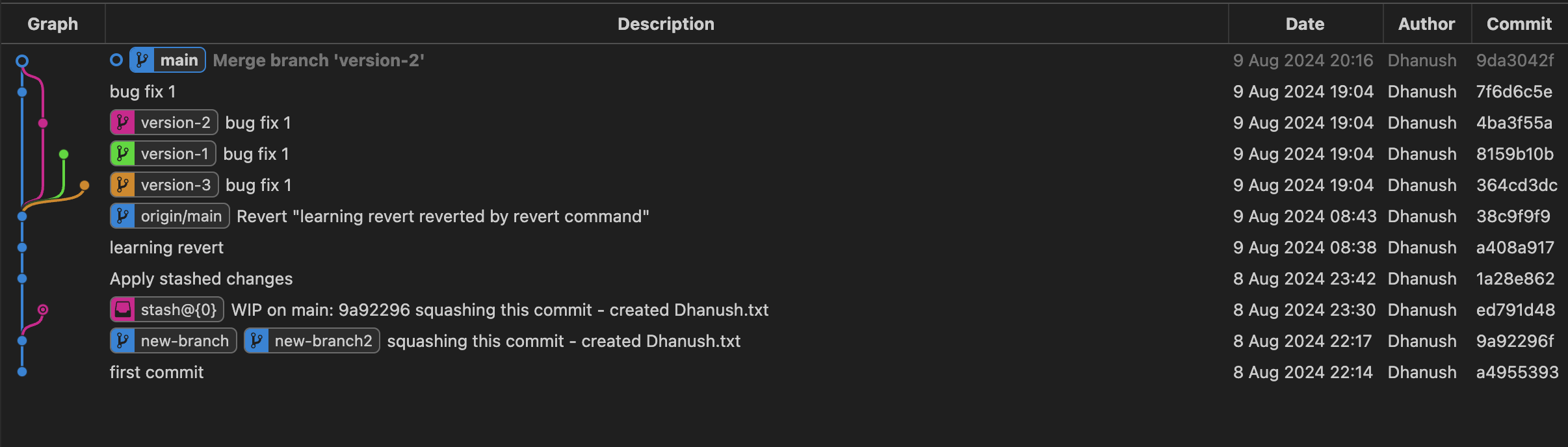
* Resolve Conflicts During Cherry-Picking
* Resolve Conflicts During Cherry-Picking



Resolving conflicts.

