Git log

The advantage of a version control system is that it records changes. These records allow us to retrieve the data like commits, figuring out bugs, updates. But, all of this history will be useless if we cannot navigate it. At this point, we need the git log command.

Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.

Generally, the git log is a record of commits. A git log contains the following data:

- **A commit hash**, which is a 40 character checksum data generated by SHA (Secure Hash Algorithm) algorithm. It is a unique number.
- **Commit Author metadata**: The information of authors such as author name and email.
- **Commit Date metadata**: It's a date timestamp for the time of the commit.
- **Commit title/message**: It is the overview of the commit given in the commit message.

How to Exit the git log Command?

There may be a situation that occurs, you run the git log command, and you stuck there. You want to type or back to bash, but you can't. When you click the **Enter** key, it will navigate you to the older command until the end flag.

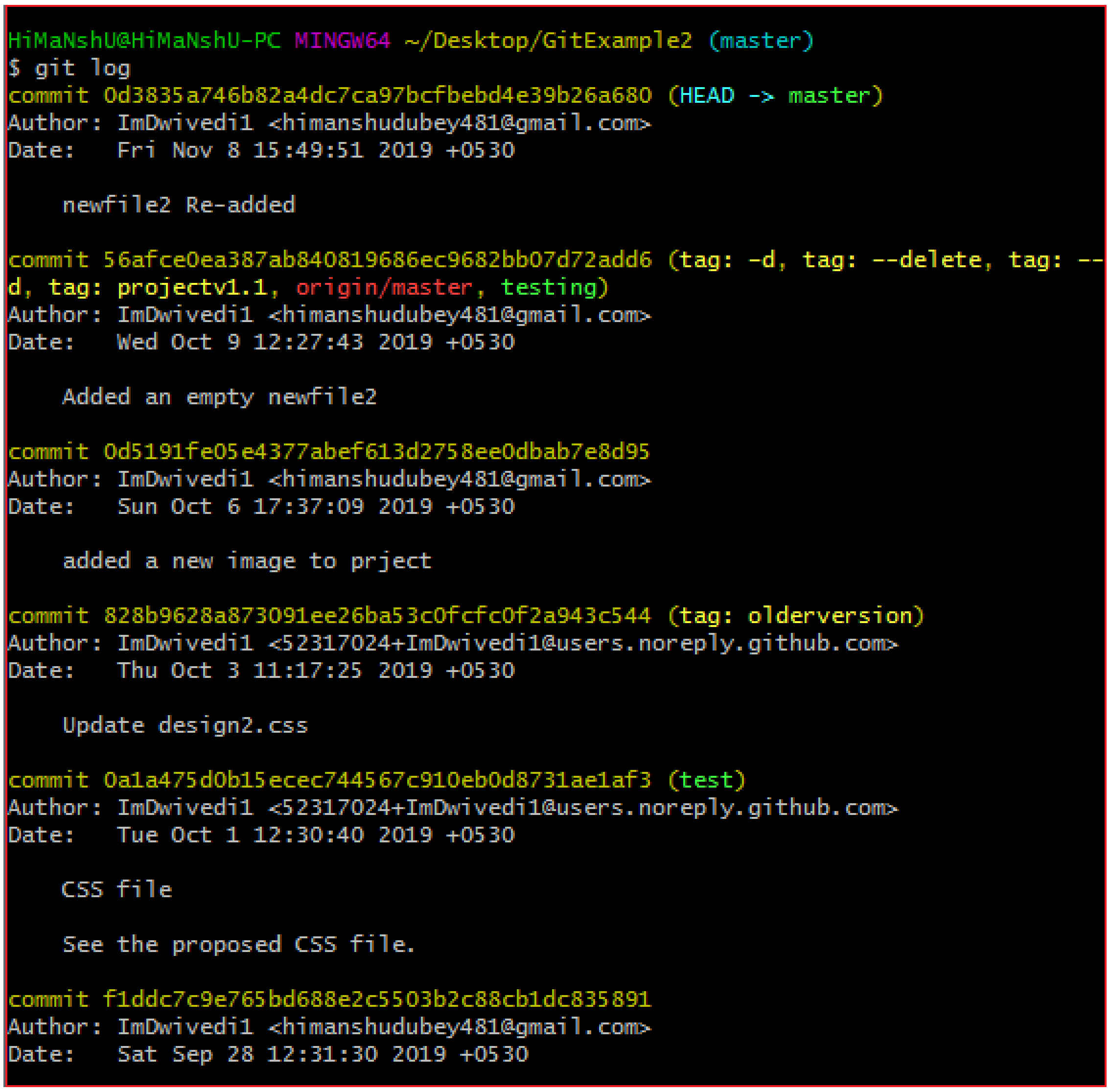
The solution to this problem is to **press** the **q (Q for quit)**. It will quit you from the situation and back you to the command line. Now, you can perform any of the commands.

