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.

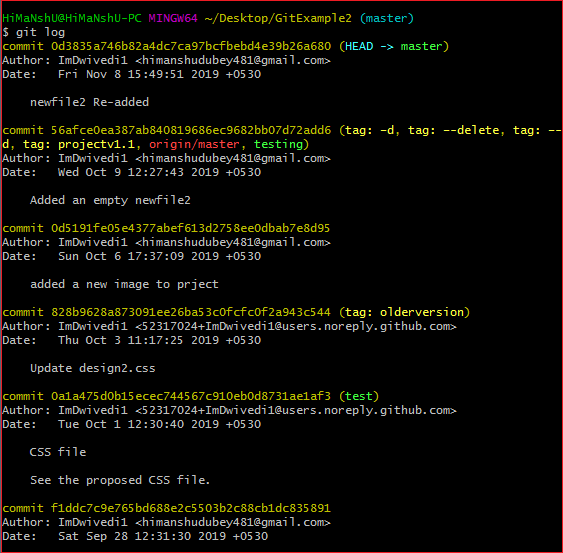
The solution to this problem is to **press** the **q (Q for quit)**. It will quit you from the situation and back you to the command line. Now, you can perform any of the commands.

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:



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.

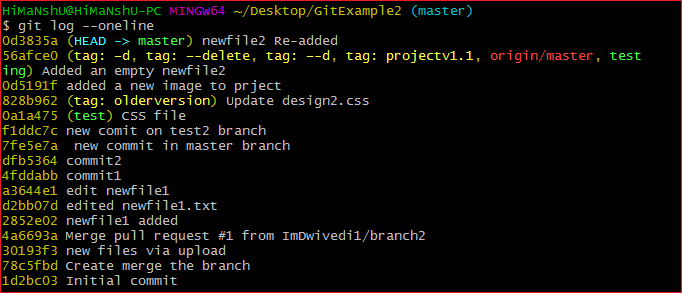
It will be used as follows:

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:



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.

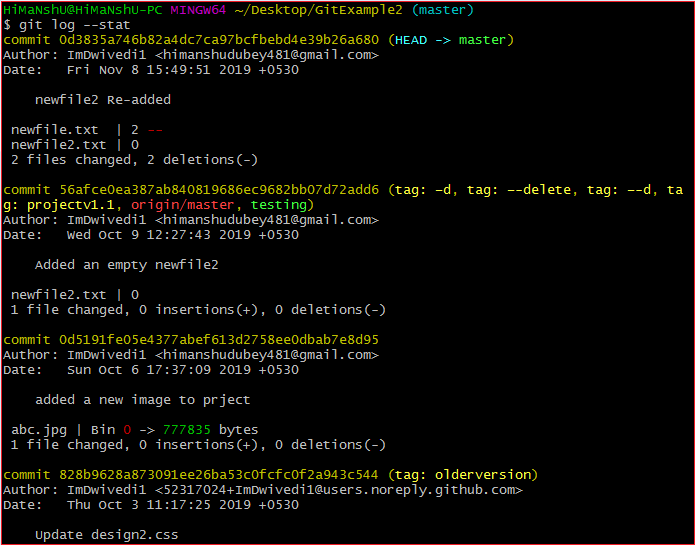
Generally, we can say that the stat option is used to display

* 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:



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

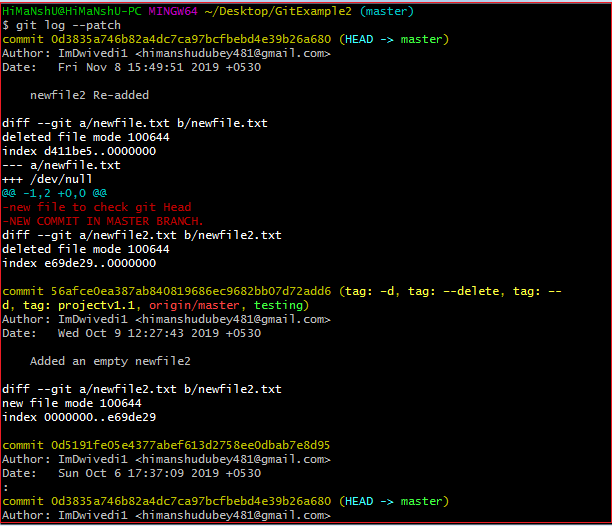
Or

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:



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

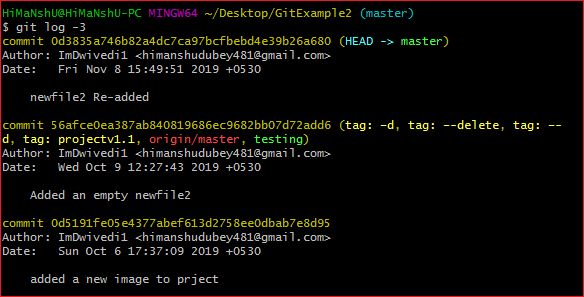
We can filter the output according to our needs. It's a unique feature of Git. We can apply many filters like amount, date, author, and more on output. Each filter has its specifications. They can be used for implementing some navigation operations on output.

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:



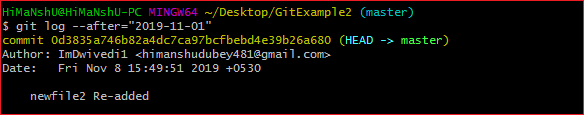
As we can see from the above output, we can limit the output of git log.

**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:

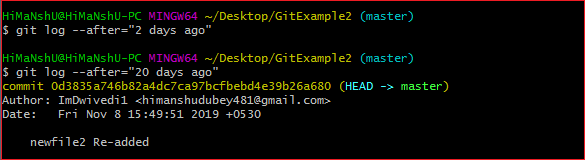


The above command is listing all the commits after "2019-11-01".

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"

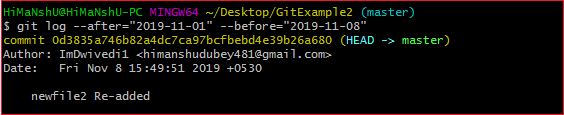
The above command will display the commits which have been made 21 days ago. Consider the below output:



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:



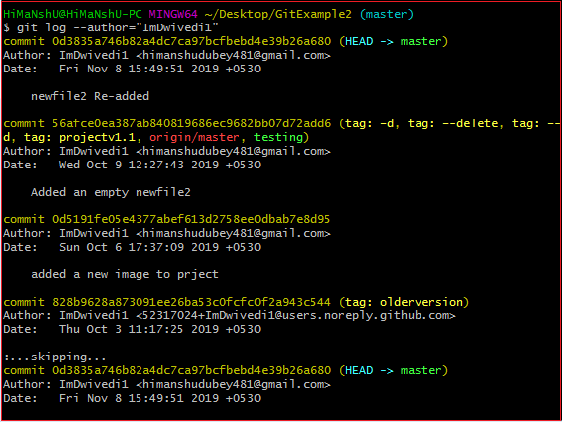
The above output is displaying the commits between the given period. We can use --since and --until instead of --after and --before. Because they are synonyms, respectively.

**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:



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:

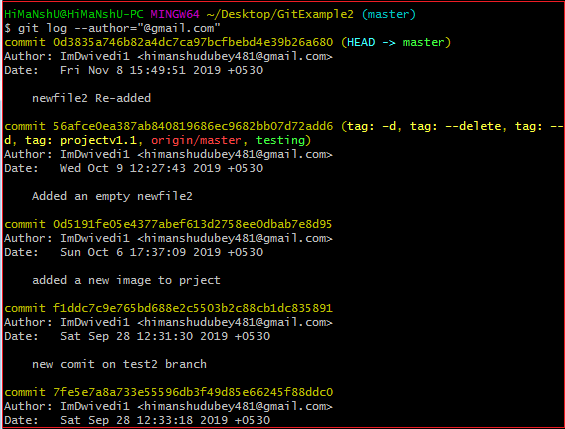
1. $ git log --author="Stephen"

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:



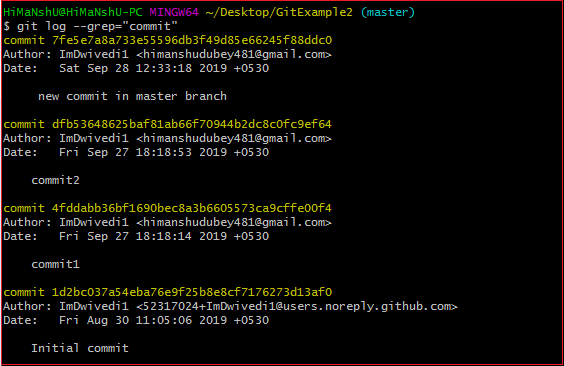
**By Commit message:**

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.



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.