EXPERIMENT-6

Aim: Add collaborators on GitHub repo.

Theory:

Collaboration is the way different people can work on the same project together. It is like creating a group inGitHub just like Groups in other social media. The people added to the collaborator's list can be able to push, merge, and do other kinds of similar things on the project.

Whenever you make a repository in GitHub, not everyone has the permission to change or push codes into your repository. The users have a read-only access. In order to allow other individuals to make changes to your repository, you need to invite them to collaborate to the project.

GitHub also restricts the number of collaborators we can invite within a period of 24-hours or we can also create an organization to collaborate with more people. Being a collaborator, the user can create, merge and close pull requests in the repository. They can also remove them as the collaborator.

Procedure:

- Login to your GitHub account and you will land on the homepage as shown below. Click onrepositories option in the menu bar.

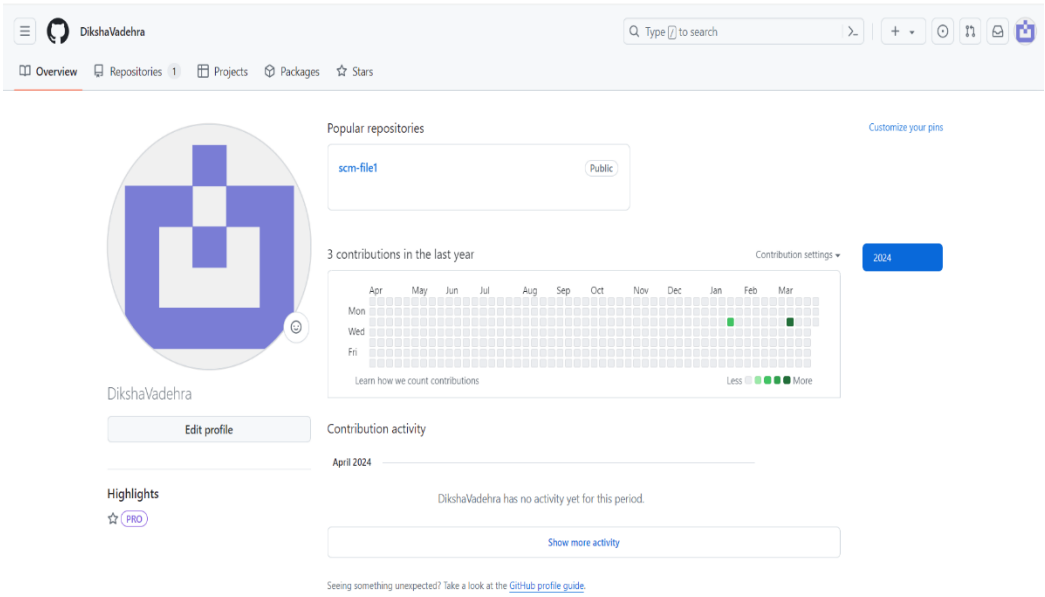


Fig -6.1 creating repository

- Click on the ‘New’ button in the top right corner.
- Enter the repository name and add the description of the repository.
- Select the option whether you want your repository to be public or private.
- Add a Readme file on the basis of your choice.

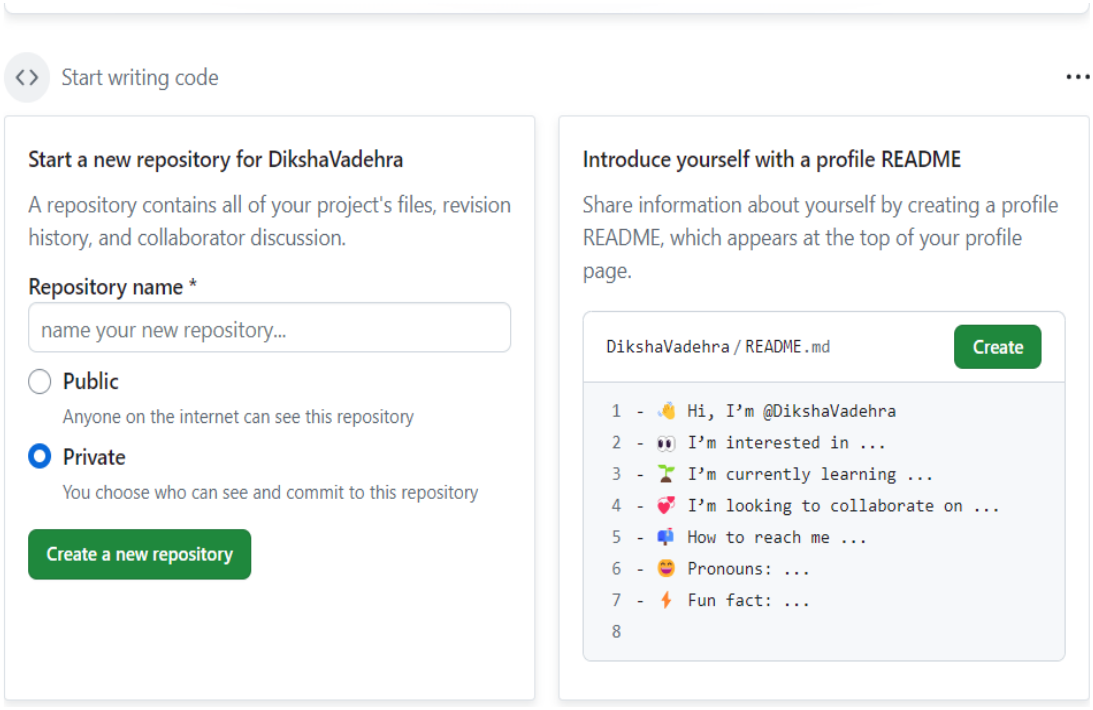
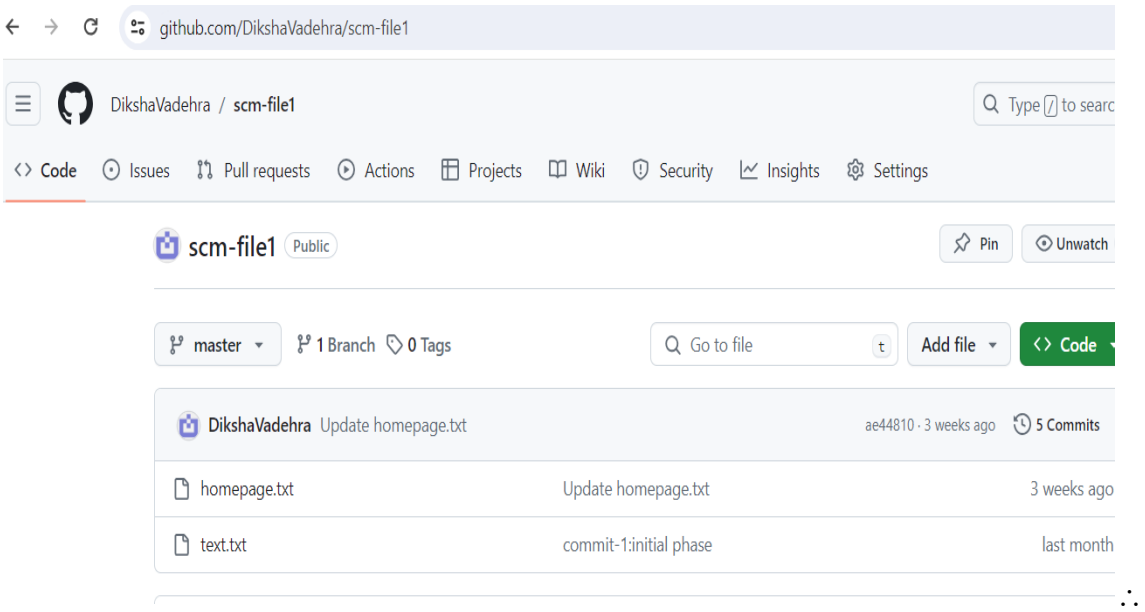