Basic Git log

Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:

```
1. $ git log
```

The above command will display the last commits. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to prject

commit 828b9628a873091ee26ba53c0fcfc0f2a943c544 (tag: olderversion)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Thu Oct 3 11:17:25 2019 +0530

    Update design2.css

commit 0a1a475d0b15ecec744567c910eb0d8731ae1af3 (test)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Tue Oct 1 12:30:40 2019 +0530

    CSS file

    See the proposed CSS file.

commit f1ddc7c9e765bd688e2c5503b2c88cb1dc835891
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:31:30 2019 +0530
```

The above command is listing all the recent commits. Each commit contains some unique sha-id, which is generated by the SHA algorithm. It also includes the date, time, author, and some additional details.

We can perform some action like scrolling, jumping, move, and quit on the command line. To scroll on the command line press k for moving up, j for moving down, the spacebar for scrolling down by a full page to scroll up by a page and q to quit from the command line.

<

Git Log Oneline

The oneline option is used to display the output as one commit per line. It also shows the output in brief like the first seven characters of the commit SHA and the commit message.

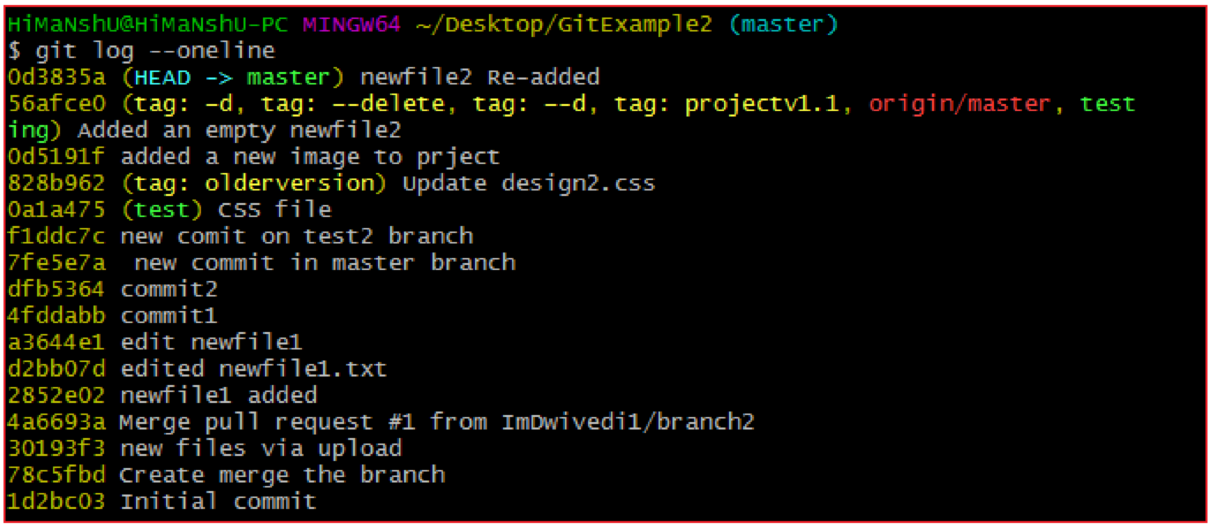
It will be used as follows:

1. $ git log --oneline

So, usually we can say that the --oneline flag causes git log to display:

- one commit per line
- the first seven characters of the SHA
- the commit message

Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --oneline
0d3835a (HEAD -> master) newfile2 Re-added
56afce0 (tag: -d, tag: --delete, tag: --d, tag: projectv1.1, origin/master, test
ing) Added an empty newfile2
0d5191f added a new image to prject
828b962 (tag: olderversion) Update design2.css
0a1a475 (test) CSS file
f1ddc7c new comit on test2 branch
7fe5e7a  new commit in master branch
dfb5364 commit2
4fddabb commit1
a3644e1 edit newfile1
d2bb07d edited newfile1.txt
2852e02 newfile1 added
4a6693a Merge pull request #1 from ImDwivedi1/branch2
30193f3 new files via upload
78c5fbd Create merge the branch
1d2bc03 Initial commit
```

As we can see more precisely from the above output, every commit is given only in one line with a seven-digit sha number and commit message.

Git Log Stat

The log command displays the files that have been modified. It also shows the number of lines and a summary line of the total records that have been updated.

