Github Session 10

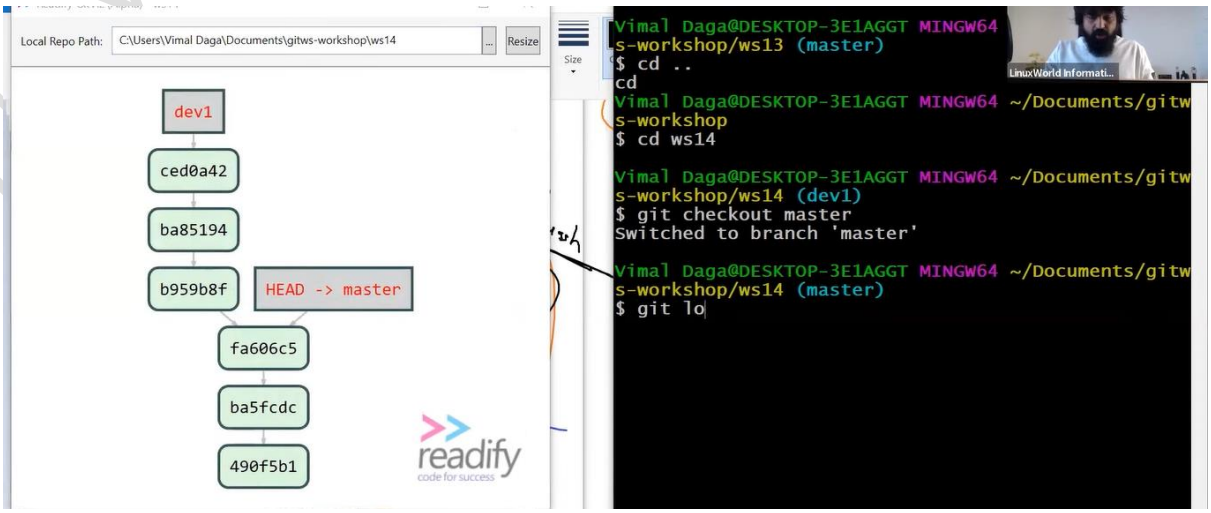
Summary 05-03-2022

- Suppose we have made multiple commits, each commit have its own history and at a point of time we will have a huge amount of histories stored including the unwanted histories also.

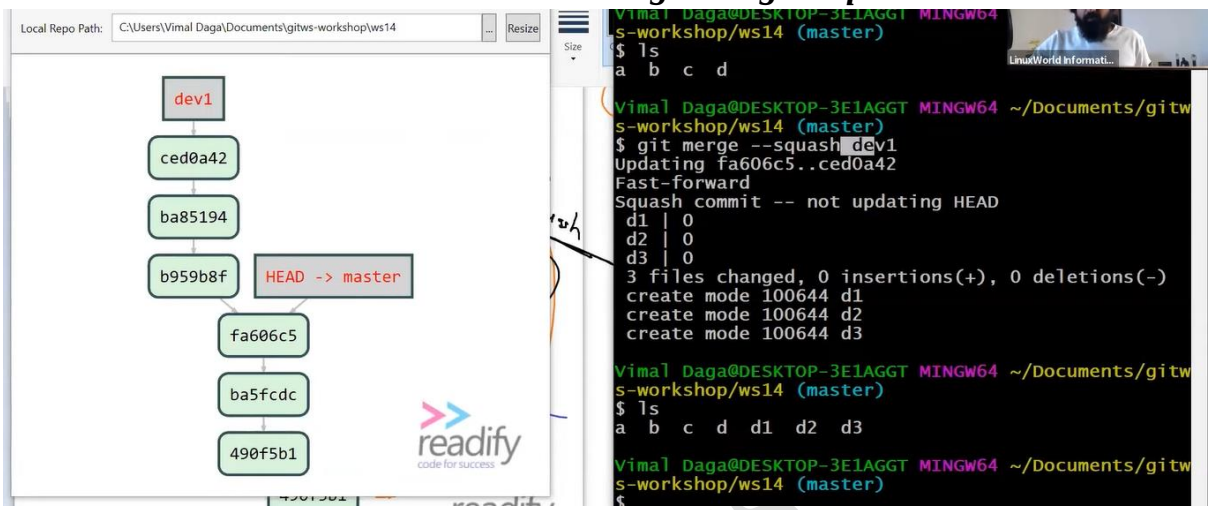
- If we want to take only relevant history while merging two branches, then **Squashing commits** allows you to create a cleaner and more concise commit history by combining multiple small, related commits into a single commit.

