**git clone <url of the repository>**

To clone the repository locally.

To commit the change:

**git commit -m ‘message to let others know the changes you have made’**

to add file in the stage area so that it can be committed:

**git add <file name>**

To know if the file is committed:

**git log**

To quickly add and commit the changes:

**git commit -am ‘message’**

**(**It adds and commits every file in that directory. So, care should be taken to use this command**)**

Undo uncommitted changes: (to discard changes in your working directory)

**git checkout -- <file name>**

to undo multiple uncommitted files:

**git checkout -- .**

undo committed changes:

**git revert <commit id>**

**(**maybe you have to type **:q** to close the ui for git revert.**)**

**(**This basically means reverting back to an earlier version of the file after a commit has already been made.**)**

The commit id can be obtained by using the **git log** command and then it can be copied from there and pasted to the desired location.

git revert directly commits the change but if you do not want to commit the change right away but run git revert and then explicitly commit the change then we have to use **-n**  with git revert.

After running git revert -n the change occurs but commit is not done. We have to explicitly commit the change.

To go to a point after which every data is not useful and we want to reset to that point.

**git reset -hard <commit id>**

This command can be very powerful so it should be used with proper care, as once it is done, the data after the set point is lost.

**git branch**

lists all the branches that you have.

To make a branch

**git branch <branch\_name>**

to make the new branch active, we use the checkout command.

**git checkout <branch\_name>**

To merge two branches together.

First, we need to go to the base branch.

**git checkout <base\_branch\_name>**

then we type the command

**git merge <branch\_name\_that\_we\_want\_to\_merge>**

To make a branch, we used two commands, **git branch** and **git chechout**. We can do both the things in one command by using:

**git chechout -b <branch\_name>**

Deleting a branch

**git branch -d <branch\_name>**

**Head -**  Head is a reference to the most recent commit in current branch (in most of the cases)

**git show HEAD**

and

**git show <commit\_id>**

both produce the same output.

**git difftool <older\_commit\_id> <newer\_commit\_id>**

used to show the difference between two commits.

The same can be achieved by using the HEAD command.

HEAD is the current commit

HEAD~1 is the commit below HEAD

HEAD~2 is the commit below HEAD~1

**git difftool HEAD~2 HEAD~1**

When HEAD does not point to the most recent commit , you go into detached HEAD state.

Sometimes, we want to go to a previous commit and based on that previous commit, we want to make some changes.

**git checkout <commit\_id>**

When we do not want to check in a file upstream github repository, we use the gitignore file.

Creating a gitignore file:

**touch .gitignore**

In this file, we put the directory we do not want to be seen using the **git status** command.

(Files won’t show up in a list of untracked files that show up after running the **git status** command.)

We put \*.extension to exclude all the files we don’t want to keep showing up in the untracked files list.

To see the difference between github (upstream) and our local repository

**git difftool origin/master**

origin/master - means github

**git difftool**

It shows the changes between the uncommitted change and the head.

For multiple difftool we can use **-t** along with **git difftool** command to launch the tool that we want to use.

**git difftool -t <tool\_name>**

How to resolve/merge conflicts

We can use mergetool to resolve conflicts. Before, we do that, we need to configure the tool that we want to use for merging.

**git mergetool**