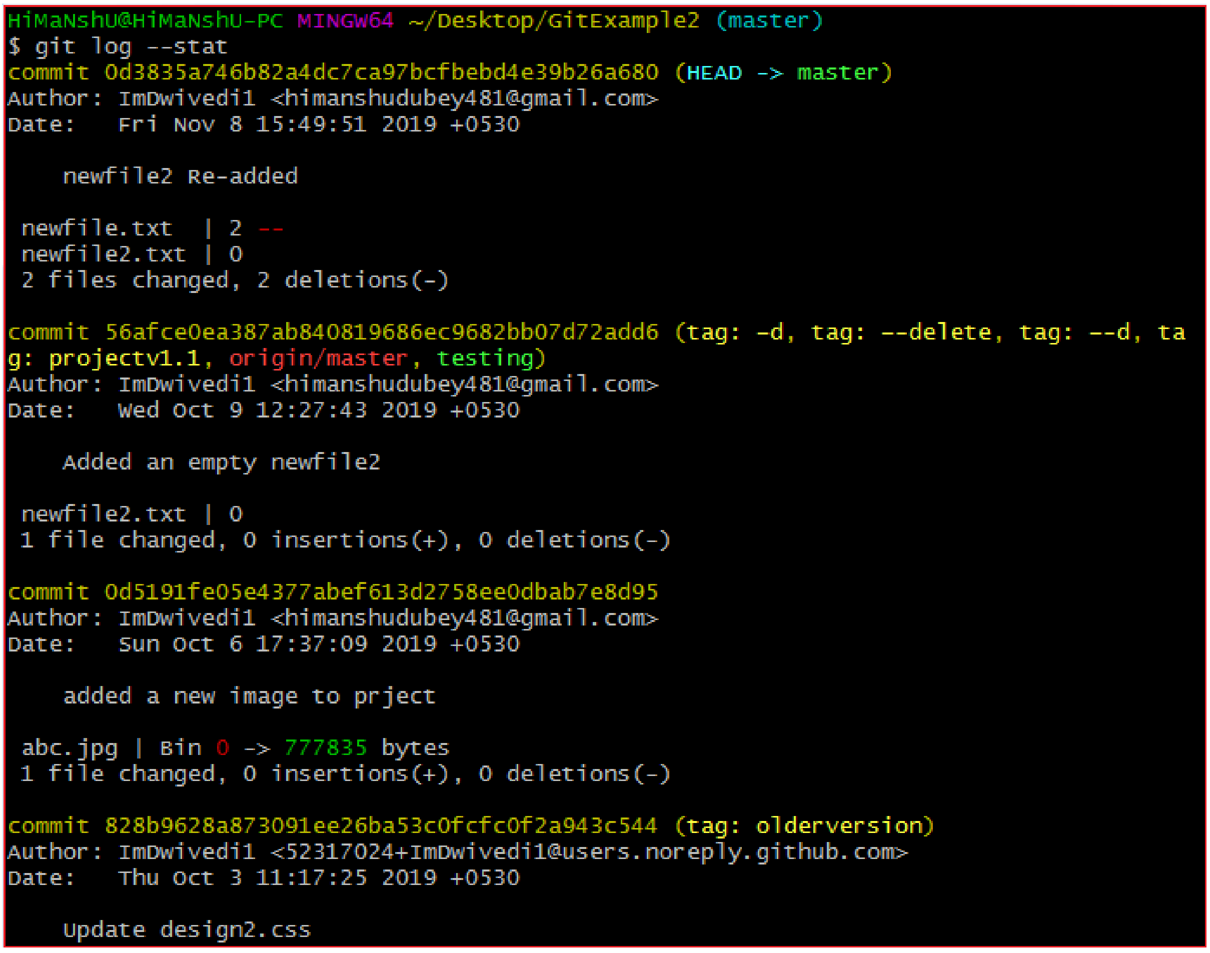
Generally, we can say that the stat option is used to display

- the modified files,
- The number of lines that have been added or removed
- A summary line of the total number of records changed
- The lines that have been added or removed.

It will be used as follows:

1. $ git log --stat

The above command will display the files that have been modified. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --stat
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

 newfile.txt  | 2 --
 newfile2.txt | 0
 2 files changed, 2 deletions(-)

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --d, ta
g: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

 newfile2.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to prject

 abc.jpg | Bin 0 -> 777835 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)

commit 828b9628a873091ee26ba53c0fcfc0f2a943c544 (tag: olderversion)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Thu Oct 3 11:17:25 2019 +0530

