git

## CONTENTS

# Git log

# Basic Git log

- Git log command is one of the most usual commands of git.
- The basic git log command will display the most recent commits and the status of the head.
- **$ git log**

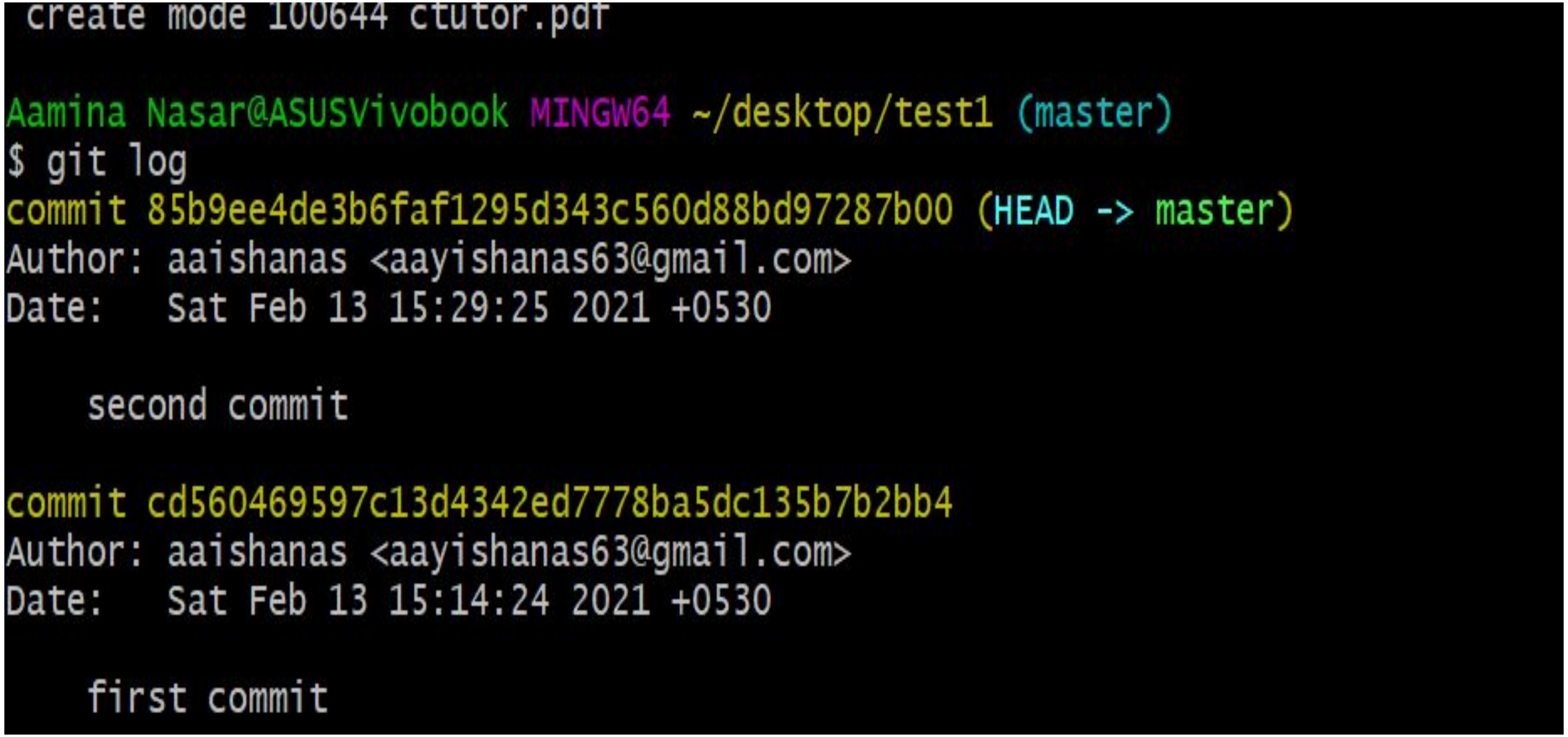
The above command will display the last commits.

## Git log

- Git log is a utility tool to review and read a history of everything that happens to a repository.

  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.