- When collaborating on a project, having a clean and focused commit history makes code reviews more manageable. Instead of reviewing numerous small commits, reviewers can focus on reviewing a smaller number of more meaningful commits, each representing a logical change or feature.

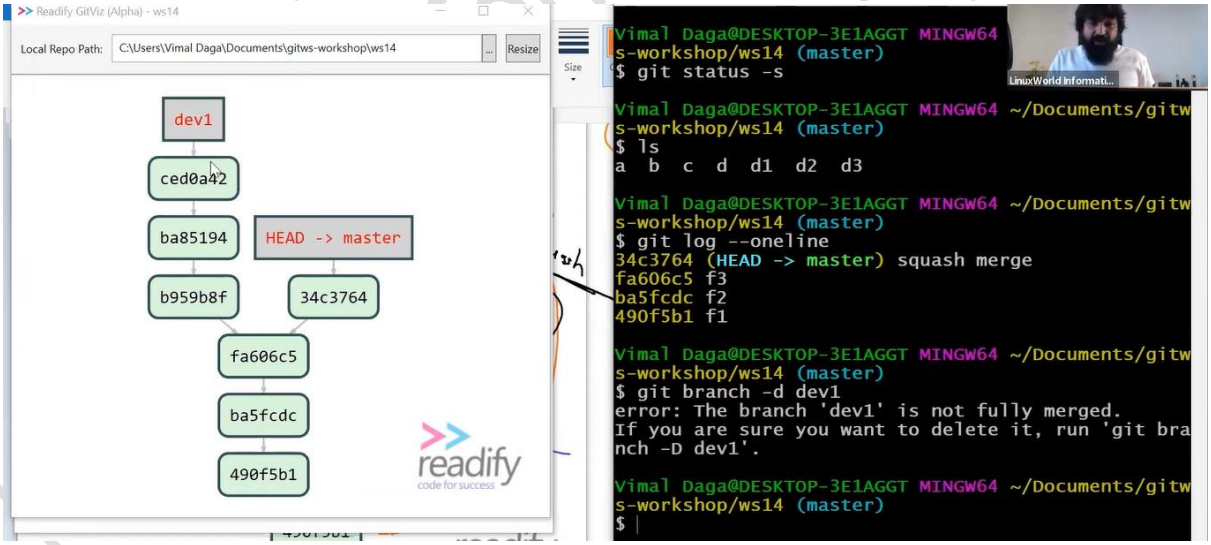
- To understand better we can see the example



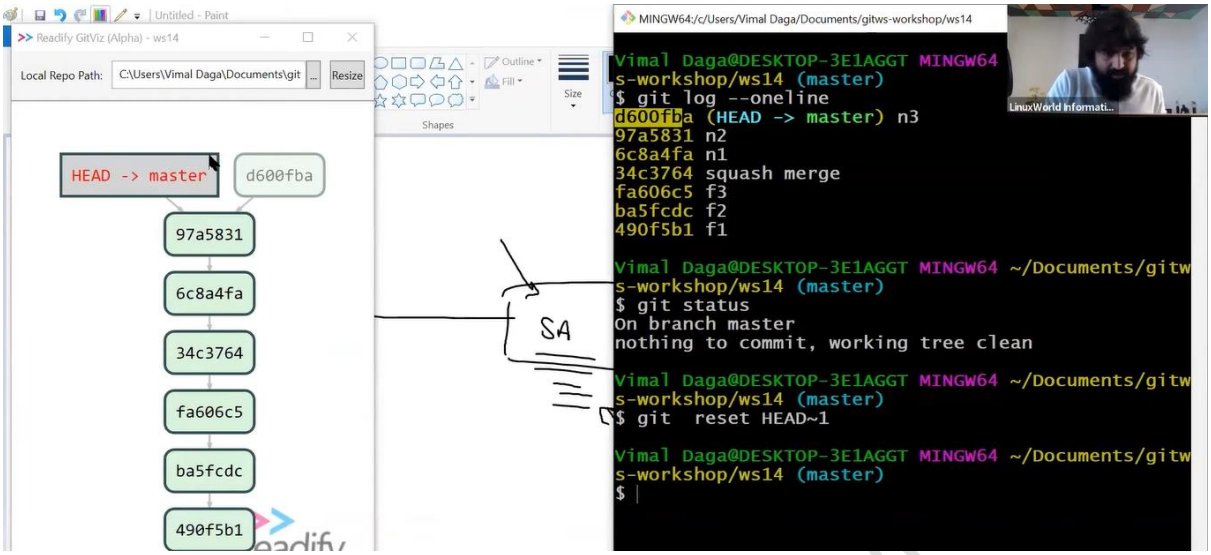
- Here we can see that the master branch has 3 commits and the dev branch have other 3 new commits
- Now we want to merge them without taking the unwanted history, so for that we will use the command ***git merge –squash dev1***