    Update design2.css
```

From the above output, we can see that all listed commits are modifications in the repository.

Git log P or Patch

The git log patch command displays the files that have been modified. It also shows the location of the added, removed, and updated lines.

It will be used as:

1. $ git log --patch

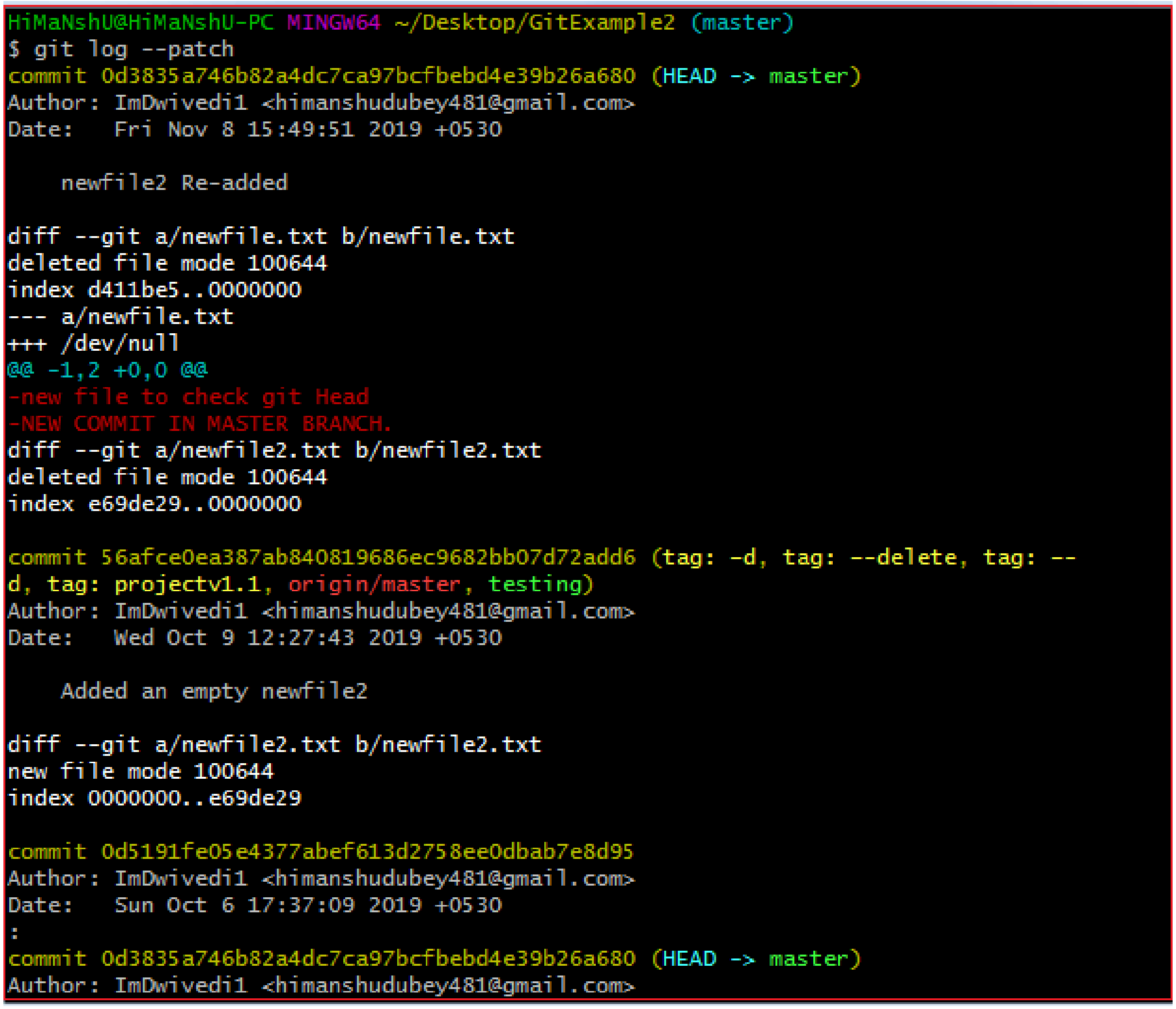
Or

1. $ git log -p

Generally, we can say that the --patch flag is used to display:

- Modified files
- The location of the lines that you added or removed
- Specific changes that have been made.

Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --patch
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

diff --git a/newfile.txt b/newfile.txt
deleted file mode 100644
index d411be5..0000000
--- a/newfile.txt
+++ /dev/null
@@ -1,2 +0,0 @@
-new file to check git Head
-NEW COMMIT IN MASTER BRANCH.
diff --git a/newfile2.txt b/newfile2.txt
deleted file mode 100644
index e69de29..0000000

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

diff --git a/newfile2.txt b/newfile2.txt
new file mode 100644
index 0000000..e69de29

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530
:
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
```

The above output is displaying the modified files with the location of lines that have been added or removed.

