Git Diff

Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.

It compares the different versions of data sources. The version control system stands for working with a modified version of files. So, the diff command is a useful tool for working with Git.

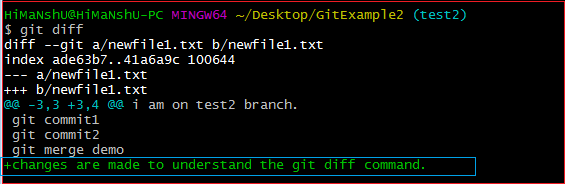
However, we can also track the changes with the help of git log command with option -p. The git log command will also work as a git diff command.

Let's understand different scenarios where we can utilize the git diff command.

**Scenerio1: Track the changes that have not been staged.**

The usual use of git diff command that we can track the changes that have not been staged.

Suppose we have edited the newfile1.txt file. Now, we want to track what changes are not staged yet. Then we can do so from the git diff command. Consider the below output:



From the above output, we can see that the changes made on newfile1.txt are displayed by git diff command. As we have edited it as "changes are made to understand the git diff command." So, the output is displaying the changes with its content. The highlighted section of the above output is the changes in the updated file. Now, we can decide whether we want to stage this file like this or not by previewing the changes.

**Scenerio2: Track the changes that have staged but not committed:**

The git diff command allows us to track the changes that are staged but not committed. We can track the changes in the staging area. To check the already staged changes, use the --staged option along with git diff command.

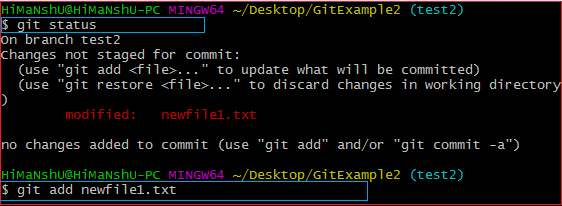
To check the untracked file, run the git status command as:

1. $ git status

The above command will display the untracked file from the repository. Now, we will add it to the staging area. To add the file in the staging area, run the git add command as:

1. $ git add < file name>

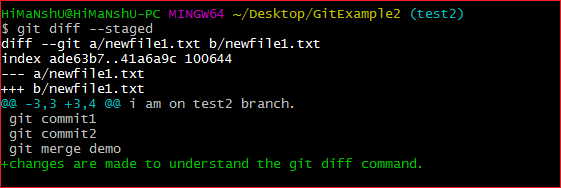
The above command will add the file in the staging area. Consider the below output:



Now, the file is added to the staging area, but it is not committed yet. So, we can track the changes in the staging area also. To check the staged changes, run the git diff command along with **--staged** option. It will be used as:

1. $ git diff --staged

The above command will display the changes of already staged files. Consider the below output:

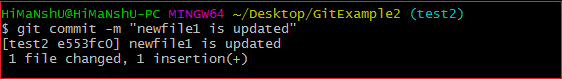


The given output is displaying the changes of newfile1.txt, which is already staged.

**Scenerio3: Track the changes after committing a file:**

Git, let us track the changes after committing a file. Suppose we have committed a file for the repository and made some additional changes after the commit. So we can track the file on this stage also.

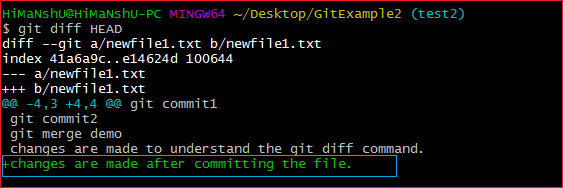
In the below output, we have committed the changes that we made on our newfile1.txt. Consider the below output:



Now, we have changed the newfile.txt file again as "Changes are made after committing the file." To track the changes of this file, run the git diff command with **HEAD** argument. It will run as follows:

1. $ git diff HEAD

The above command will display the changes in the terminal. Consider the below output:



The above command is displaying the updates of the file newfile1.txt on the highlighted section.