Fig -6.2 new repository

- Now you have created your repository successfully.



∴

Fig -6.3

- Add the files to your project either directly through 'Add file' or through Git Bash using the following commands:

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hello.txt

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:   hello.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        CodeBlocks.lnk
        file.txt
        file2.txt


Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ touch a.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ touch b.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git init
Reinitialized existing Git repository in C:/Users/Diksha Vadehra/Desktop/SCM/.git/

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ add a.txt
bash: add: command not found

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git add a.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git add b.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git commit -m "hello"
[master 1547a2d] hello
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
 create mode 100644 b.txt
 create mode 100644 hello.txt
```

Fig-6.4 -check ‘git status’

To add collaborator to your repository, open your repository and select settings option in the navigation bar.

- Click on the ‘Collaborators’ option under the access tab.
-

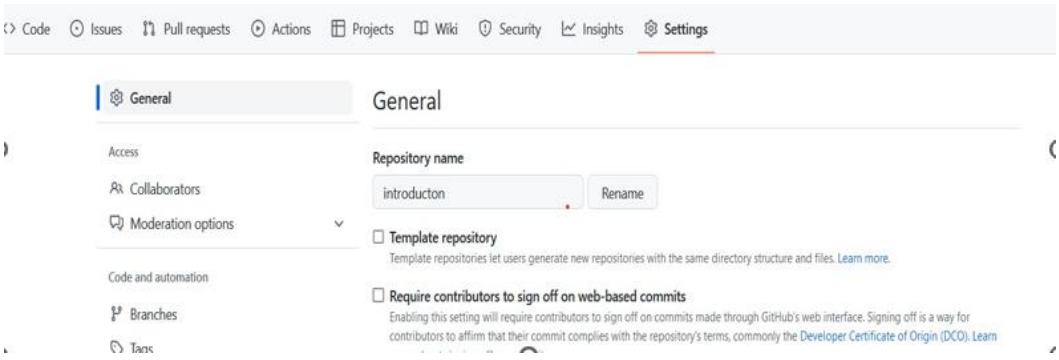
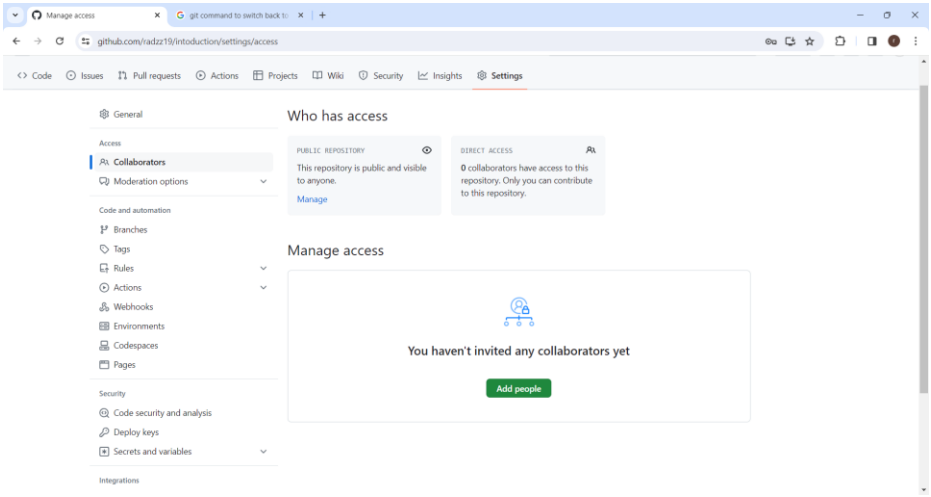


Fig – 6.5 ‘repository name’

After clicking on collaborators, GitHub asks you to enter your password to confirm the

access to the repository.

- After entering the password, you can manage access and add or remove team members to your project.
- To add members, click on the add people option and search the id of your respective team member.



- To remove any member, click on the remove option available in the last column of member'srespective row.

Result: We have successfully collaborated members on our GitHub repo.

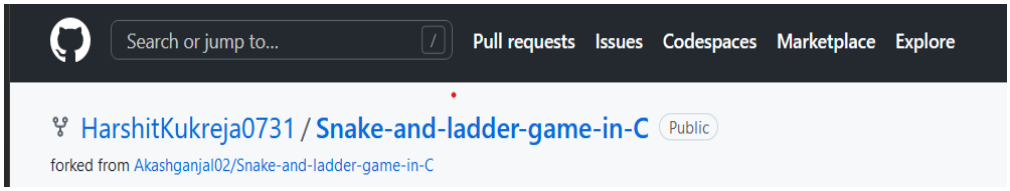
EXPERIMENT-7

Aim: To perform Fork and commit in Git.

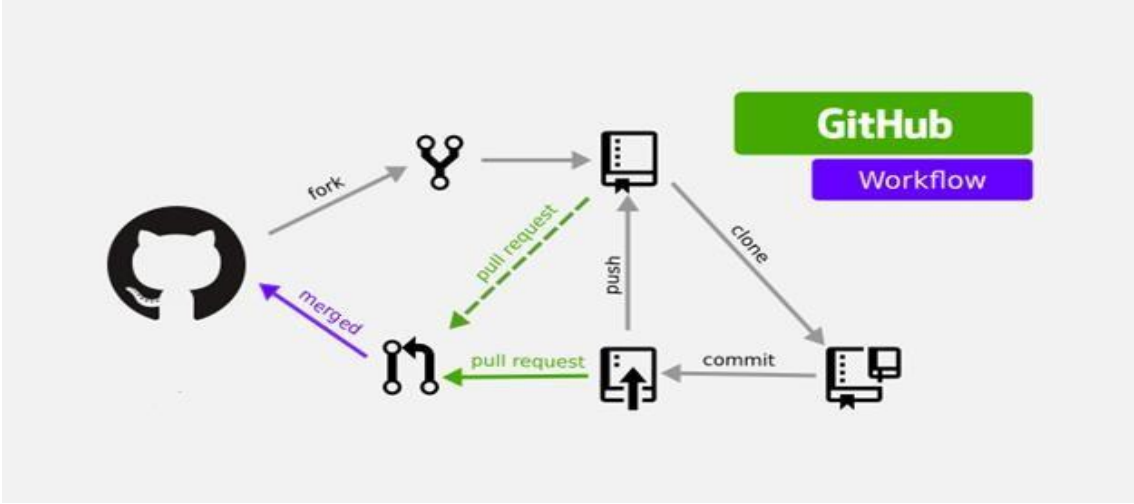
Theory:

A fork is a new repository that shares code and visibility settings with the original “upstream” repository. Forks let you make changes to a project without affecting the original repository, also known as the "upstream" repository. After you fork a repository, you can fetch updates from the upstream repository to keep your fork up to date, and you can propose changes from your fork to the upstream repository with pull requests. A fork can be owned by either a personal account or an organization.

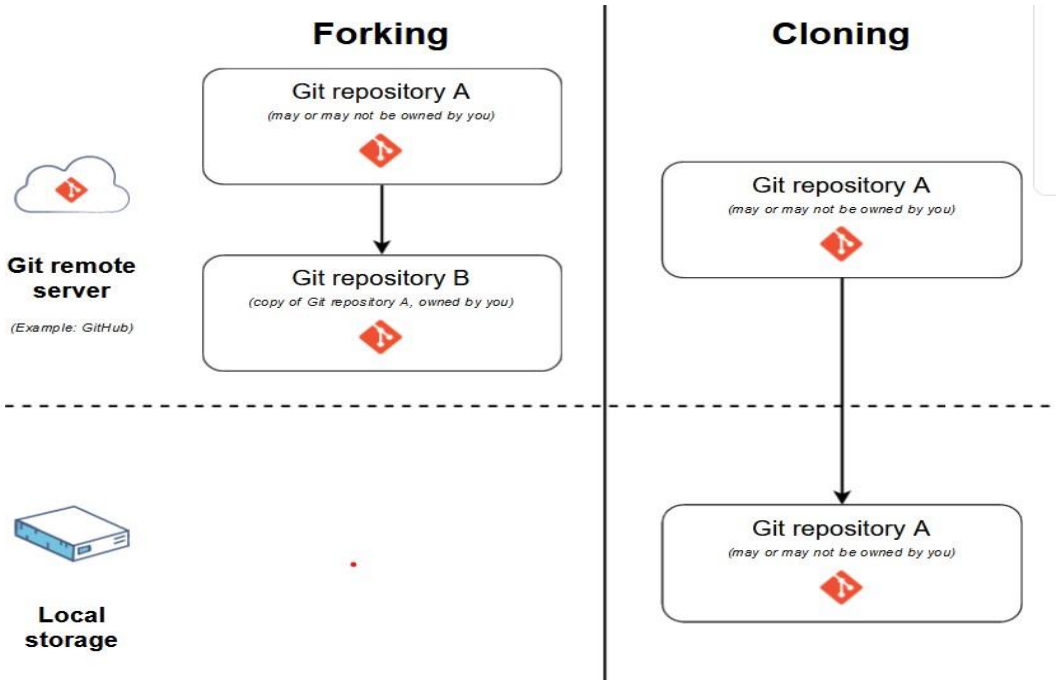
When you view a forked repository on GitHub, the upstream repository is indicated below the name of the fork.



In opensource projects, forks are often used to iterate on ideas or changes before incorporating the changes into the upstream repository. If you fork a public repository to your personal account, make changes, then open a pull request to propose your changes to the upstream repository, you can give anyone with push access to the upstream repository permission to push changes to your pull request branch (including deletingthe branch). This speeds up collaboration by allowing repository maintainers to make commits or run tests locally to your pull request branch from a user-owned fork before merging.



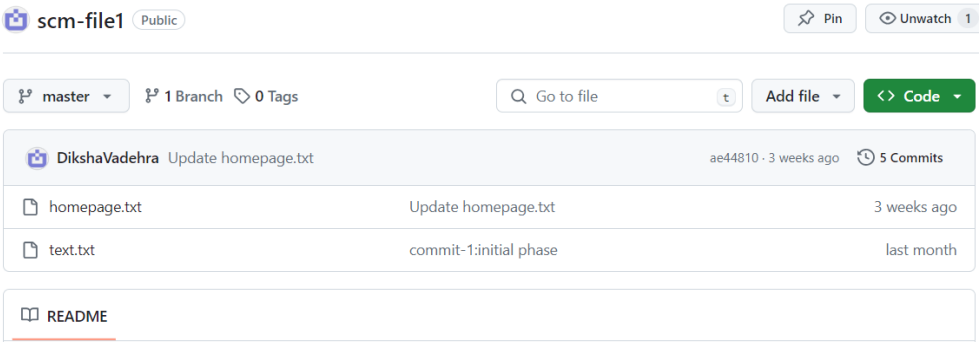
Forking creates your own copy of a repository in a remote location (for example, GitHub). Your own copy means that you will be able to contribute changes to your copy of the repository without affecting the original repository. Cloning makes a local copy of a repository, not your own copy. Think of it as downloading a repository onto your local hard drive. Unlike forks, clones have references to their original repositories.



Forking is a better method than cloning any repository, as in cloning only the default branch is cloned whereas forking creates the clone of the whole repo and also allow us to push changes to the main repositoryby using open and close pull requests.

Procedure:

- To fork a repository the first thing you need to do is, sign in to your GitHub account and then you come to the repository you want to fork, so here for demo purpose am using Akashganjal02/ Snake- and-ladder-game-in-C.



- Click on the fork button on the right upside corner. Then it will ask to create a

new fork, add description if you want and then click on create fork .

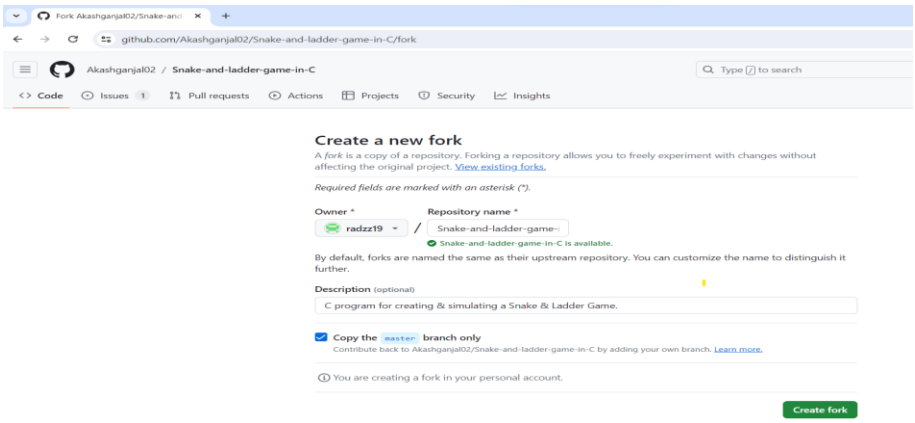


Fig-7.2