Git Log Graph

Git log command allows viewing your git log as a graph. To list the commits in the form of a graph, run the git log command with --graph option. It will run as follows:

1. $ git log --graph

To make the output more specific, you can combine this command with --oneline option. It will operate as follows:

1. $ git log --graph --oneline

Filtering the Commit History

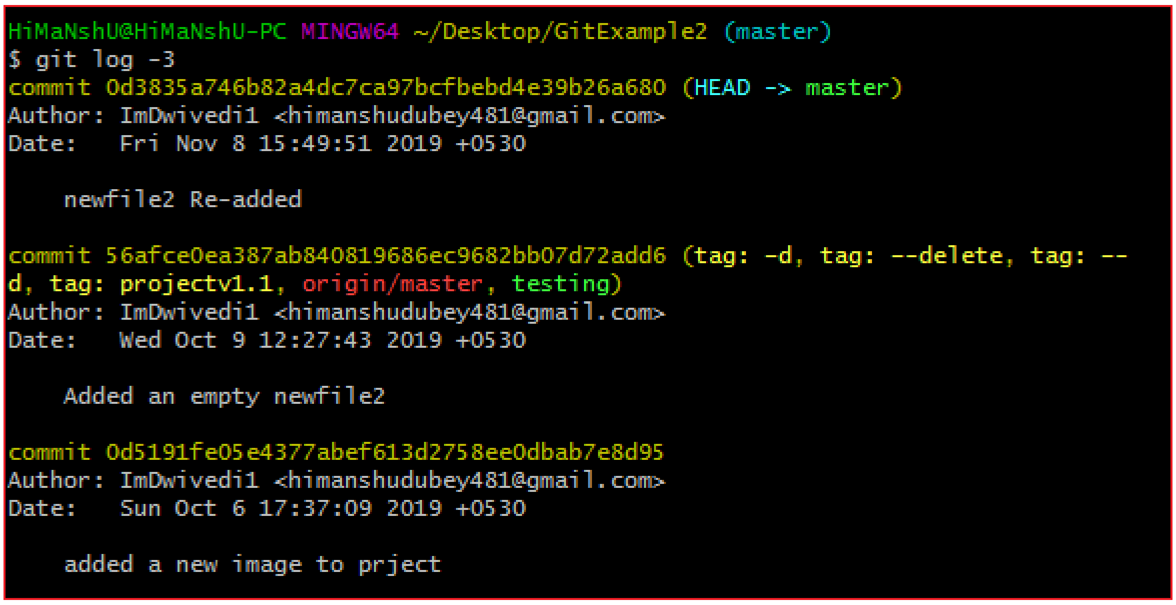
We can filter the output according to our needs. It's a unique feature of Git. We can apply many filters like amount, date, author, and more on output. Each filter has its specifications. They can be used for implementing some navigation operations on output.

Let's understand each of these filters in detail.

By Amount:

We can limit the number of output commit by using git log command. It is the most specific command. This command will remove the complexity if you are interested in fewer commits.

To limit the git log's output, including the -<n> option. If we want only the last three commit, then we can pass the argument -3 in the git log command. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log -3
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to prject
```

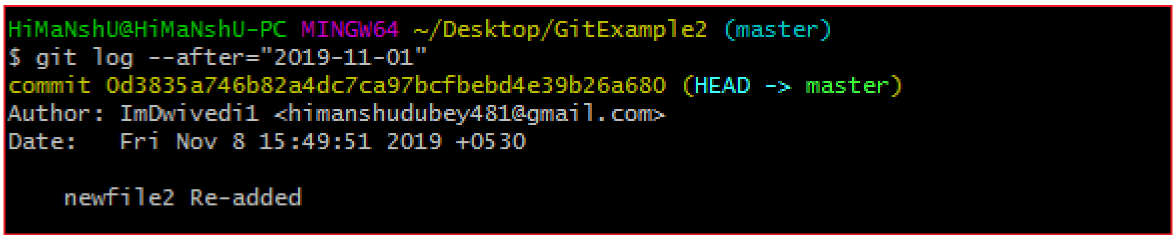
As we can see from the above output, we can limit the output of git log.

By Date and Time:

We can filter the output by date and time. We have to pass **--after** or **-before** argument to specify the date. These both argument accept a variety of date formats. It will run as follows:

1. $ git log --after="yy-mm-dd"

The above command will display all the commits made after the given date. Consider the below output:



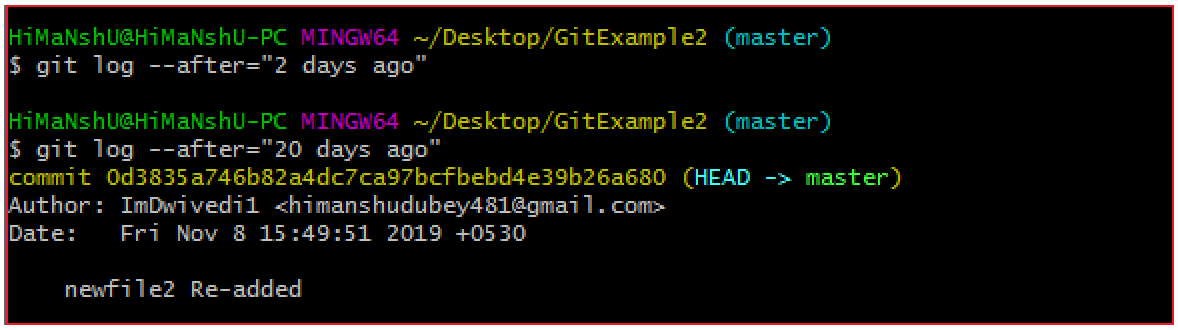
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --after="2019-11-01"
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added
```

The above command is listing all the commits after "2019-11-01".

