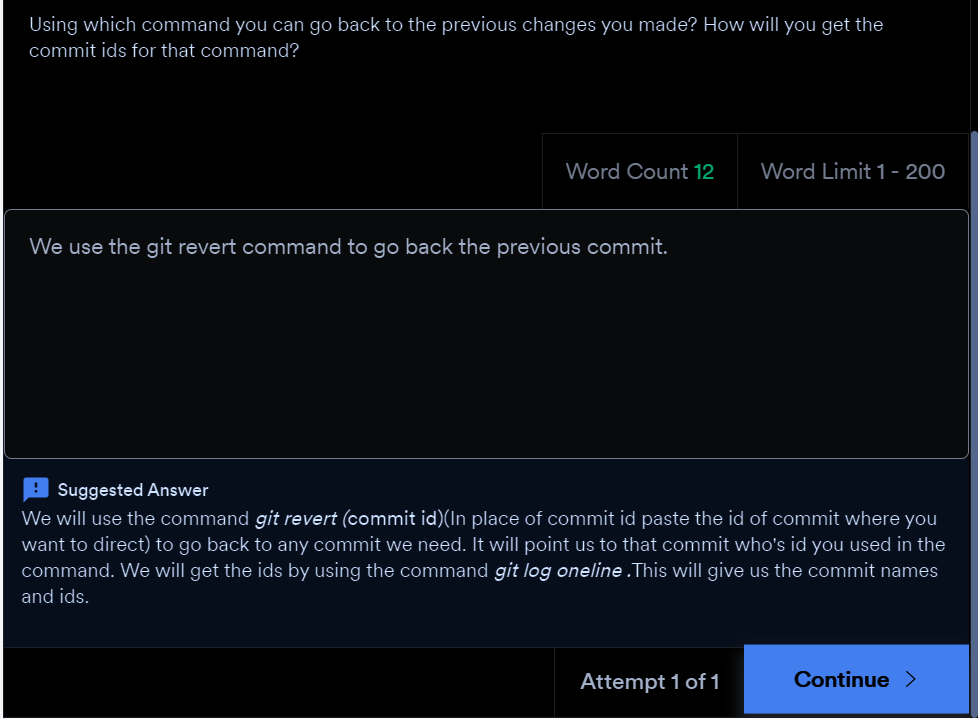
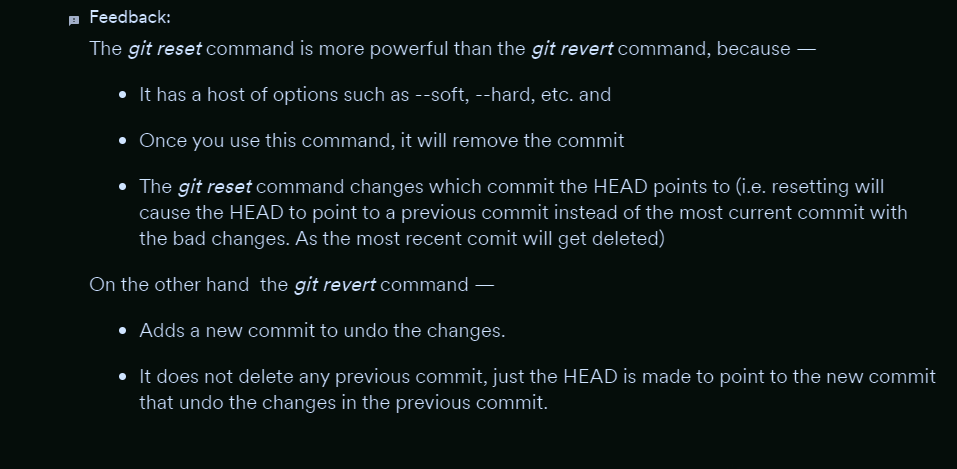


* **Some important git commands that are used very frequently:**
  + **git status**: This command will display the state of the working directory and the staging area. In other words, it lets you see the changes that have been staged and the changes that haven’t been added to the staging area.
  + **git log**: This command shows you the commit details. It lists out the commits made in the repository in reverse-chronological order, that is, the most recent commits show up first. It shows commits with the following details:
    - The commit ID or SHA
    - Author’s name (who made the commit)
    - Date and time
    - Commit message

**More on the ‘git remote add origin url’ command**

* + Using this command, you can add a new remote repository to your local repository. To do so, you should use the **git remote add**command on the terminal, in the directory your repository is stored at. **The git remote add** command takes two arguments:
    - A remote name, for example, origin (it can be any name)
    - A remote URL, for example, <https://github.com/user/repo.git> (the address of the repository on your GitHub account that you want to link your local repository to.)