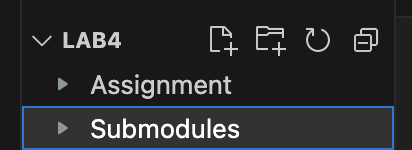
A screenshot of a computer

Description automatically generated



1. **Working with Submodules**

**My file structure:**

****

**Already Assignment has been initialised and it has a remote repository.(Lab\_4\_2).**

**Now I made a folder named submodule and initialised with another git repo.(submodule\_learning).**

**Now Lab4 is initialised with another git repo. (Lab4).**

**Now after pusing changes in Lab4 remote repo:**

**A screenshot of a computer program

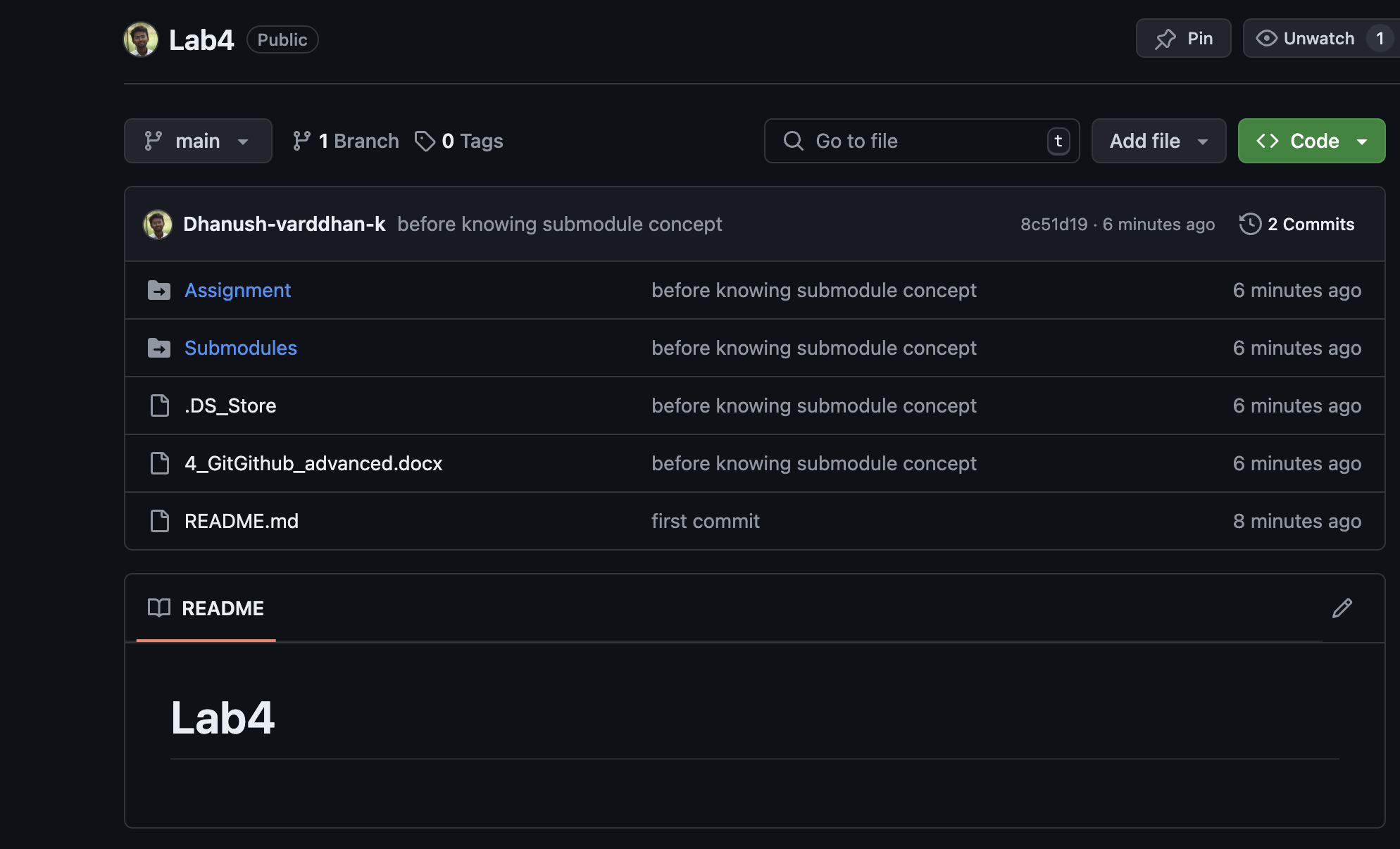
Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