- Now you will have a copy of a repo you have forked from another user. Now you Ican do any modification you want without making changes to main source code

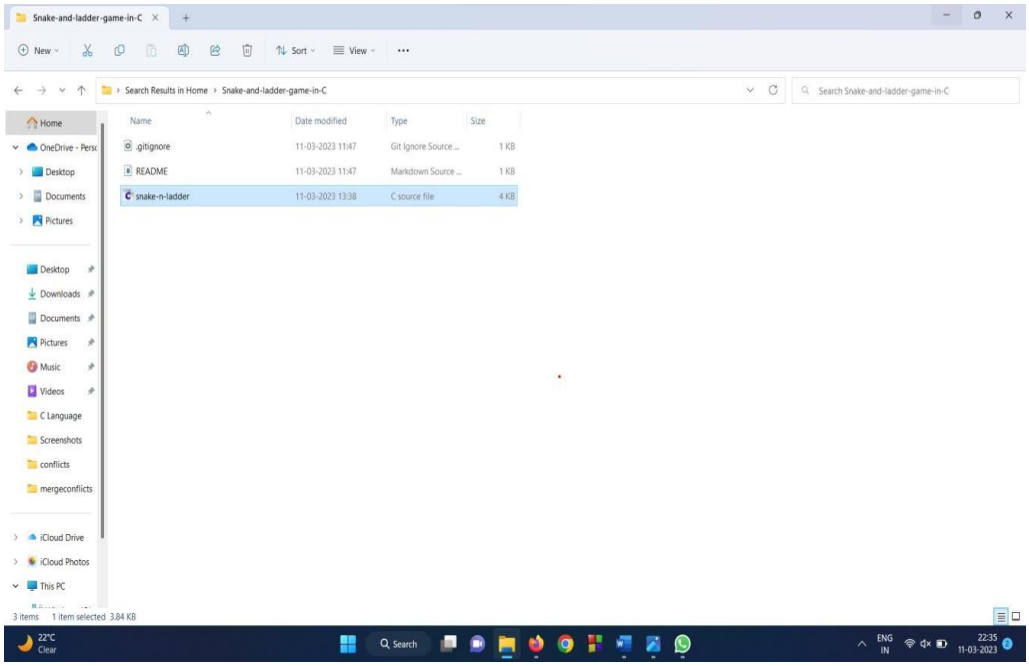


Fig -7.3

Clone the forked repository:

1. Copy the URL from GitHub.

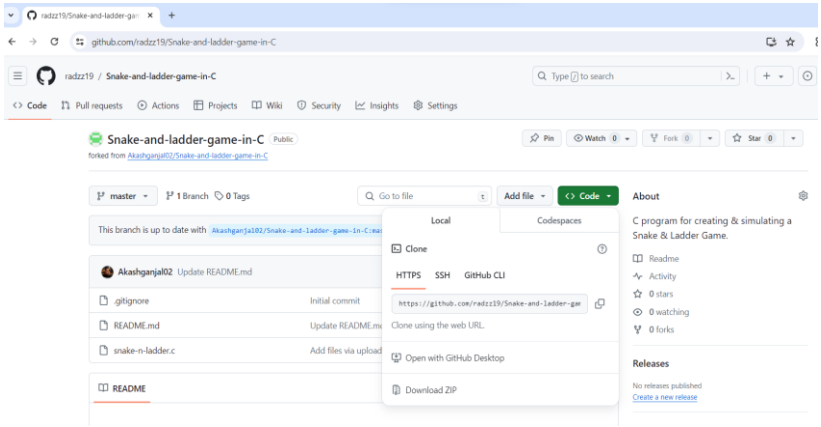


Fig-7.4

2. And then paste it with the git clone command on your computer. This will create a directory and allow you to access the files on your local machine.

"git clone <github_HTTPS_link>"

3. On your local machine enter the command to move into the repository using the command "cd <repository name>"

4. Use vi to open the file by entering the command "vi <file name>".

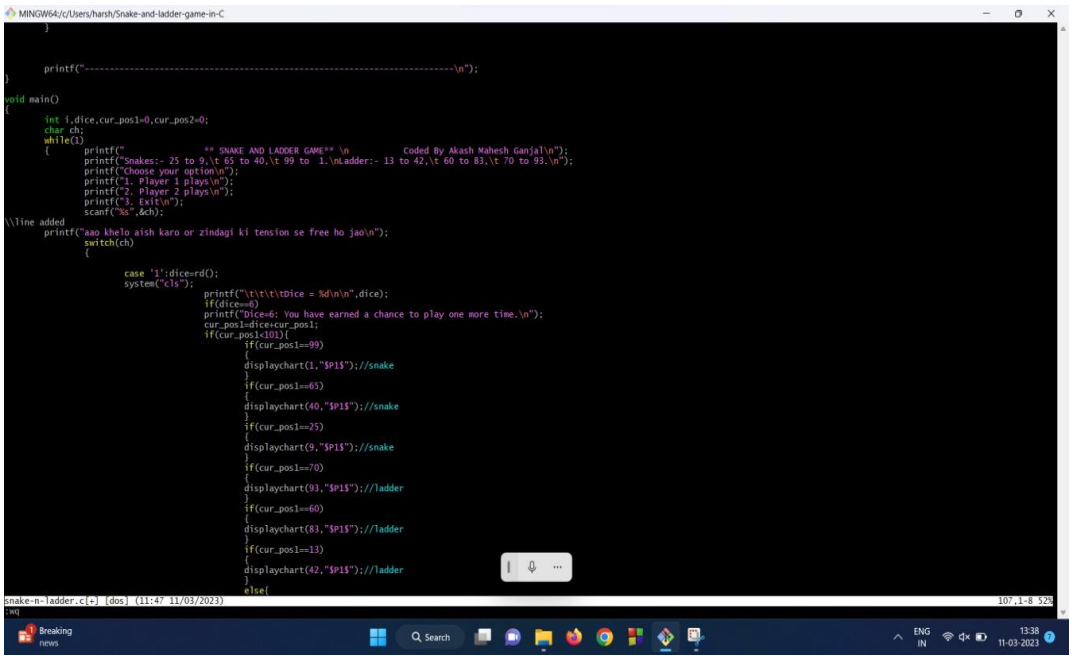
Once the file is open, make your personal changes then hit **ESC** and type ":wq" to save and exit.

vi stands for "visual interface". It is a standard command-line text editor.

It involves different modes.To run a ci we can call it from the command line, appending the file that we want to edit:

vi is a modal text editor: It has multiple modes. When a file is opened, we're not insert mode but command mode. This is why we cannot yet write on the file!

To enter the Insert mode, we have to press: i.



After performing the modifications, we can go back to command mode with the esc key. To save and quit, we use: ":wq" or to save and exit use ":x".

After performing the modifications, we can go back to command mode with the esc key. To save andquit, we use: ":wq" or to save and exit use ":x".

- The next step is to add these changes to the staging area and commit them using the commands "gitadd" and "git commit -m <message>".

The "git status" command is then executed in order to see the reflected changes.

On your local machine type this command to get the remote name: "git remote".

Once you have the remote name enter this command "git push <remote name>"

to push the file back to GitHub.

Once complete you will see that the changes have been made and are now shown on the GitHub repository.

- Submit a pull request: Here on the GitHub dashboard click the pull requests tab and then click NewPull Request. To finish just add your comments on your changes then click "Create pull request.

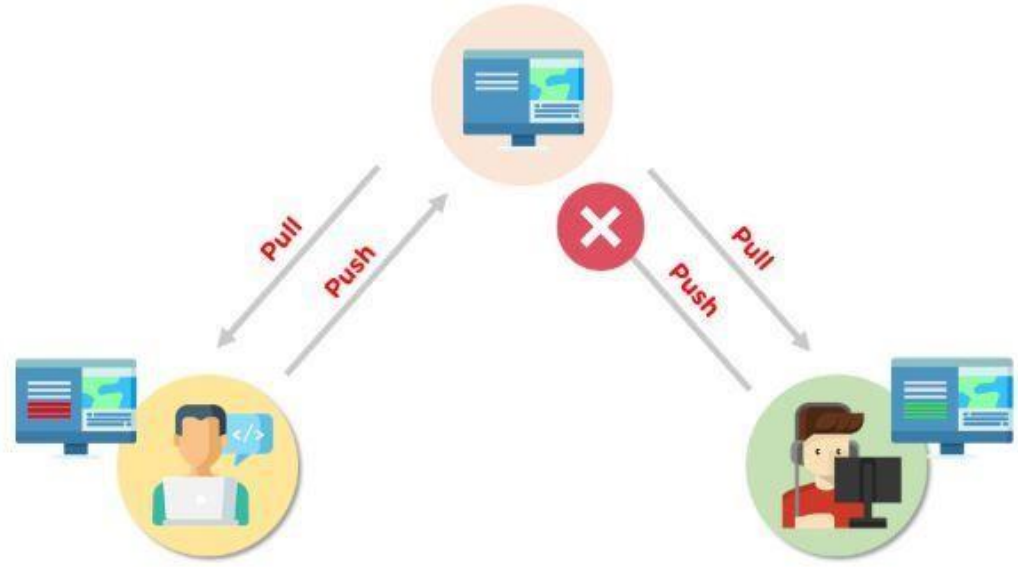
Result: We have successfully forked a repository and committed as well.

EXPERIMENT-8