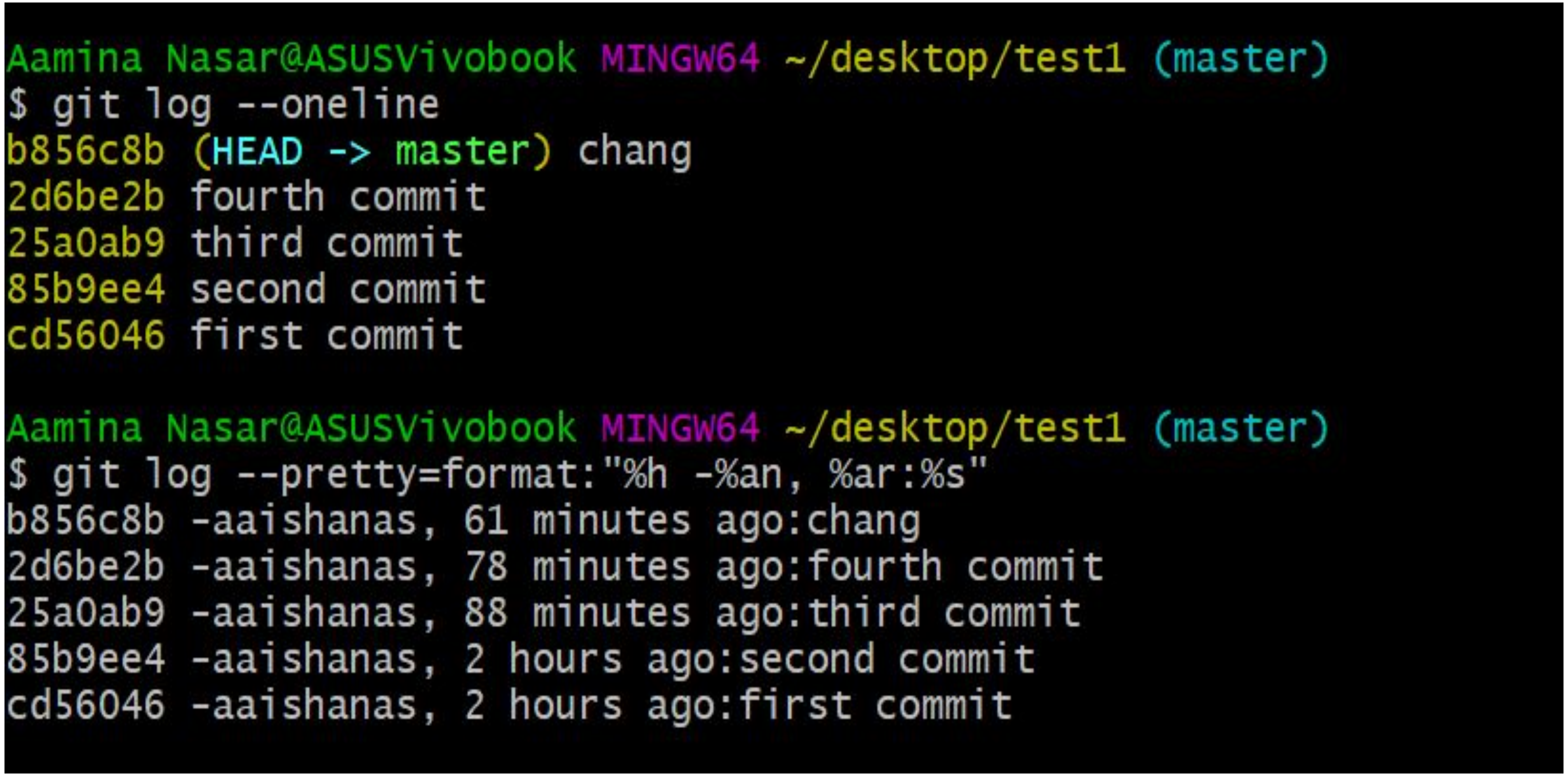


```
 create mode 100644 ctutor.pdf

Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log
commit 85b9ee4de3b6faf1295d343c560d88bd97287b00 (HEAD -> master)
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 15:29:25 2021 +0530

    second commit

commit cd560469597c13d4342ed7778ba5dc135b7b2bb4
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 15:14:24 2021 +0530

    first commit
```