- We can see that initially the master branch had only 3 commits(a b c d) and after merging we have the dev branch commits also in the master branch(a b c d d1 d2 d3).
- All of the dev branch changes combines and make a single commit and will merge to the master branch when we do squashing.

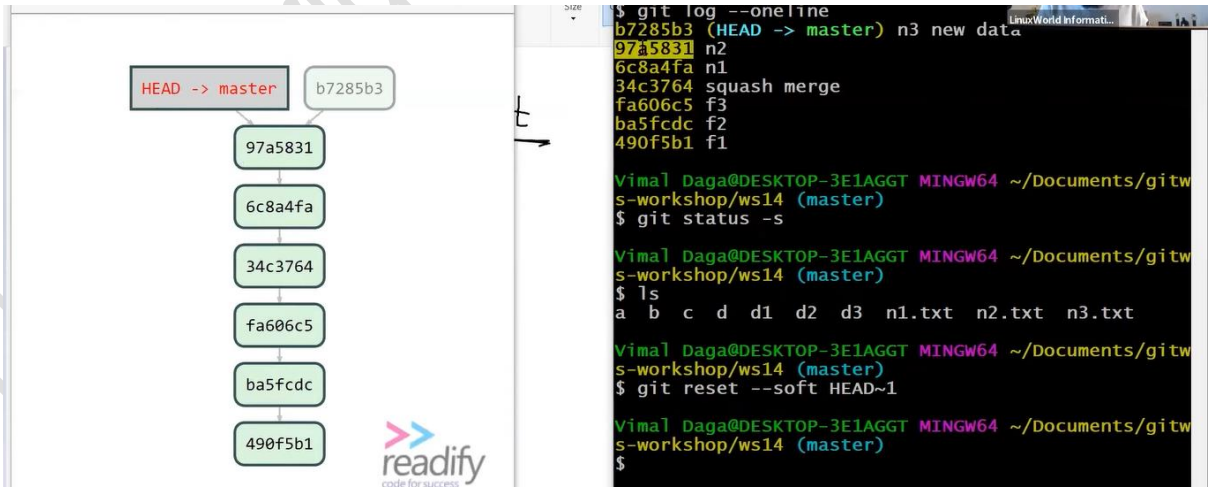


- To take back a file from the commit area to the staging area we use the **reset** for it ***git reset HEAD~1***
  - This command will remove the last commit from the commit area and will take it to the working area.