Aim: Merge and resolve conflicts created due to own activity and collaborators' activity.

Theory:

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in codebetween two commits. Git can merge the changes automatically only if the commits are on different lines orbranches. The following is an example of how a Git merge conflict works:



Let's assume there are two developers: Developer A and Developer B. Both of them pull the same code filefrom the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.

If you have a merge conflict on the command line on the command line, you cannot push your local changesto GitHub until you resolve the merge conflict locally on your computer.

To alleviate occurrence of conflicts, developers will work in separate isolated branches. If a merge conflict still arises between the compare branch and base branch in your pull request, you can view a list of the files with conflicting changes above the merge pull request button. The merge pull request button is deactivated until you have resolved all conflicts between the compare branch and base Branch.

Procedure:

- Initialize two repositories using the command "git init". Both of these repositories represent twodifferent repositories of two different developers.
- Add the remote address in the c repository using command "git remote add origin <address>".

- The next step is to pull all the changes from the central repository to the local repository a using the command "git pull origin master".

- Now move to the d repository using command "cd\" and "cd d". Follow the same process to add the origin in the b repository. The pull command is executed again to retrieve all the content from the remote repository and move it to the local repository b.

- Now again move back to the a repository using command "cd\" and "cd a

- In the c repository, any uploaded file is opened in order to make various changes using vi editor.

  vi stands for "visual interface". It is a standard command-line text editor. It involves different modes.To run a ci we can call it from the command line, appending the file that we want to edit

  vi is a modal text editor: It has multiple modes. When a file is opened, we're not insert mode but command mode. This is why we cannot yet write on the file! To enter the Insert mode, we have to press: i.