## Git Log Oneline

**$ git log --oneline** flag causes git log to display:

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



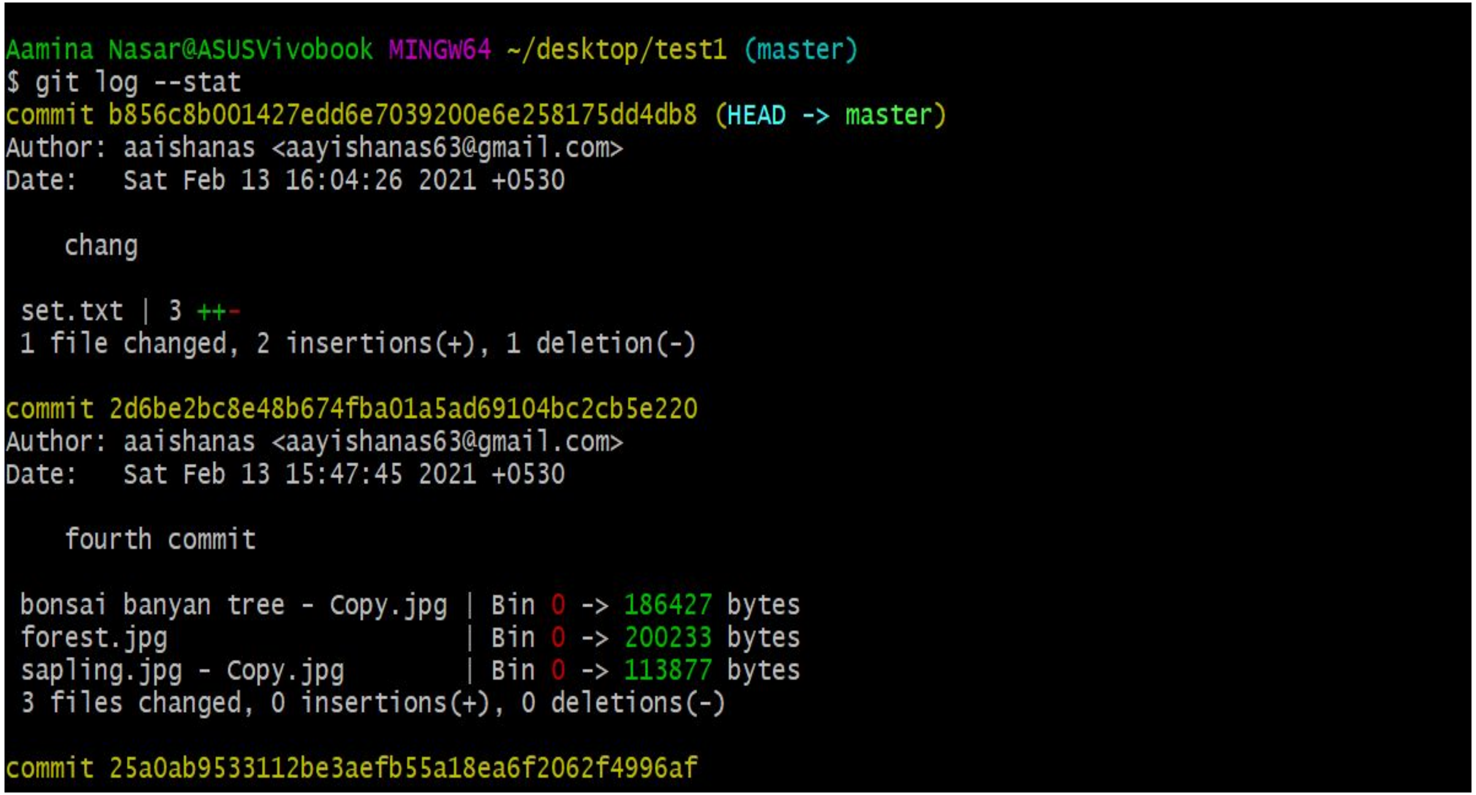
```
Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --oneline
b856c8b (HEAD -> master) chang
2d6be2b fourth commit
25a0ab9 third commit
85b9ee4 second commit
cd56046 first commit

Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --pretty=format:"%h -%an, %ar:%s"
b856c8b -aaishanas, 61 minutes ago:chang
2d6be2b -aaishanas, 78 minutes ago:fourth commit
25a0ab9 -aaishanas, 88 minutes ago:third commit
85b9ee4 -aaishanas, 2 hours ago:second commit
cd56046 -aaishanas, 2 hours ago:first commit
```

## Git log stat

- 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:

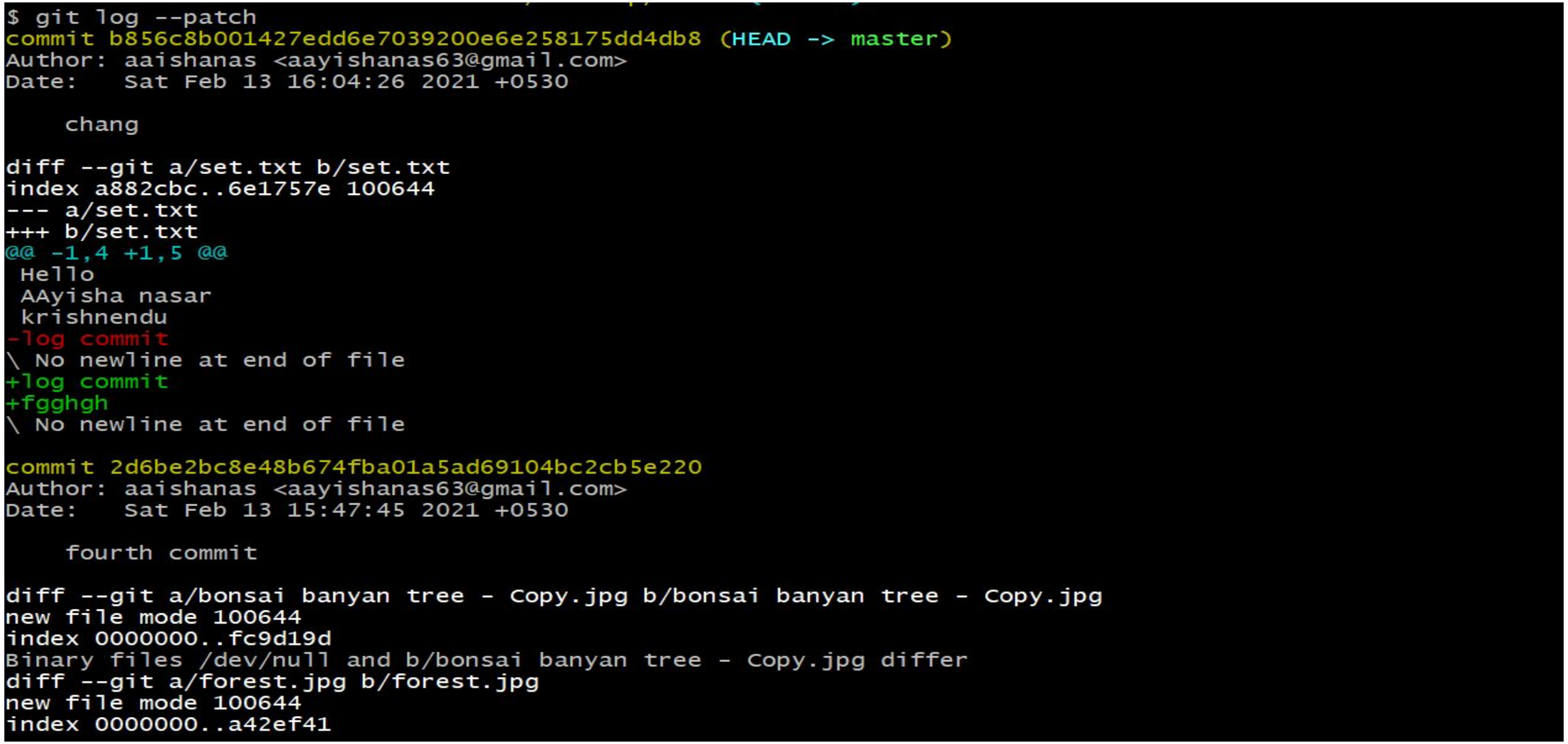
1. `$ git log --stat`



```
Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --stat
commit b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master)
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 16:04:26 2021 +0530

    chang

 set.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

commit 2d6be2bc8e48b674fba01a5ad69104bc2cb5e220
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 15:47:45 2021 +0530

    fourth commit

 bonsai banyan tree - Copy.jpg | Bin 0 -> 186427 bytes
 forest.jpg                    | Bin 0 -> 200233 bytes
 sapling.jpg - Copy.jpg        | Bin 0 -> 113877 bytes
 3 files changed, 0 insertions(+), 0 deletions(-)

commit 25a0ab9533112be3aefb55a18ea6f2062f4996af
```