We can also pass the applicable reference statement like "yesterday," "1 week ago", "21 days ago," and more. It will run as:

1. git log --after="21 days ago"

The above command will display the commits which have been made 21 days ago. Consider the below output:

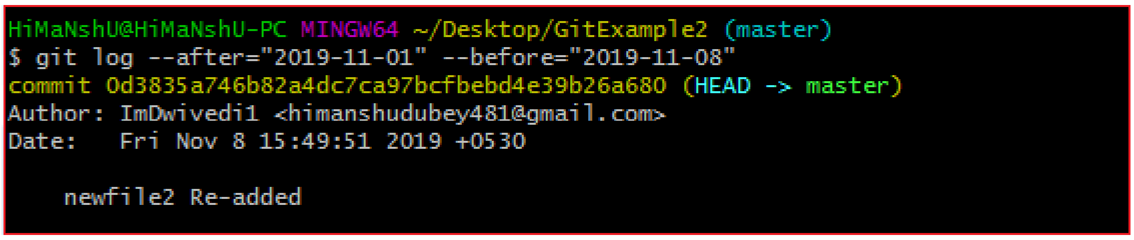


```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --after="2 days ago"

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --after="20 days ago"
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added
```

We can also track the commits between two dates. To track the commits that were created between two dates, pass a statement reference **--before** and **--after** the date. Suppose, we want to track the commits between "2019-11-01" and "2019-11-08". We will run the command as follows:

1. $ git log --after="2019-11-01" --before="2019-11-08 "

The above command will display the commits made between the dates. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --after="2019-11-01" --before="2019-11-08"
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added
```

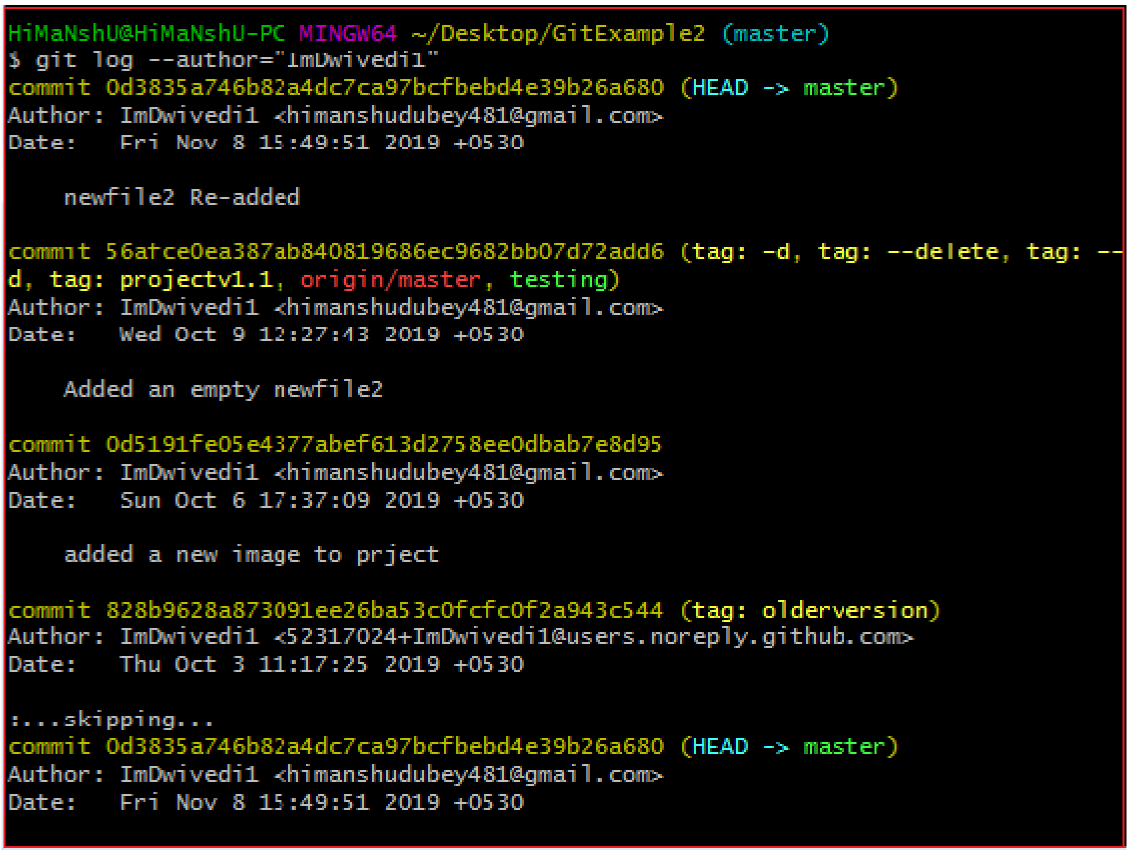
The above output is displaying the commits between the given period. We can use --since and --until instead of --after and --before. Because they are synonyms, respectively.

By Author:

We can filter the commits by a particular user. Suppose, we want to list the commits only made by a particular team member. We can use -author flag to filter the commits by author name. This command takes a regular expression and returns the list of commits made by authors that match that pattern. You can use the exact name instead of the pattern. This command will run as follows:

1. $ git log --author="Author name"

The above command will display all the commits made by the given author. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --author="ImDwivedi1"
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to prject

commit 828b9628a873091ee26ba53c0fcfc0f2a943c544 (tag: olderversion)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Thu Oct 3 11:17:25 2019 +0530

:...skipping...
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530
```