  After performing the modifications, we can go back to command mode with the esc

key. To save andquit, we use: ":wq" or to save and exit use ":x".

- The "git status" command is then executed in order to see the reflected changes.
- The next step is to add these changes to the staging area and commit them using the commands "gitadd" and "git commit -m <message>".
- After the commit is finished, the changed file is pushed to the remote repository using the command"git push origin master".
- Now, return to the d repository and open the same file in order to make changes through it. Itrepresents change from any other collaborator.
- The "git status" command is then executed in order to see the reflected changes.
- Make changes to the file, save it, and close it. After that, add the changed file and commit it usingthe commands "git add" and "git commit -m <message>".
- The next step is to push the file to the remote repository using the command "git push origin master

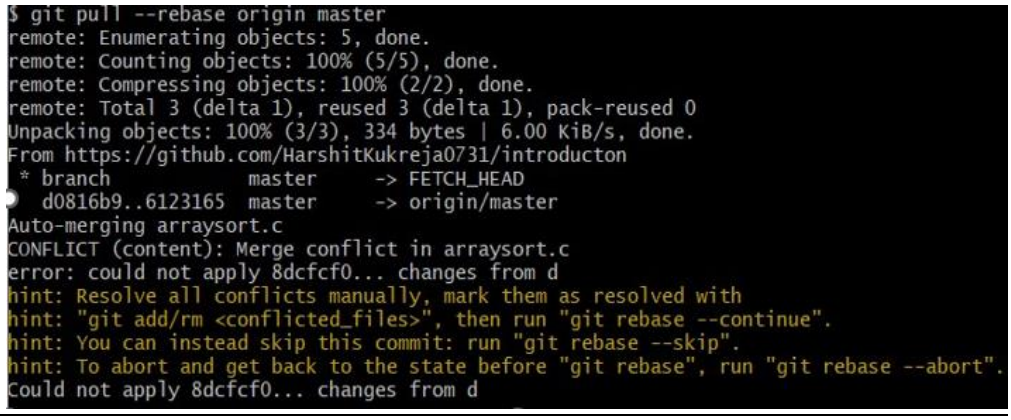


Fig -8.1

An error is shown, meaning that the updates are rejected.

- Next, we need to execute "git pull –rebase origin master". With Git pull rebase, the unpublished changes will be reapplied on top of the published changes and no new commit will be added to your history. With this in mind, you can see that Git pull rebase will result in a linear and cleaner project history by removing the unneeded merge commit.

1. Currently, there are visible conflicts that need to be resolved manually. If you want to skip this commit, you can type git rebase --skip, or if you want to abort this rebase, you can type git rebase --abort.

- Now use the command "git mergetool" to solve the conflict. It will run merge conflict resolution tools to resolve merge conflicts.

  After we input this command, all of the files will be processed.

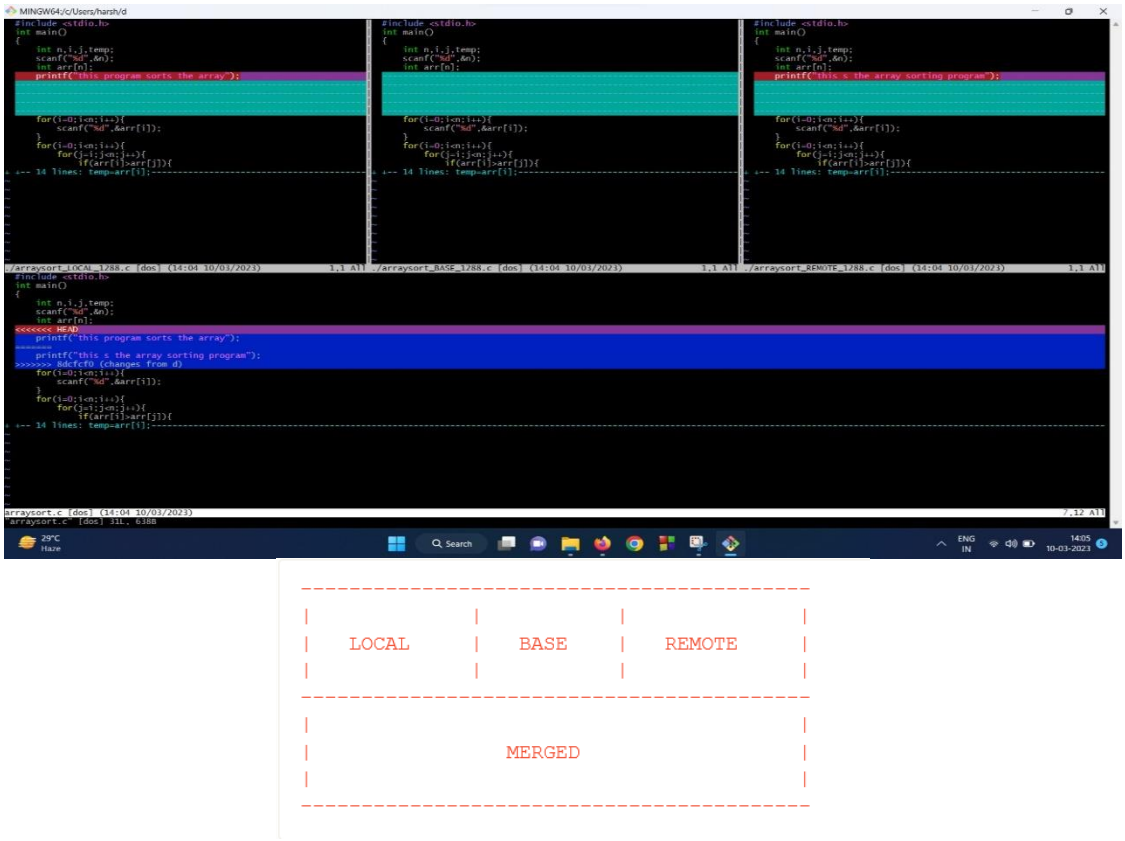


Fig -8.2

LOCAL, BASE and REMOTE are read-only buffers showing the contents of the conflicting file in specific commits. MERGED is a writable buffer where you have to resolve the conflicts (using the other read-only buffers as a reference). Once you are done, save and exit Vim as usual (:wq) (:x)or, ifyou want to abort, exit using (:cq).

- Now we will run the command "git rebase –continue". It will continue with the changes that you made.