## Git log P or Patch

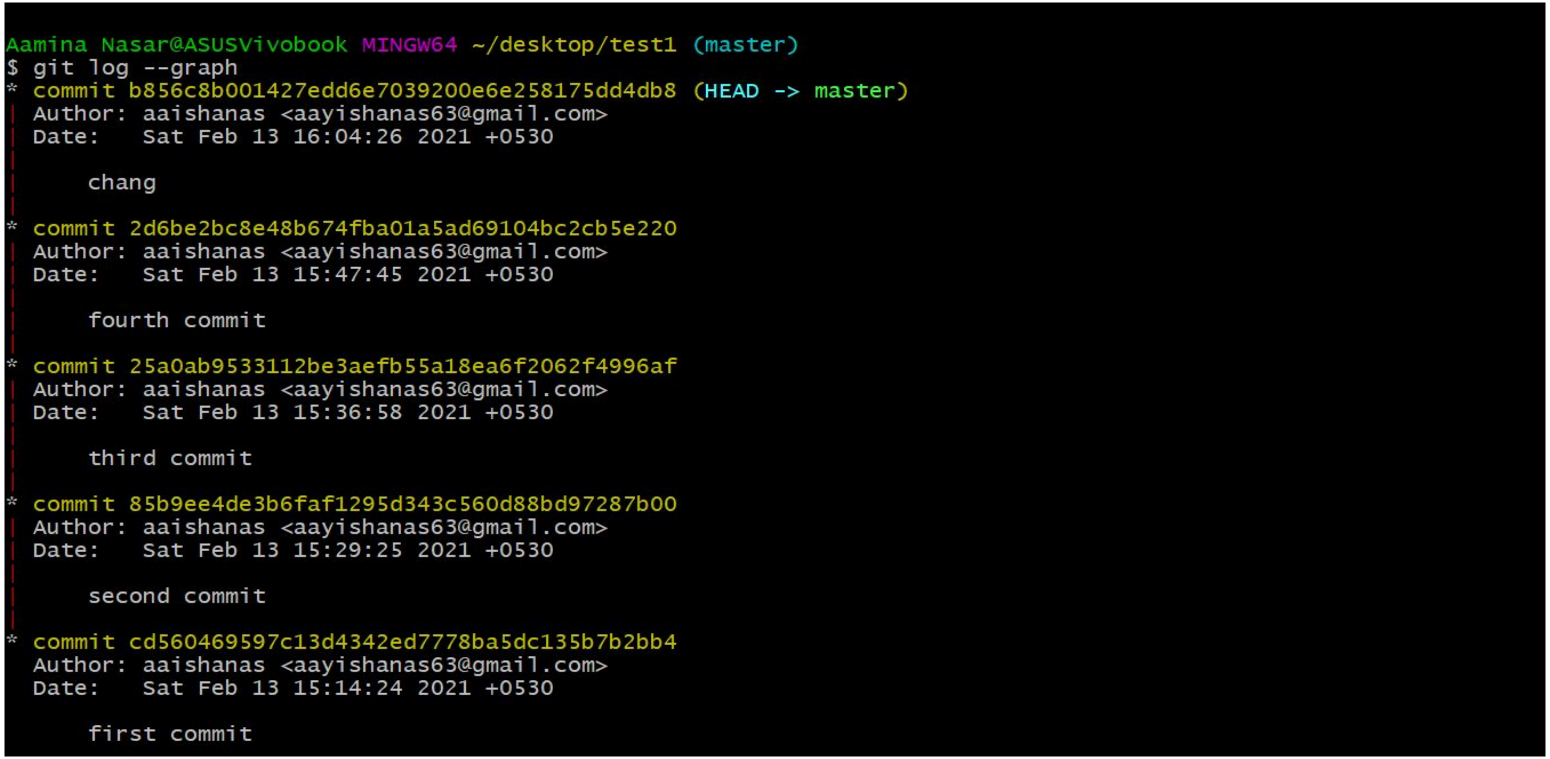
- The git log patch command displays the files that have been modified.
- It also shows the location of the added, removed, and updated lines.
- **$ git log --patch  Or**
- **$ git log -p**

