**Git Core Concepts:**

There is a difference in Git and Github

Git is the version control tool while Github is a hosting platform which hosts git repositories.

We can push the local git repositories to the remote web hosting platform - Github using different commands

**Repositories** in git are like the location of certain data/app related files

And the **branches** are like the folders in that location

**Commit** is different stages - changes made in that folder

**Push** is to push the changes to the remote platform ie Github from your local machine

**Pull** is the command used to pull any changes or the files from the remote branch to the local app folder

**git init** - will tell git to manage the project in this folder and so git will initialise an empty git repository and create a .git file

**git status** - will give you the current status, eg: if there are untracked files which are not yet committed to the remote repository

So, we should explicitly let git know which files to include on the Github repository

To track these untracked files the following command:

For all files : **git add .**

For specific file: **git add filename**

This will add the project files to the git's tracking logic which will then help git to track all these files

This will stage the files.

To unstage the files the command used is : **git rm --cached filename**

After adding the files to be staged, we commit these files

**git commit -m "write some msg related the commit"**

-m is a short form for message

commit will help the git to know that the files are to be added in the next version

**git branch**

Lists all branches in the repository

Git automatically creates the master branch

Any changes made are added to the current version of the branch whenever you commit the changes

To list all the commits done for the current version we use the command - **git log**

Every commit had a unique id. So to go back to the previous commit we use:

**git checkout commit-id**

this will keep the latest commits as well

To jump back to any previous commit and delete the latest commits:

**git reset --hard commit-id**

For example: if you want to just go back to a previous commit without deleting any other commits, use   
**git checkout commit-id**

But, if you want to go back to a previous commit and want to permanently delete the latest changes and commits we use: git checkout commit-id

here, commit-id is the id of the commit you want to keep and work on.

To revert all unstage changes (not use the git add . command) and go back to the last commit:

**git checkout -- .**