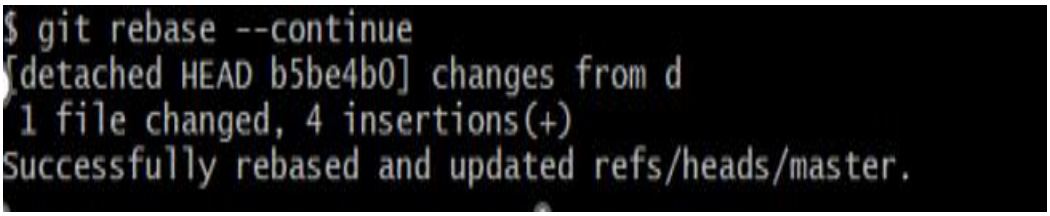


Fig-8.3

- Now, when the conflict is resolved, we must be able to push the file to the remote repository usingthe command “git push origin master”

Result: We have successfully merged and resolved conflicts created due to own activity and collaborators’activity.

Experiment – 9

Aim: Reset and Revert.

Theory:

The git reset command is used to undo the changes in your working directory and get back to a specific commit while discarding all the commits made after that one. For instance, imagine you made ten commits. Using git reset on the first commit will remove all nine commits, taking you back to the first commit stage.

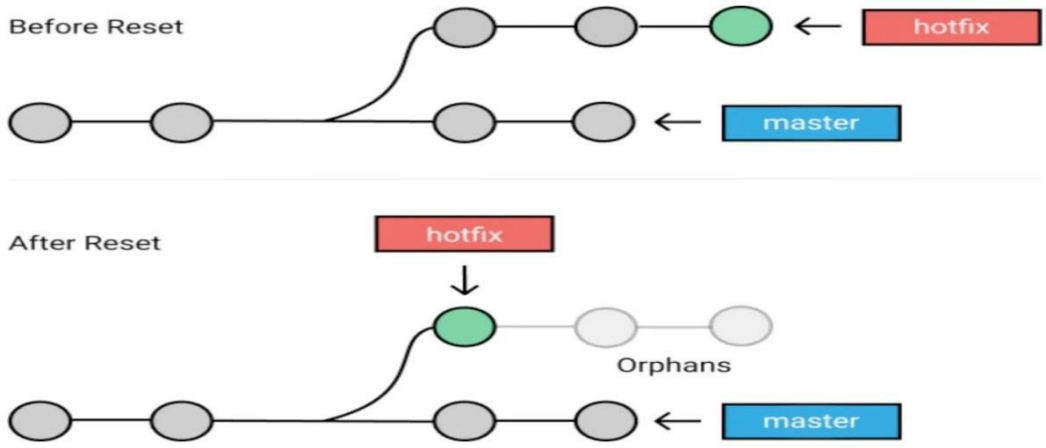


Fig -9.1

Before using git reset, it is important to consider the type of changes you plan to make; otherwise, you will create more chaos than good. You can use multiple options along with git reset, but these are the main ones. Each one is used depending on a specific situation: git reset --soft, git reset --mixed, and git reset –hard Git revert is similar to git reset, but the approach is slightly different. Instead of removing all the commits in its way, the revert only undoes a single commit by taking you back to the staged files before the commit.

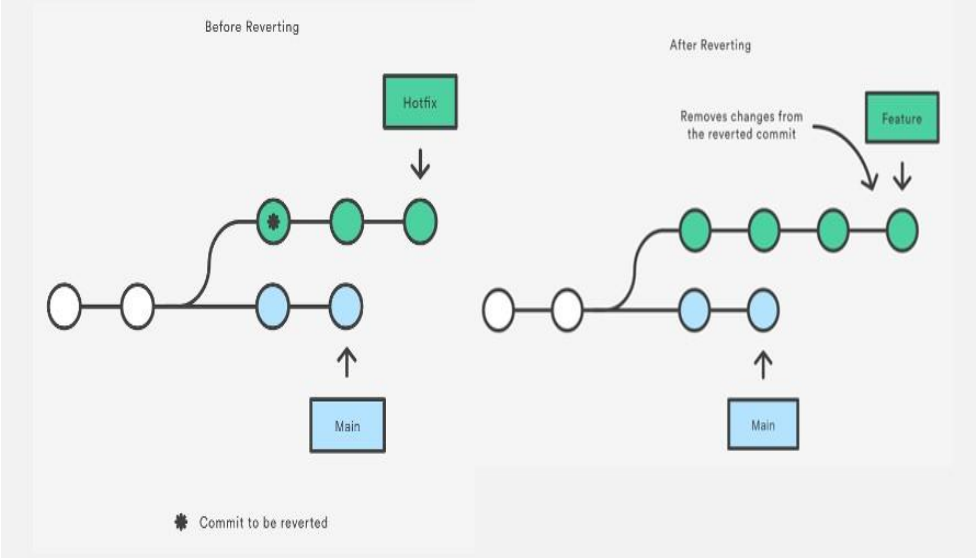


Fig 9.2

So, instead of removing a commit, git revert inverts the changes introduced by the original commit by creating a new commit with the underlying inverse content. This is a safe way to revoke a commit because itprevents you from losing your history. It is important to know that the revert will have no effect unless the commit hash or reference is specified.

Procedure:

Reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ touch ab.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ vi ab.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git add ab.txt
warning: in the working copy of 'ab.txt', LF will be replaced by CRLF the next time Git touches it

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ab.txt

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:   hello.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1
        CodeBlocks.lnk
        d.txt.i.txt
        file.txt
        file2.txt
        k.txt
```