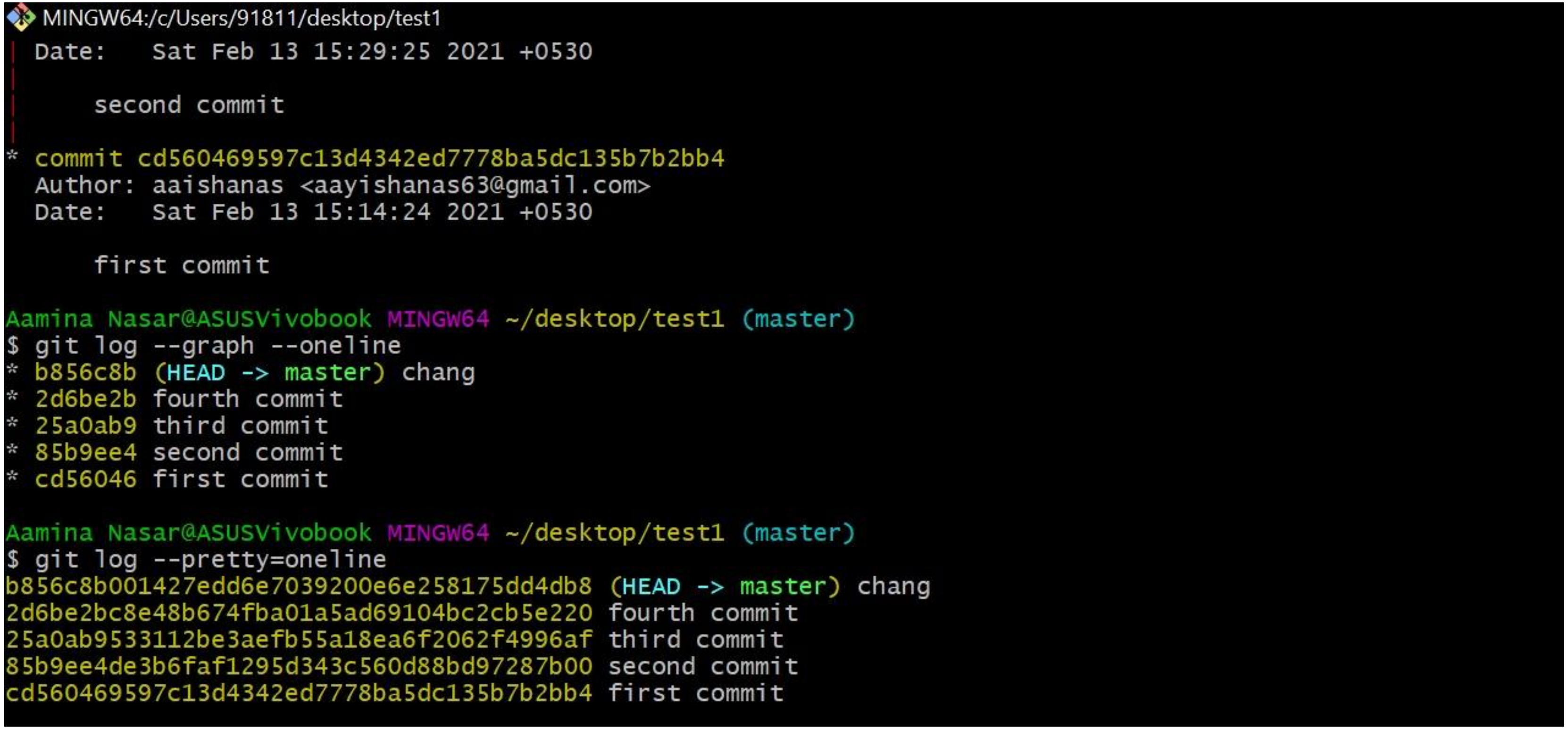


```
$ git log --patch
commit b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master)
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 16:04:26 2021 +0530

    chang

diff --git a/set.txt b/set.txt
index a882cbc..6e1757e 100644
--- a/set.txt
+++ b/set.txt
@@ -1,4 +1,5 @@
 Hello
 AAyisha nasar
 krishnendu
-log commit
\ No newline at end of file
+log commit
+fgghgh
\ No newline at end of file

commit 2d6be2bc8e48b674fba01a5ad69104bc2cb5e220
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 15:47:45 2021 +0530

    fourth commit

diff --git a/bonsai banyan tree - Copy.jpg b/bonsai banyan tree - Copy.jpg
new file mode 100644
index 0000000..fc9d19d
Binary files /dev/null and b/bonsai banyan tree - Copy.jpg differ
diff --git a/forest.jpg b/forest.jpg
new file mode 100644
index 0000000..a42ef41
```