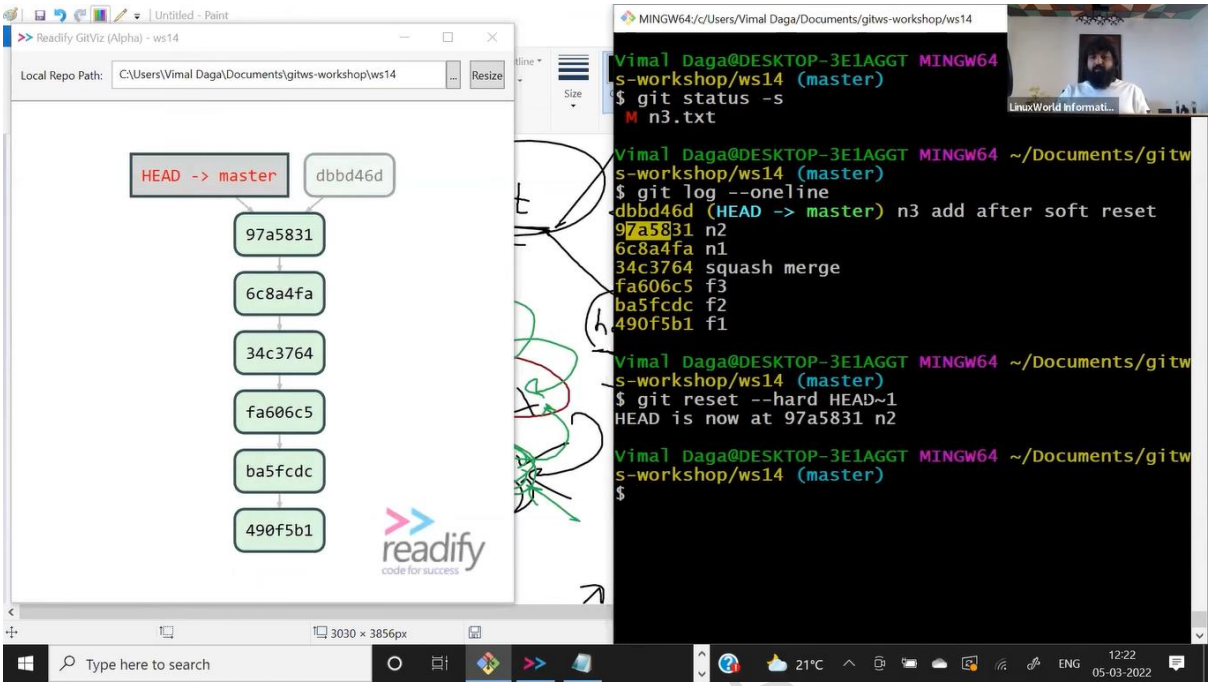


- A mixed reset is the default behaviour when no option is specified. It moves the HEAD to the specified commit and updates the staging area to match the state of the specified commit, but does not change the working directory.

- A soft reset moves the HEAD to the specified commit, leaving your working directory and index (staging area) unchanged. This means that the changes in your working directory and staging area remain intact, but the commits are "uncommitted" and moved to the staging area. For soft reset use the ***git reset –soft HEAD~1***

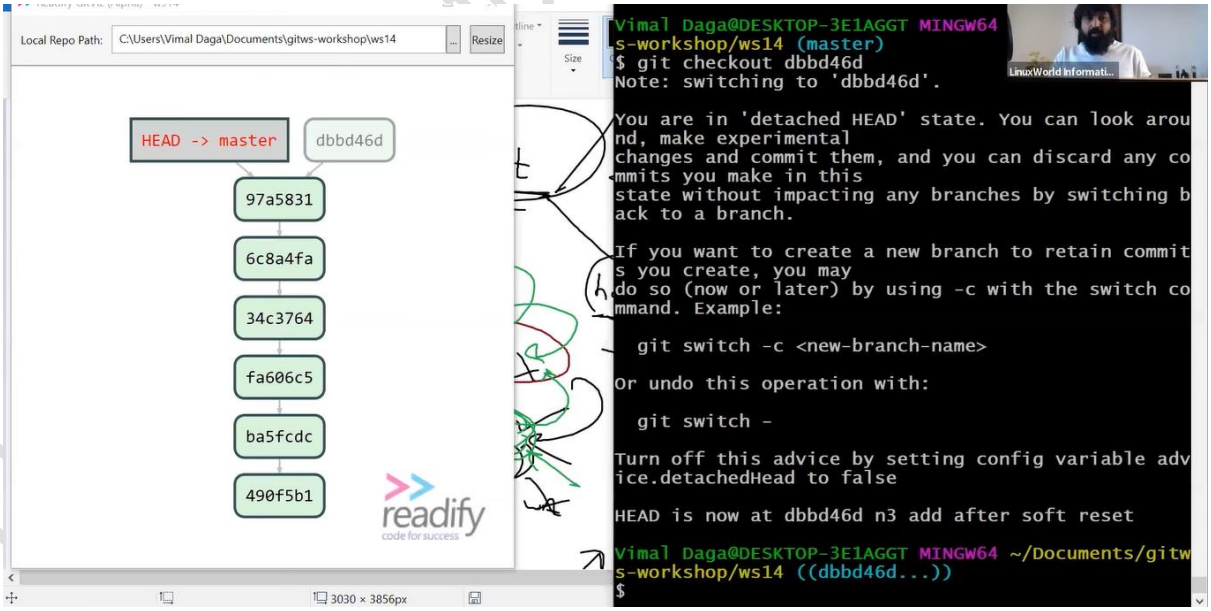


- A hard reset moves the HEAD to the specified commit and resets both the staging area and the working directory to match the state of the specified commit.
- The hard reset removes everything from the working area also.