* **git checkout**-- **filename**: This command helps you when you have made some modifications to your file, say, file1, and you haven’t added those changes to the staging area or development history. Using this command will take you back to its previous state.
* **git revert commit\_id**: This command helps you when you have already staged your files and committed the changes and want to go back to your previous commit. Instead of removing the previous commit from the project history, git revert will figure out how to undo the changes introduced by the previous commit and appends a new commit that reverts the content and the changes introduced by the previous commit. This prevents git from deleting any commits, which is important to maintain the integrity of your revision history
* **git reset**--**hard commit**\_**id:**This command helps you when you have already staged your files and committed your changes, want to go back to your previous commit, and want to remove your present commit.Specifically, this command tells git to think of <commit\_id> as the latest commit in your history and revert any file to what they were at <commit\_id>. Any commit that you made after <commit\_id> will no longer be in your history. In other words, git will dispose of any commits that happened after <commit\_id> as if they never took place.

**Glimpse of the next segment:**

The next segment is a summary of all that you have learnt in this session so far.

Worried about what will happen if you mistakenly delete the file from your local system?

No worries git hub is here. You can bring back the file from your git repository using a simple command.

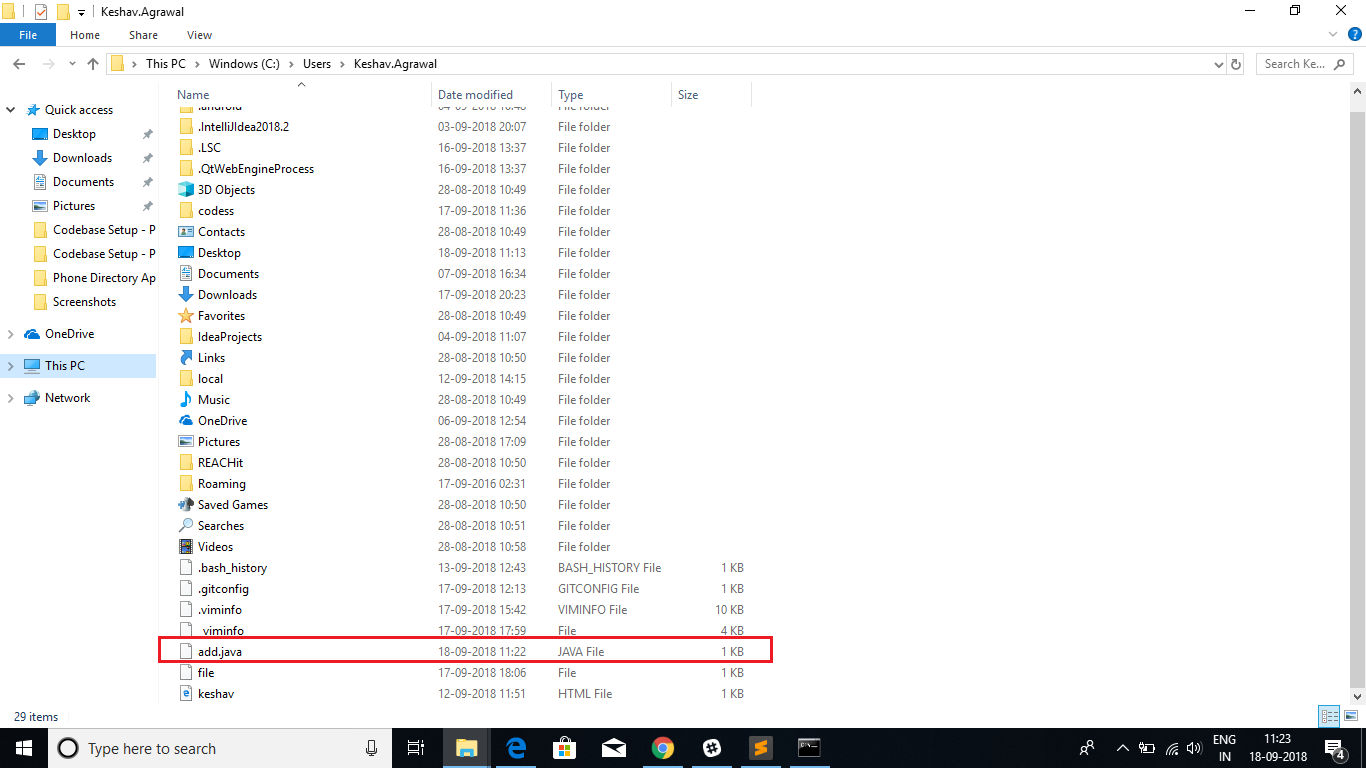
***git checkout***  it will show all the files available for checkout(in windows) then write the command