## Git Log Graph

- Git log command allows viewing  git log as a graph.
- To list the commits in the form of a graph, run -
- **$ git log --graph**
- To make the output more specific,combine this command with --oneline option. It will operate as follows:
- **$ git log --graph --oneline**


```
Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --graph
* commit b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master)
| Author: aaishanas <aayishanas63@gmail.com>
| Date:   Sat Feb 13 16:04:26 2021 +0530
|
|     chang
|
* commit 2d6be2bc8e48b674fba01a5ad69104bc2cb5e220
| Author: aaishanas <aayishanas63@gmail.com>
| Date:   Sat Feb 13 15:47:45 2021 +0530
|
|     fourth commit
|
* commit 25a0ab9533112be3aefb55a18ea6f2062f4996af
| Author: aaishanas <aayishanas63@gmail.com>
| Date:   Sat Feb 13 15:36:58 2021 +0530
|
|     third commit
|
* commit 85b9ee4de3b6faf1295d343c560d88bd97287b00
| Author: aaishanas <aayishanas63@gmail.com>
| Date:   Sat Feb 13 15:29:25 2021 +0530
|
|     second commit
|
* commit cd560469597c13d4342ed7778ba5dc135b7b2bb4
  Author: aaishanas <aayishanas63@gmail.com>
  Date:   Sat Feb 13 15:14:24 2021 +0530

      first commit
```



```
MINGW64:/c/Users/91811/desktop/test1
| Date:   Sat Feb 13 15:29:25 2021 +0530
|
|     second commit
|
* commit cd560469597c13d4342ed7778ba5dc135b7b2bb4
  Author: aaishanas <aayishanas63@gmail.com>
  Date:   Sat Feb 13 15:14:24 2021 +0530

      first commit

Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --graph --oneline
* b856c8b (HEAD -> master) chang
* 2d6be2b fourth commit
* 25a0ab9 third commit
* 85b9ee4 second commit
* cd56046 first commit

Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --pretty=oneline
b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master) chang
2d6be2bc8e48b674fba01a5ad69104bc2cb5e220 fourth commit
25a0ab9533112be3aefb55a18ea6f2062f4996af third commit
85b9ee4de3b6faf1295d343c560d88bd97287b00 second commit
cd560469597c13d4342ed7778ba5dc135b7b2bb4 first commit
```