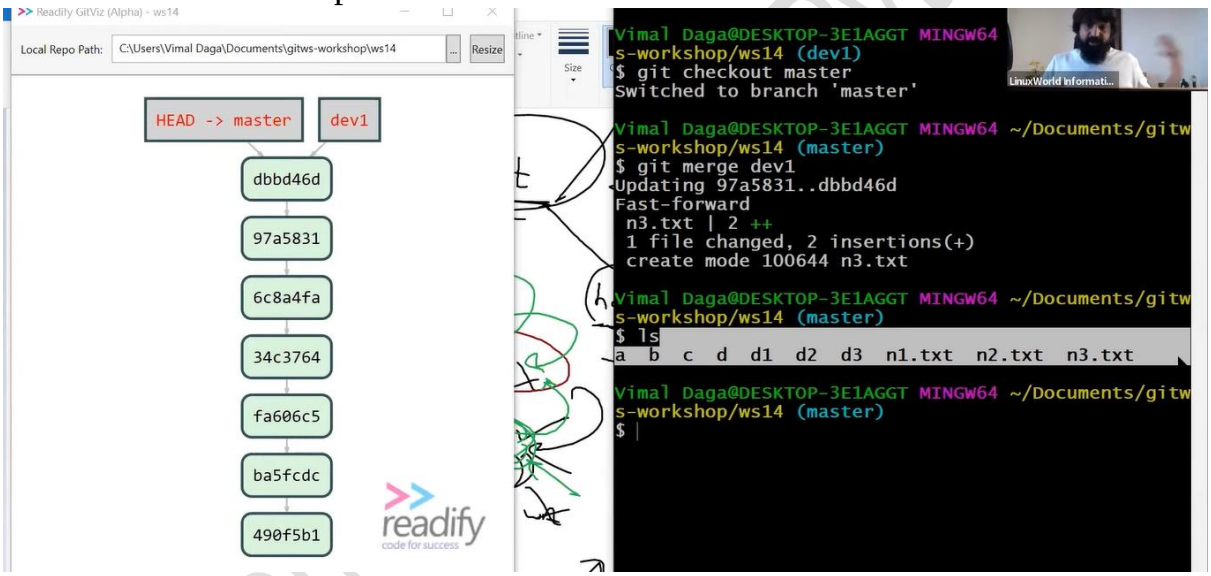


- The orphaned commit is removed from the timeline but git internally keeps it for a safe side and we can restore it back.

  - To restore back the deleted or the orphaned commit, first move the HEAD to that commit then create a new branch at that head.



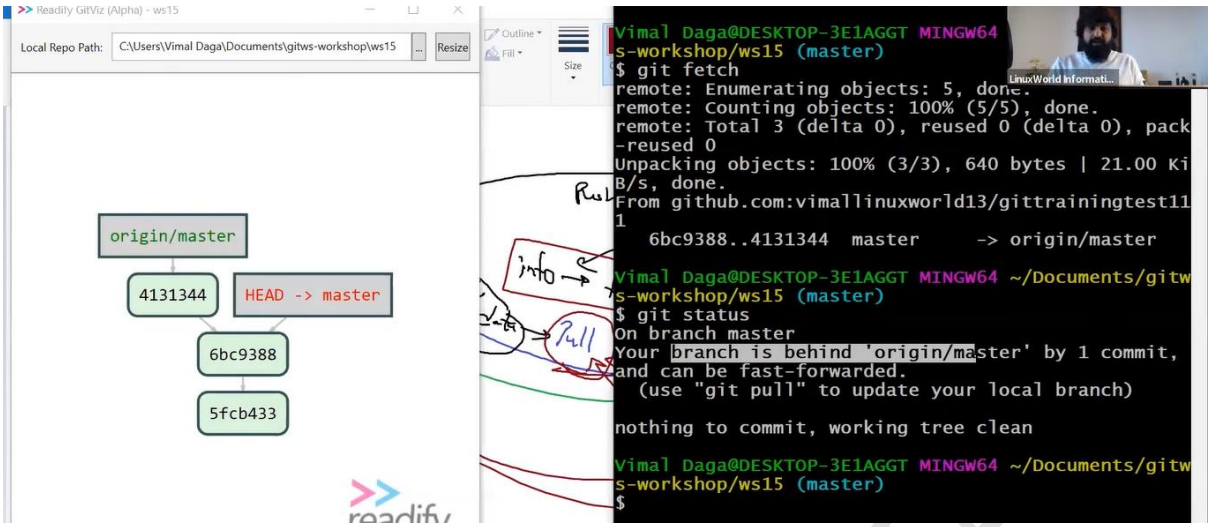
- Once the new branch is created, merge it back to the master branch to restore the orphaned commit into the timeline.