From the above output, we can see that all the commits by the author **ImDwivedi1** are listed.

We can use a string instead of a regular expression or exact name. Consider the below statement:

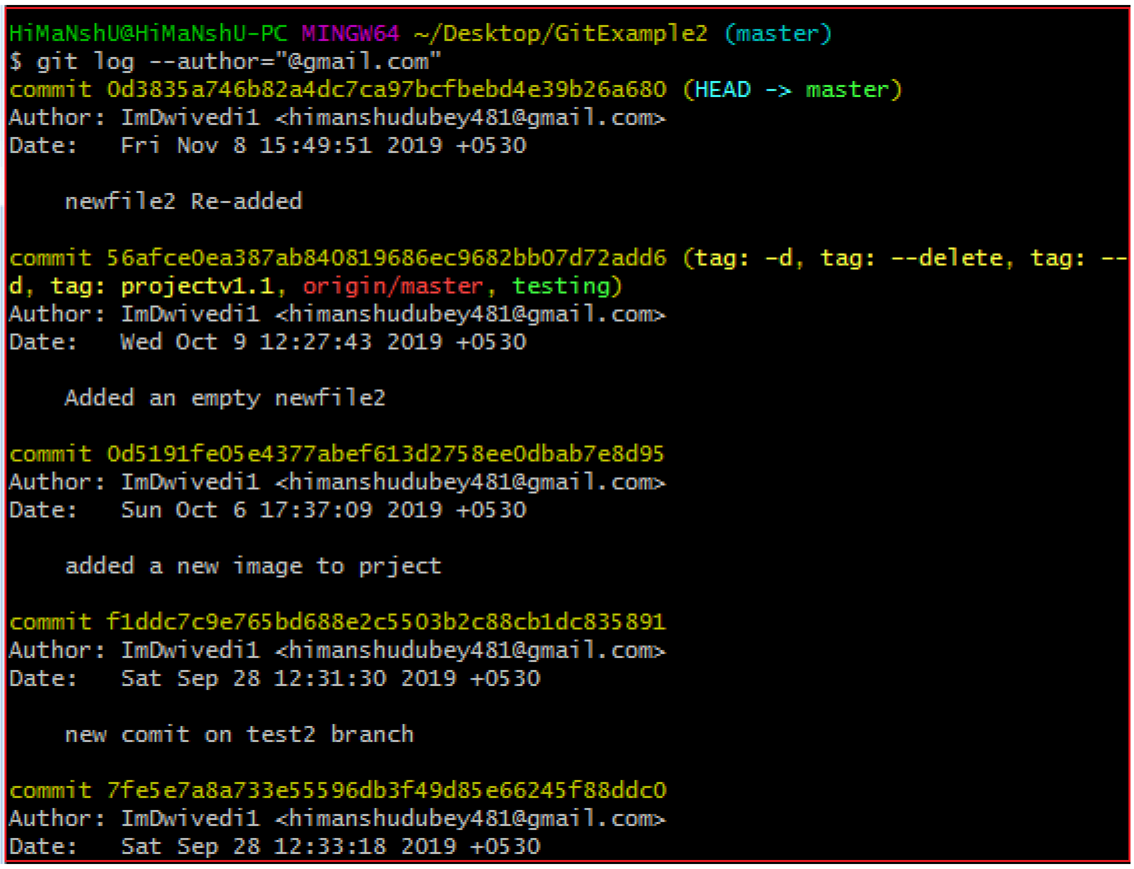
1. $ git log --author="Stephen"

The above statement will display all commits whose author includes the name, Stephen. The author's name doesn't need to be an exact match; it just has the specified phrase.