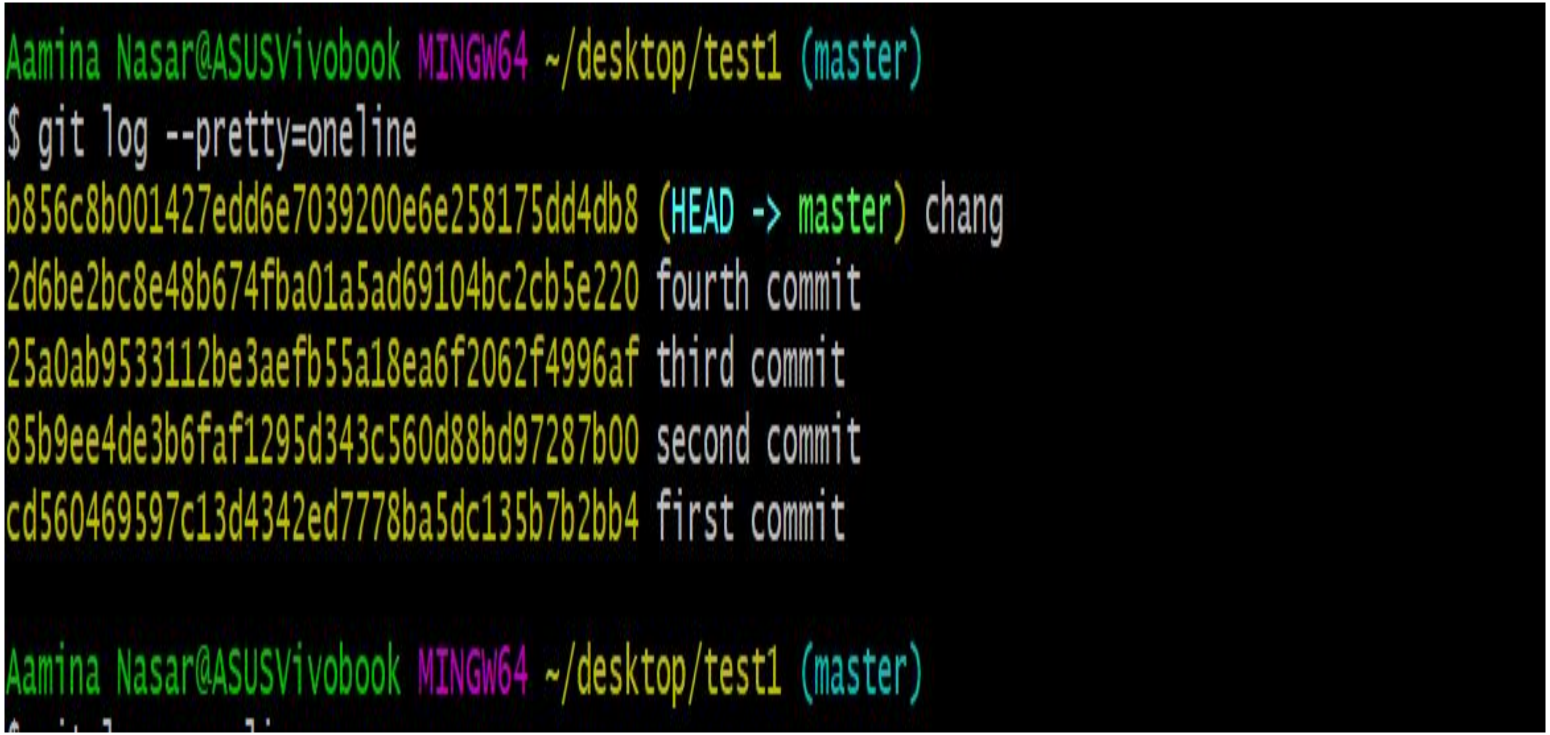
## Git log --pretty=online

- The --pretty=online parameter which makes it all fit on a single line.
- **$ git log --**

  **pretty=format:"%h - %an, %ar : %s"**

## Abbreviations

- %H Commit hash
- %h Abbreviated commit hash
- %T Tree hash
- %t Abbreviated tree hash
- %P Parent hash
- %p Abbreviated parent hash
- %an Author name
- %ae Author e-mail
- %ad Author date (format respects the –date= option
- %ar Author date, relative
- %cn Committer name
- %ce Committer e-mail
- %cd Committer date
- %cr Committer date, relative
- %s Subject



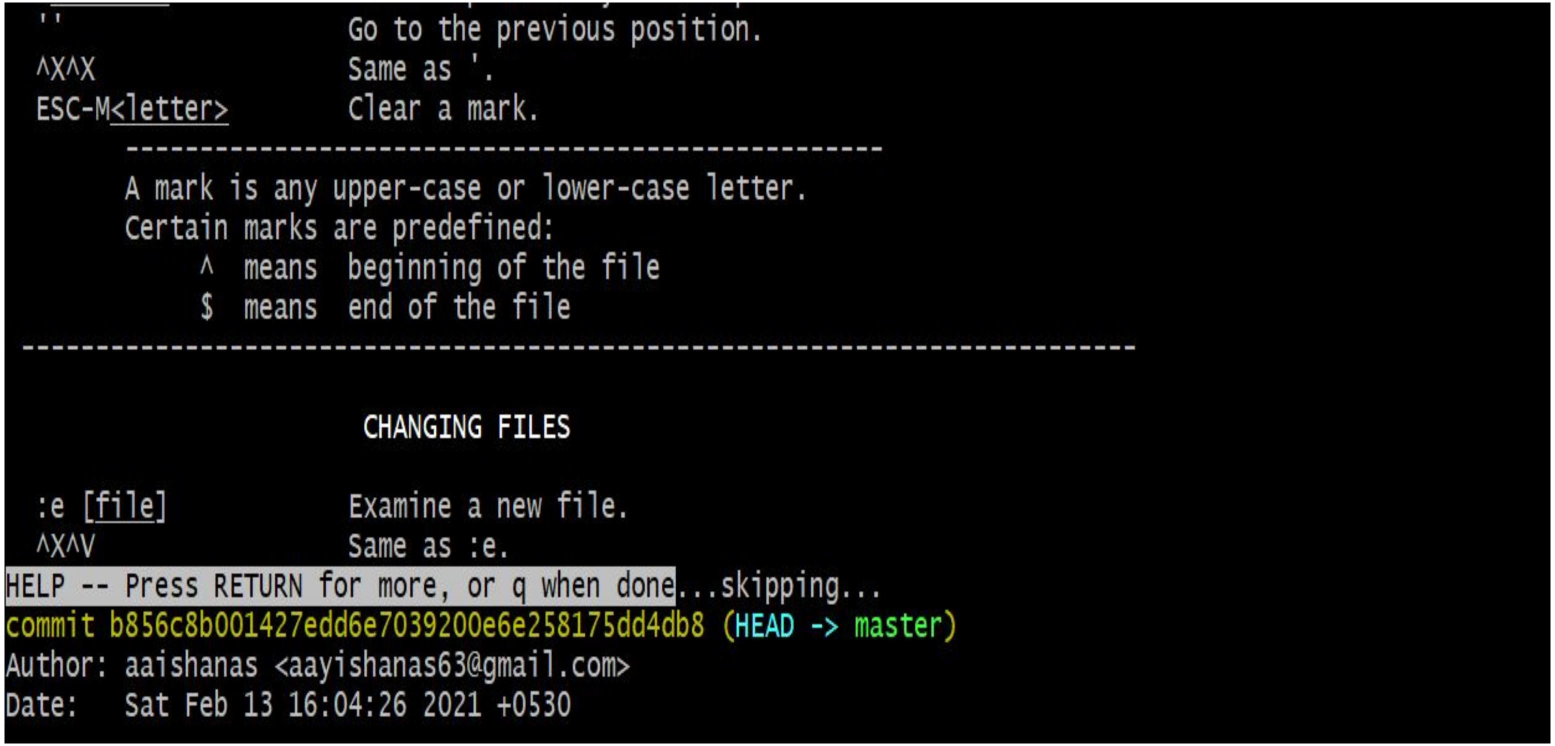
```
Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
$ git log --pretty=oneline
b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master) chang
2d6be2bc8e48b674fba01a5ad69104bc2cb5e220 fourth commit
25a0ab9533112be3aefb55a18ea6f2062f4996af third commit
85b9ee4de3b6faf1295d343c560d88bd97287b00 second commit
cd560469597c13d4342ed7778ba5dc135b7b2bb4 first commit


Aamina Nasar@ASUSVivobook MINGW64 ~/desktop/test1 (master)
```

## How to Exit the git log Command?

- There may be a situation that occurs, you run the git log command, and you stuck there.