***git checkout (file name)***

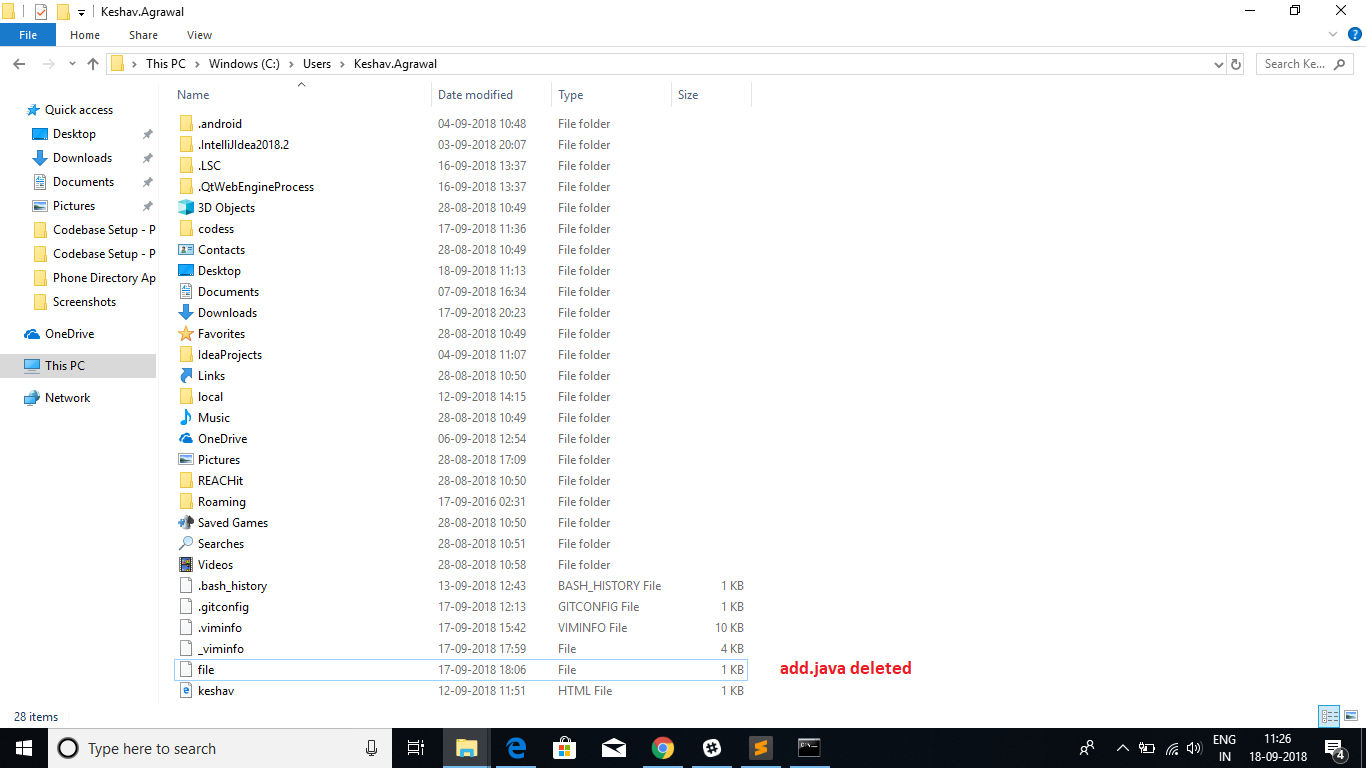
For eg. in our case, it will be

***Git checkout add.java***(Refer the screenshots below for better understanding).

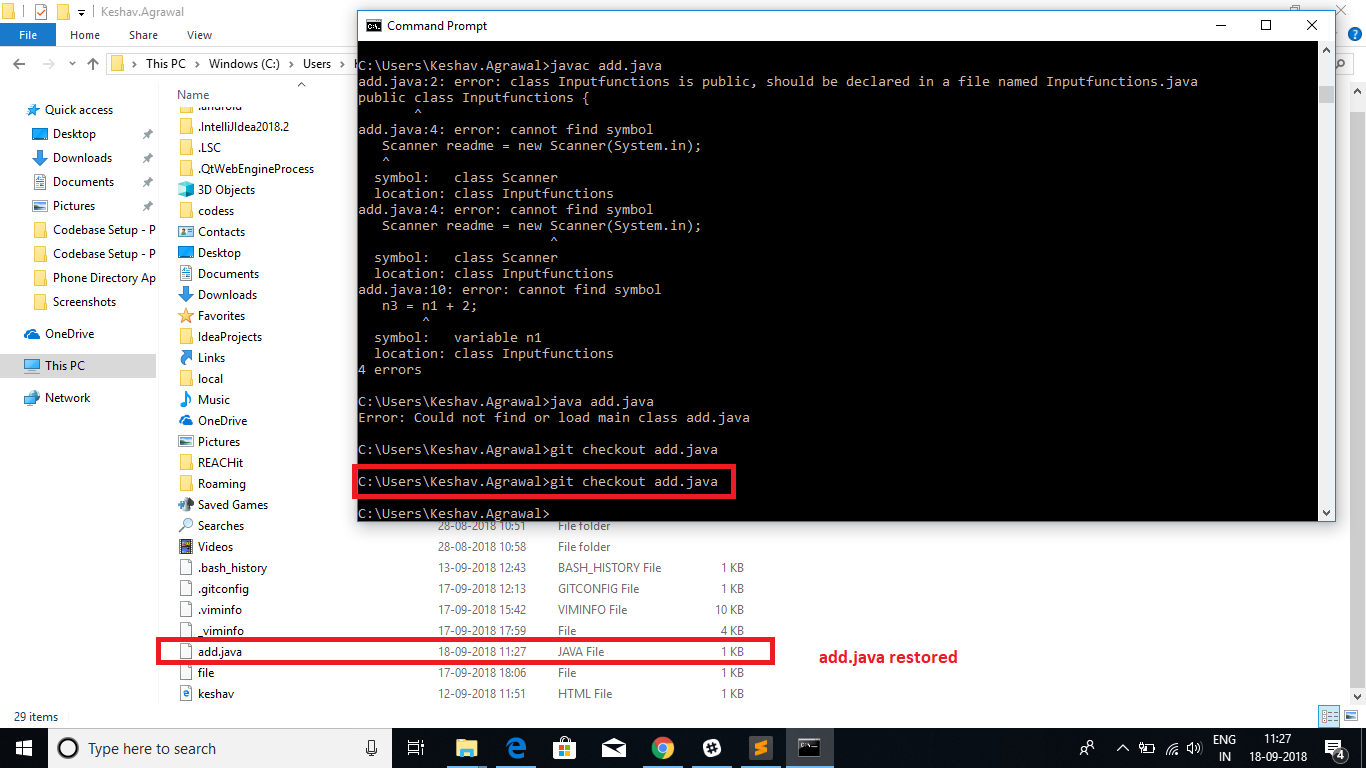
Delete the file ***add.java***



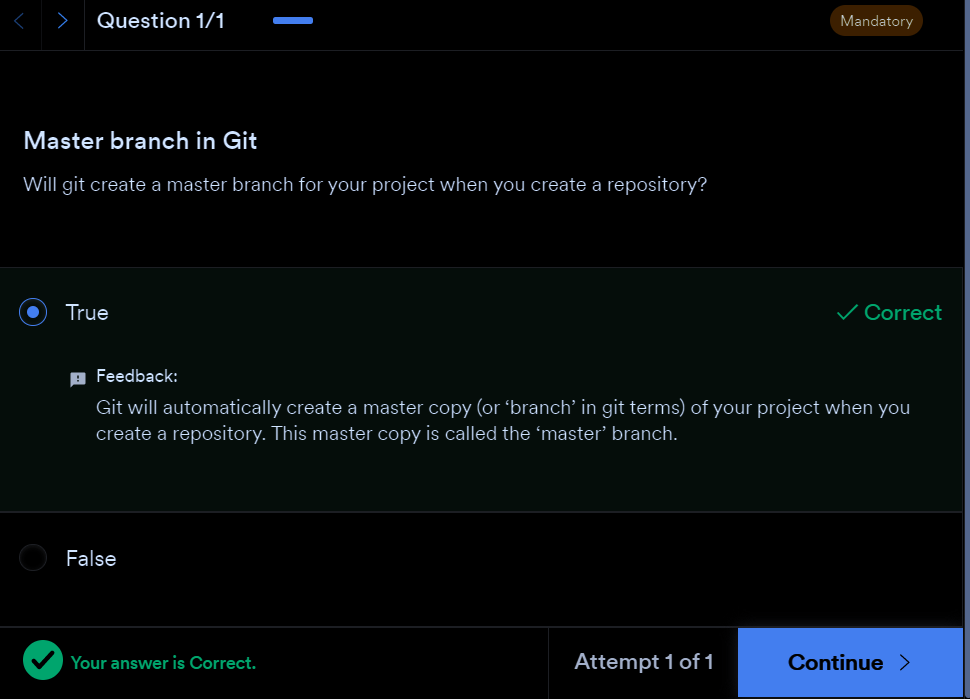
File ***add.java*** deleted

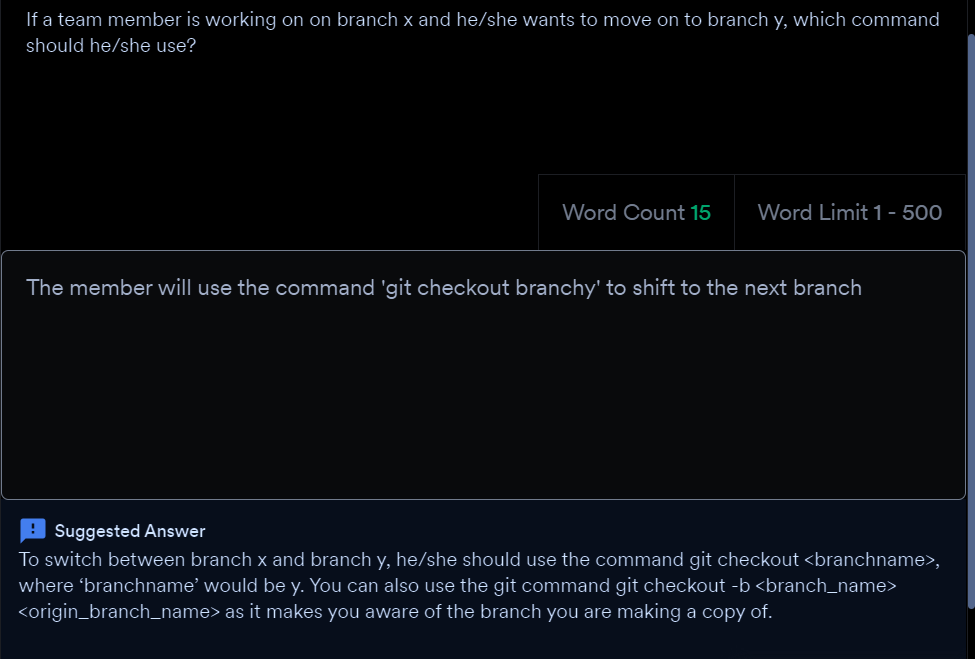


Now run command ***Git checkout add.java***



Our file ***add.java***is restored back in the folder.





* **Merging branches**: For merging, you can use the command **git merge <branchname>**
  + Note: This command will merge the changes in the branch <branchname> with the branch that you are currently working on. Merging can happen between all the branches. Imagine that you have three branches, namely —
    - Master
    - Branch1
    - Branch2

                    You can merge any one of the branches above with another one.

* **Deleting branches**: You can use the command **git branch -d <branchname>**

Let's learn more about the **git diff** command.

* The **git diff**command: This command is used to show the changes performed between commits. The main objective of version control is to enable you to work with different versions of the same file. Hence, git provides the command 'diff' to allow you to compare between the different versions of your files. The most common scenario where 'diff' is used would be when you need to see what changes you had made after your last commit. Ways in which we can use the 'git diff' command:
  + **git diff commitid1 commitid2**: To see the difference between two commits using their commit IDs
  + **git diff branch\_name1..branch\_name2**: To see the difference between two different branches. Here, 'branch\_name1' represents the branch you are currently working on
  + **git diff**: This will show you all the changes made to all the files and branches, all at once