As we know, the author's email is also involved with the author's name, so that we can use the author's email as the pattern or exact search. Suppose, we want to track the commits by the authors whose email service is google. To do so, we can use wild cards as "@gmail.com." Consider the below statement:

1. $ git log -author="@gmail.com"

The above command will display the commits by authors as given in the pattern. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --author="@gmail.com"
commit 0d3835a746b82a4dc7ca97bcfbebd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to prject

commit f1ddc7c9e765bd688e2c5503b2c88cb1dc835891
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:31:30 2019 +0530

    new comit on test2 branch

commit 7fe5e7a8a733e55596db3f49d85e66245f88ddc0
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:33:18 2019 +0530
```

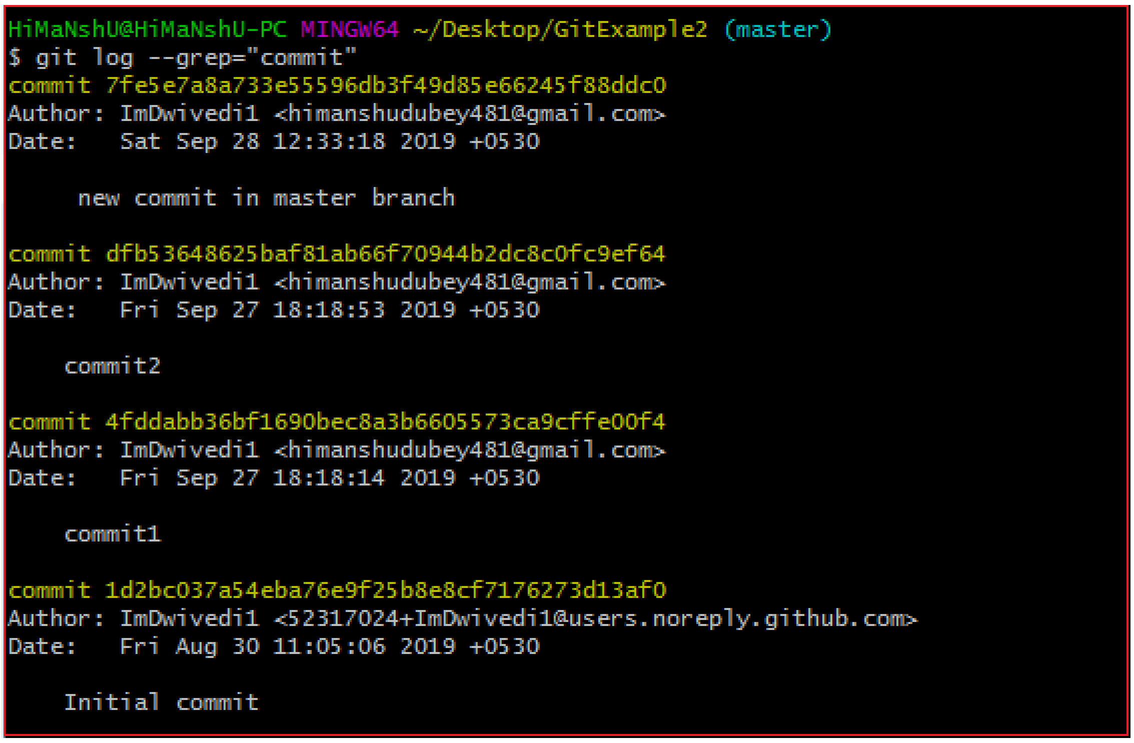
By Commit message:

To filter the commits by the commit message. We can use the grep option, and it will work as the author option.

It will run as follows:

1. $ git log --grep=" Commit message."

We can use the short form of commit message instead of a complete message. Consider the below output.



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log --grep="commit"
commit 7fe5e7a8a733e55596db3f49d85e66245f88ddc0
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:33:18 2019 +0530

     new commit in master branch

commit dfb53648625baf81ab66f70944b2dc8c0fc9ef64
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Sep 27 18:18:53 2019 +0530

    commit2

commit 4fddabb36bf1690bec8a3b6605573ca9cffe00f4
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Sep 27 18:18:14 2019 +0530

    commit1

commit 1d2bc037a54eba76e9f25b8e8cf7176273d13af0
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Fri Aug 30 11:05:06 2019 +0530

    Initial commit
```

The above output is displaying all the commits that contain the word commit in its commit message.

There are many other filtering options available like we can filter by file name, content, and more.

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.