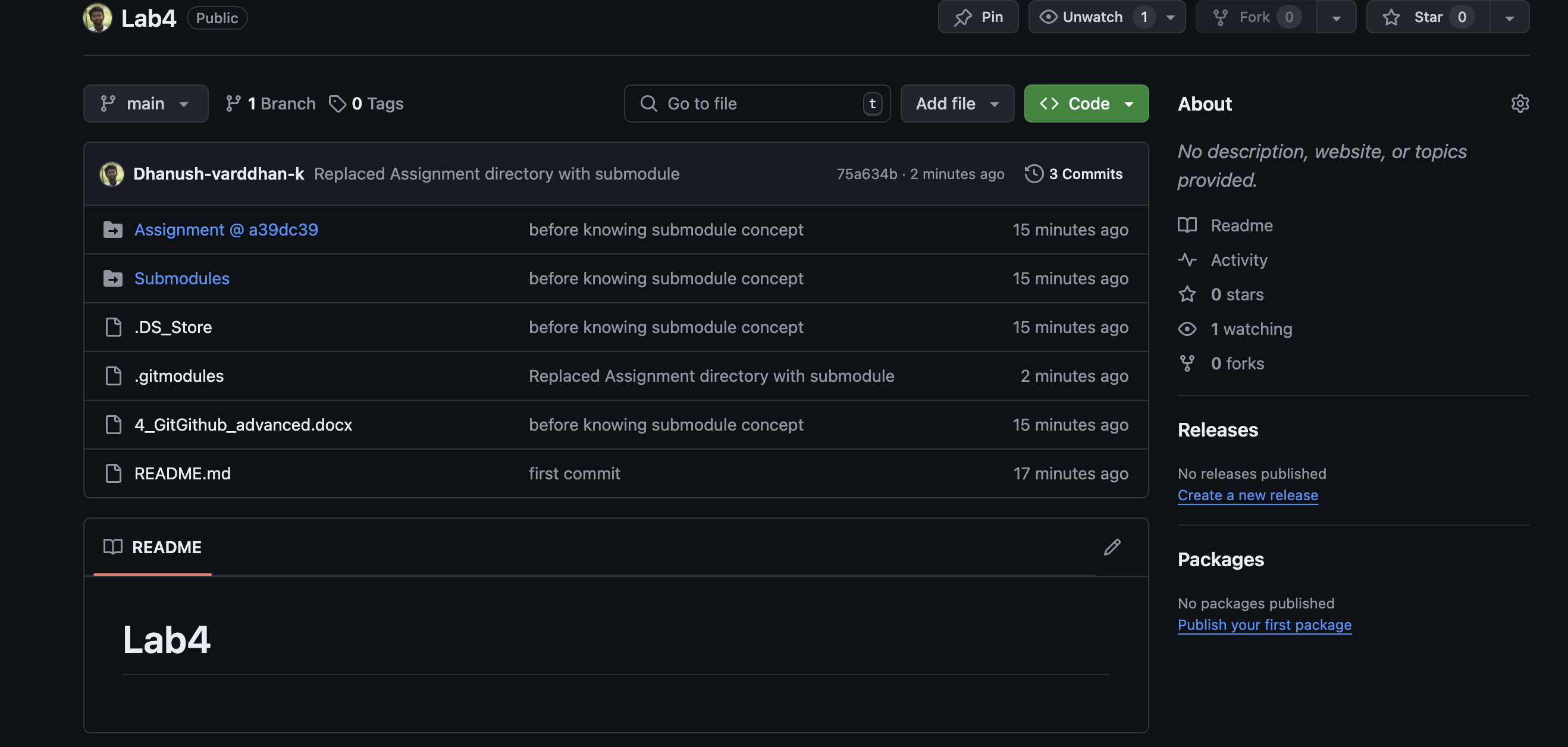
**A black rectangular object with a black border

Description automatically generated**

* In your repository, add another repository as a submodule.