Fig -9.1

- Create few files, stage them and commit.

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git commit -m "commit:2"
[master 00243bf] commit:2
 1 file changed, 2 insertions(+)

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

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:   hello.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1
        CodeBlocks.lnk
        d.txt.i.txt
        file.txt
        file2.txt
        k.txt

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

Fig -9.2

- Check the git log.

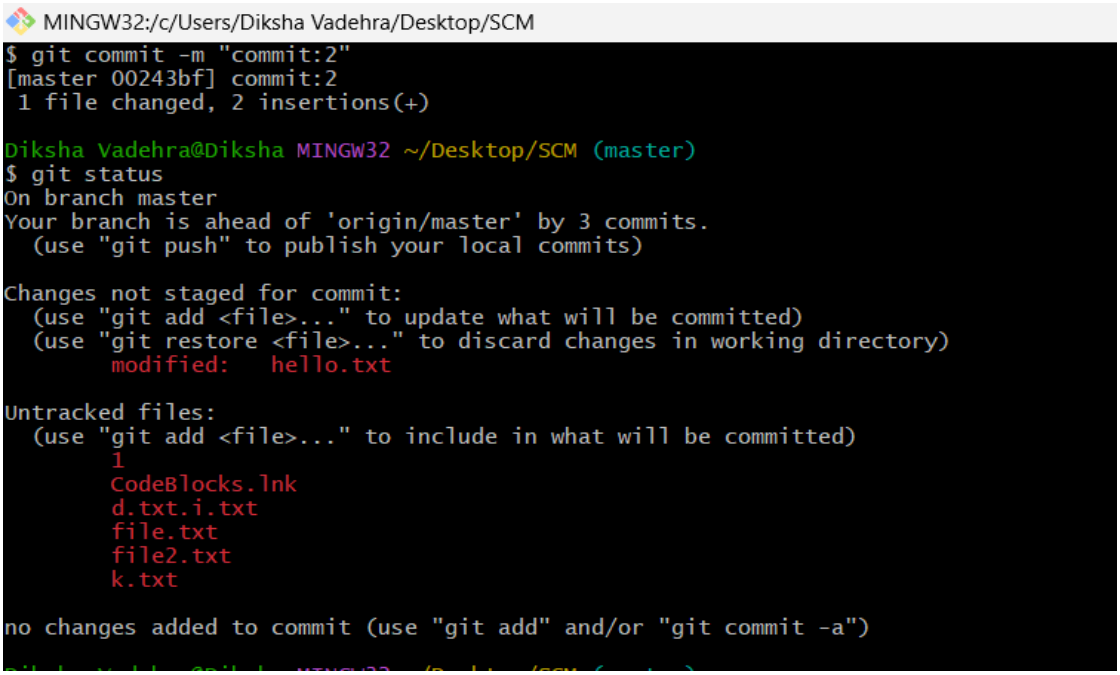


Fig -9.3

- Pick any commit where you want the repository to rollback. Copy its checksum and paste it in the"$ git reset checksum" command.

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git reset
Unstaged changes after reset:
M       hello.txt

Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$
```

git reset –soft:

Git reset is --soft for the Git reset soft command. This option moves HEAD back to the specified commit, undoes all the changes made between where HEAD was pointing and the specified commit, and saves all the changes in the index. In other words, Git re-adds the changes as staged, ready to be committed again.

git reset –mixed:

performing a Git reset with the --mixed option will undo all the changes between HEAD and the specified commit, but will preserve your changes in the Working Directory, as unstaged changes. If you perform a Git reset and do not supply an option of --soft, --mixed, or -- hard, Git will use --mixed by default.

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git reset --mixed HEAD
```

Fig -9.4

- **git reset --hard:**

  This option has the potential of being dangerous. So, be cautious when using it!

```
Diksha Vadehra@Diksha MINGW32 ~/Desktop/SCM (master)
$ git reset --hard
HEAD is now at 00243bf commit:2
```

Fig -9.5

Basically, when using the hard reset on a specific commit, it forces the HEAD to get back to thatcommit and deletes everything else after that.

Before running the hard reset, let's have a look at the content of the folder.

Revert: Follow these steps to revert any change:

- make a new repo, make files in it , add them in staged area and commit them differently.

• Pick the change where you want the project to revert back. Copy its checksum and paste it in therevert command.

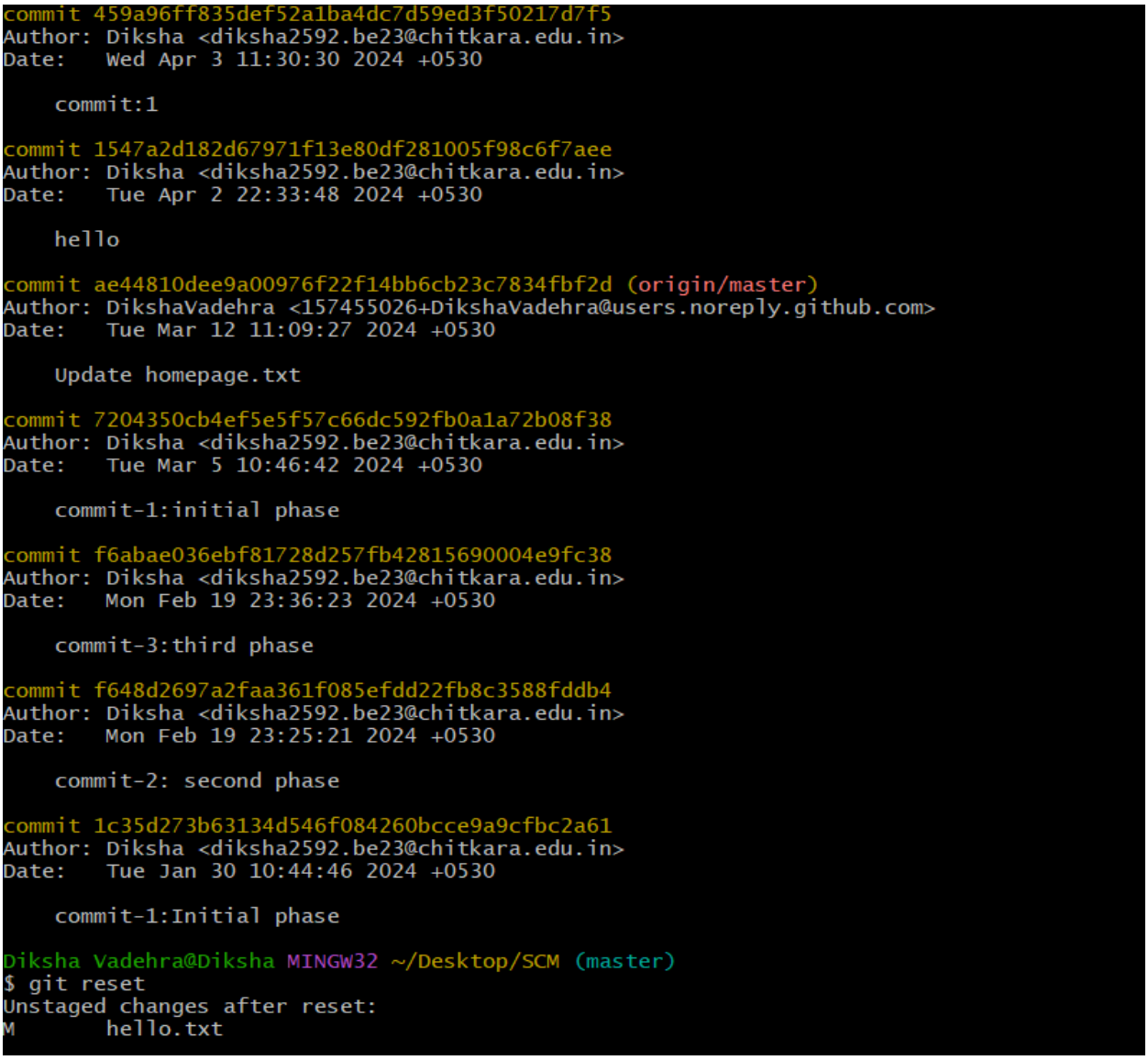


Fig -9.6

- A window will appear. Press ‘I” and write the statement you want to be displayed for reverting thechange.
- After completing press ‘esc’ and write: wq in the terminal.
- Check the git log and you will find another commit is added without affecting the rest commits.
- The change associated to the reverted commit has disappeared.

Result: We have successfully done reset and revert in git.