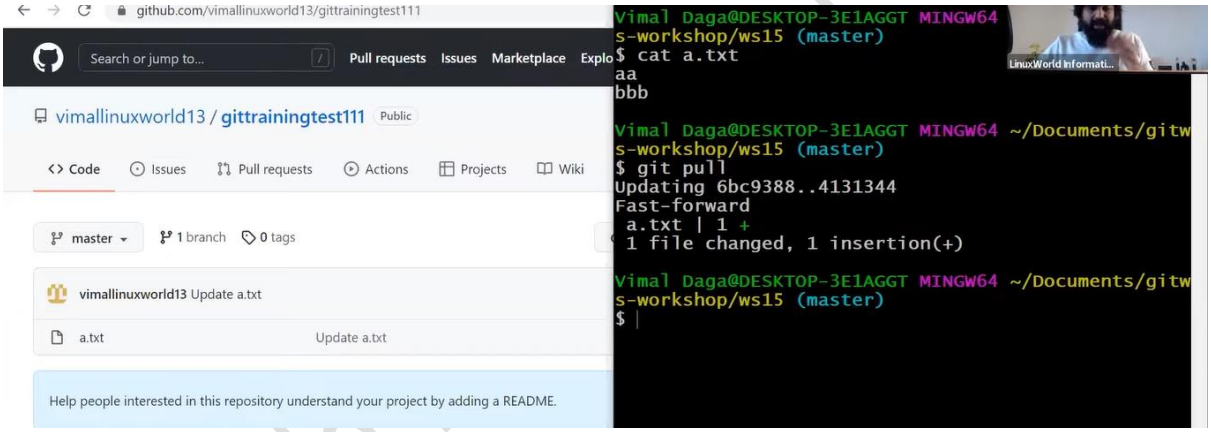
- Whenever we do some changes in the github directly, we doesn't get updated in the local system or we can say that local system will not get to know about any changes done on the github.

- To deal with this problem or to get information about any changes that happened in the github we can use the **fetch** for it. ***git fetch .***

- Fetch will give you only the information about any update or the change but will not give you the data.

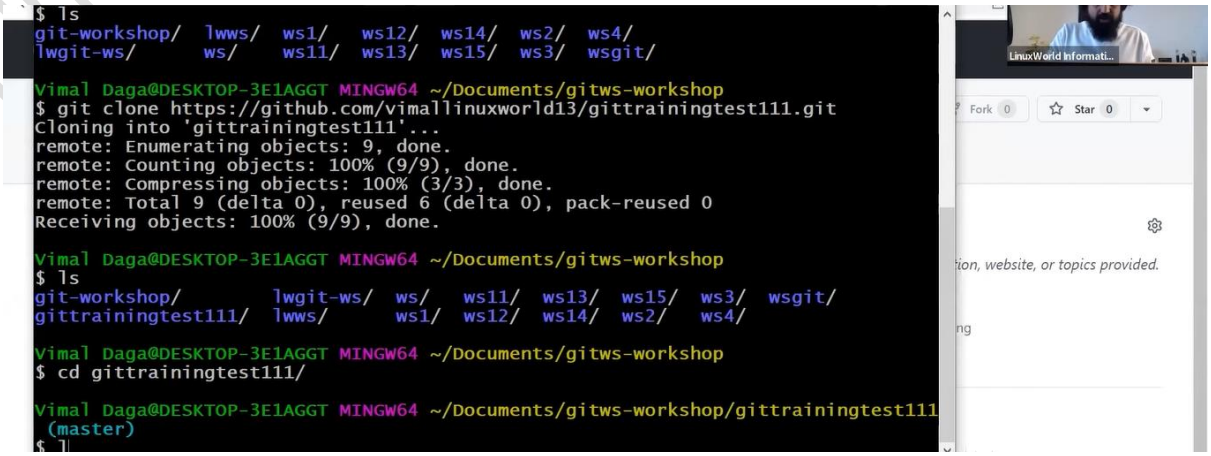


- To get the data also, we uses the command ***git pull*** for it.



- Cloning a repository allows you to start working with a copy of the project, preserving its entire history and allowing you to make changes locally.

- To clone any repository we can use the command ***git clone <repository url> .***



- **git clone** is used to create a copy of an existing repository whereas **git pull** is used to update your local repository with changes from a remote repository.