A screen shot of a computer

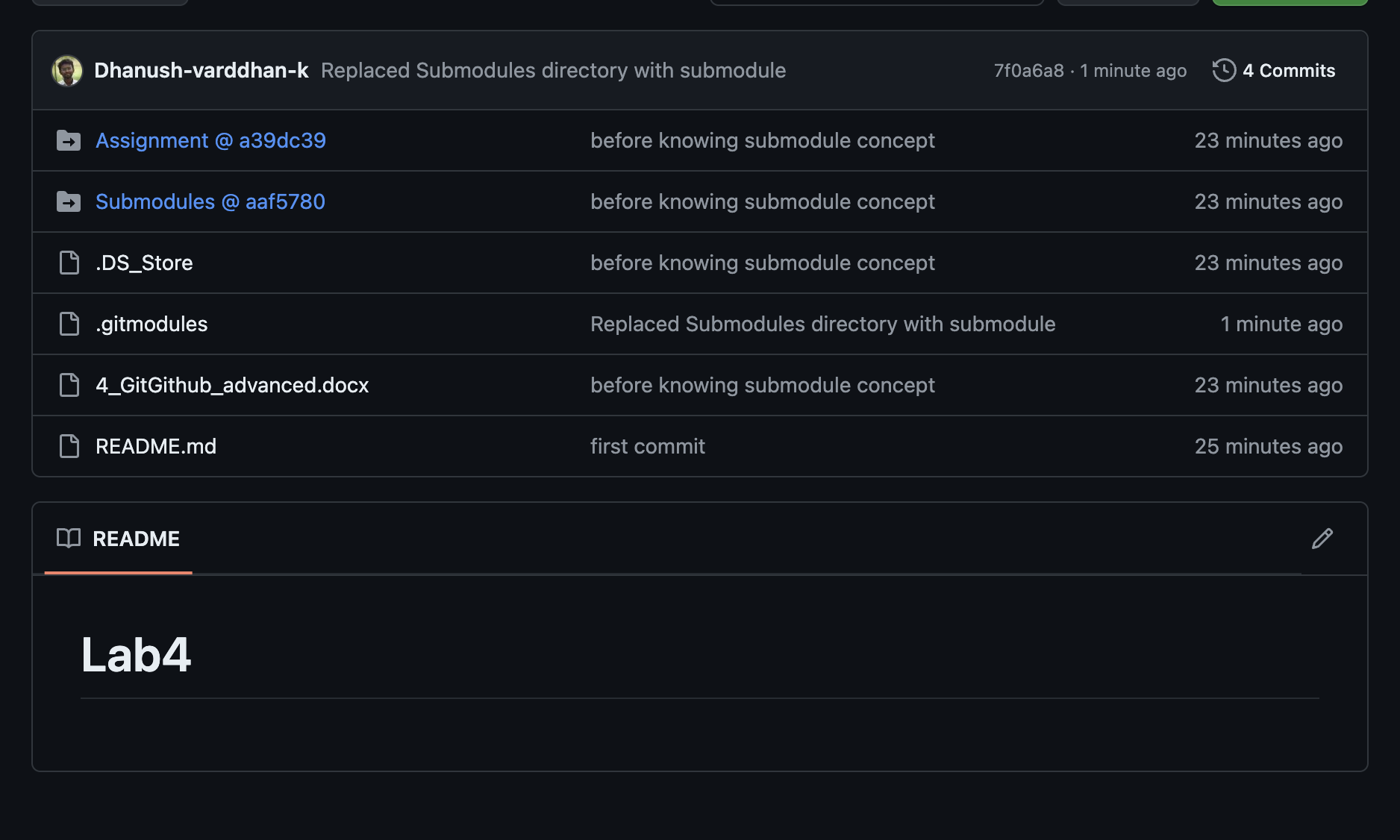
Description automatically generated





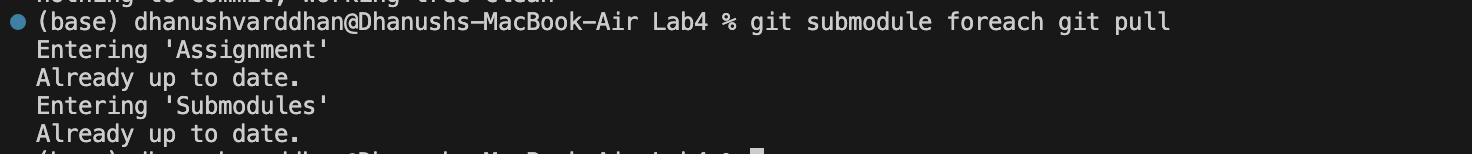


I faced a error and this is how I cleared it.





Checking the status.



For pulling changes from remote.

* Clone a Repository with Submodules
* Update and Synchronize Submodules

1. **Git Hooks**

* Create a Pre-Commit Hook that prevents commits with “TODO” comments
* Use the below script

#!/bin/sh

if grep -r "TODO" .; then

echo "Commit rejected: please remove TODO comments."

exit 1

fi

* Make the script executable and try committing a file with a TODO comment.
* Explore other types of hooks like post-merge, pre-push, or post-checkout, and create simple scripts for them.