**Scenario4: Track the changes between two commits:**

We can track the changes between two different commits. Git allows us to track changes between two commits, whether it is the latest commit or the old commit. But the required thing for this is that we must have a list of commits so that we can compare. The usual command to list the commits in the git log command. To display the recent commits, we can run the command as:

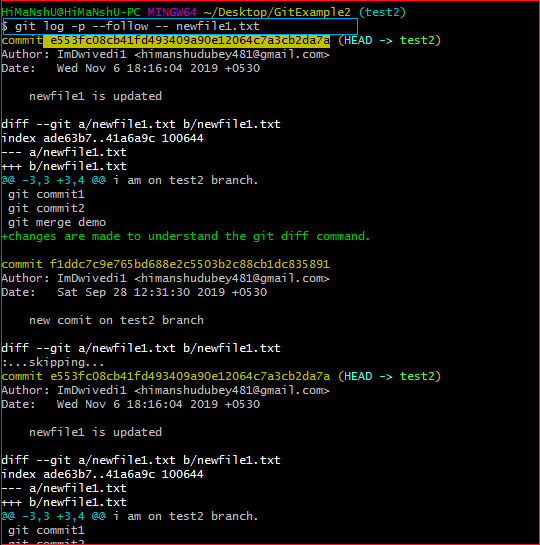
1. $ git log

The above command will list the recent commits.

Suppose, we want to track changes of a specified from an earlier commit. To do so, we must need the commits of that specified file. To display the commits of any specified, run the git log command as:

1. $ git log -p --follow -- filename

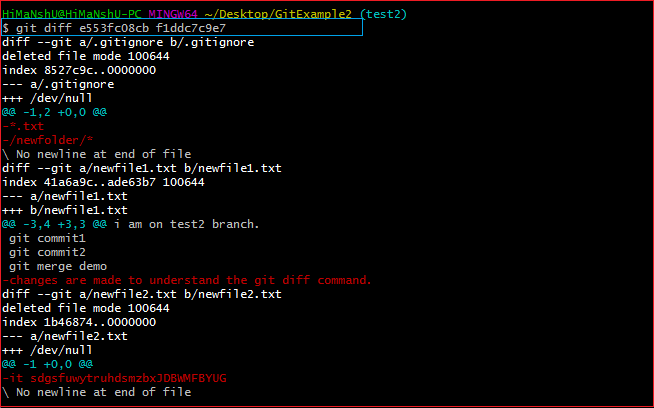
The above command will display all the commits of a specified file. Consider the below output:



The above output is displaying all the commits of newfile1.txt. Suppose we want to track the changes between commits **e553fc08cb** and **f1ddc7c9e7**. The git diff command lets track the changes between two commits. It will be commanded as:

1. $ git diff <commit1-sha> <commit2-sha>

The above command will display the changes between two commits. Consider the below output:



The above output is displaying all the changes made on **newfile1.txt** from commit **e553fc08cb** (most recent) to commit **f1ddc7c9e7** (previous).

Git Diff Branches

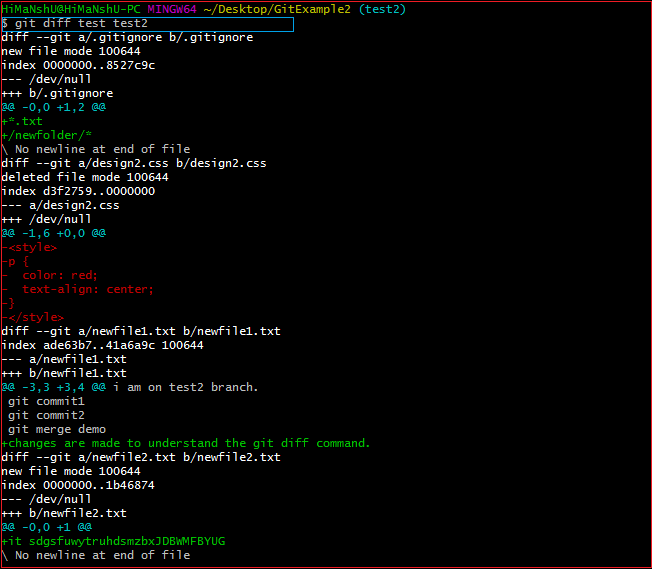
Git allows comparing the branches. If you are a master in branching, then you can understand the importance of analyzing the branches before merging. Many conflicts can arise if you merge the branch without comparing it. So to avoid these conflicts, Git allows many handy commands to preview, compare, and edit the changes.

We can track changes of the branch with the git status command, but few more commands can explain it in detail. The git diff command is a widely used tool to track the changes.

The git diff command allows us to compare different versions of branches and repository. To get the difference between branches, run the git diff command as follows:

1. $ git diff <branch 1> < branch 2>

The above command will display the differences between branch 1 and branch 2. So that you can decide whether you want to merge the branch or not. Consider the below output:



The above output is displaying the differences between my repository branches **test** and **test2**. The git diff command is giving a preview of both branches. So, it will be helpful to perform any operation on branches.