- 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)**.



```
  ''                    Go to the previous position.
  ^X^X                  Same as '.
  ESC-M<letter>         Clear a mark.
        ---------------------------------------------------
        A mark is any upper-case or lower-case letter.
        Certain marks are predefined:
             ^  means  beginning of the file
             $  means  end of the file
 ---------------------------------------------------------------------------


                         CHANGING FILES

  :e [file]             Examine a new file.
  ^X^V                  Same as :e.
HELP -- Press RETURN for more, or q when done...skipping...
commit b856c8b001427edd6e7039200e6e258175dd4db8 (HEAD -> master)
Author: aaishanas <aayishanas63@gmail.com>
Date:   Sat Feb 13 16:04:26 2021 +0530
```

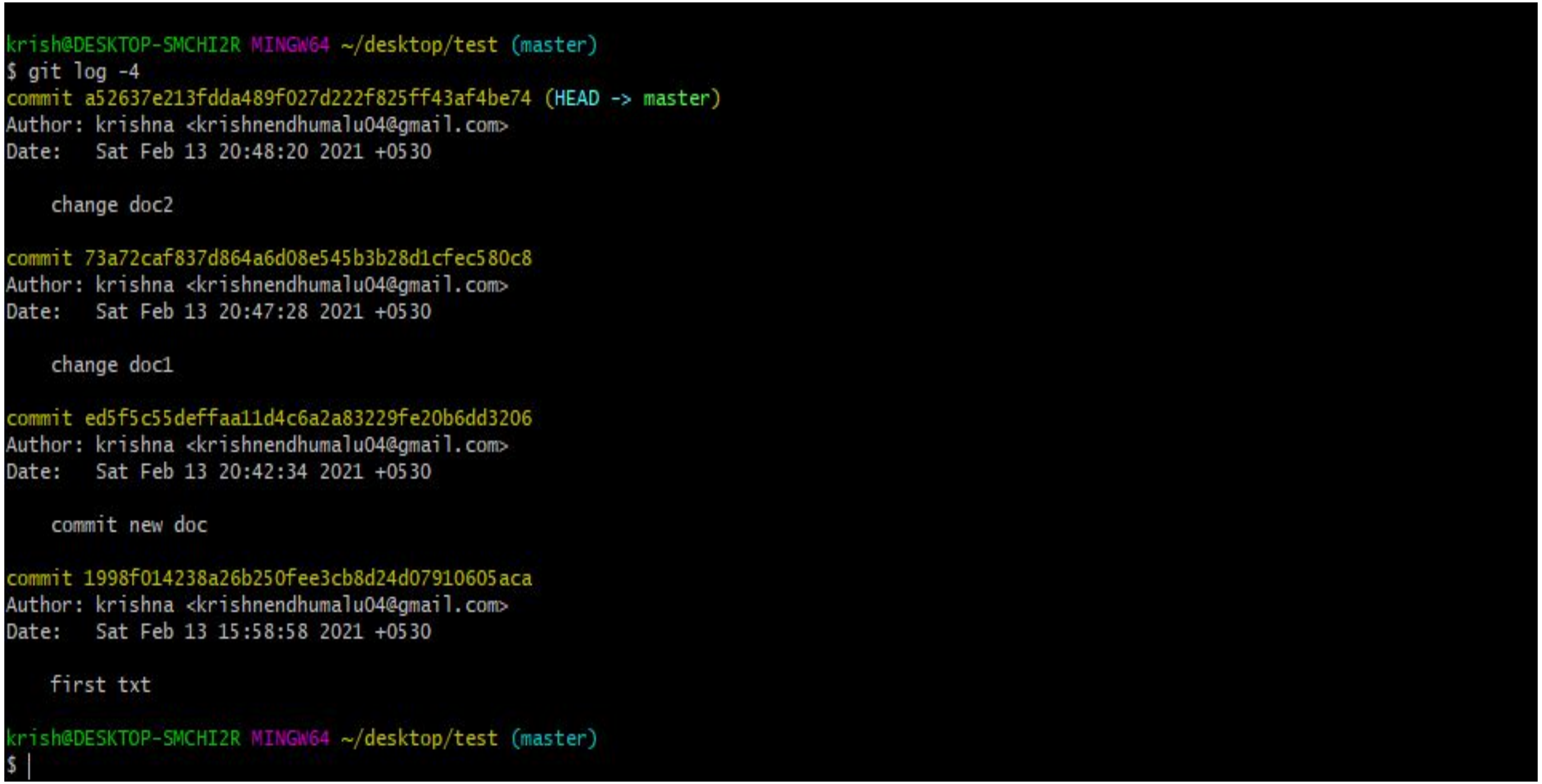
## Limiting Log Output

- We can filter the output according to our needs.
- Git log takes a number of useful limiting options
- It's a unique feature of Git.
- We can apply many filters like amount, date, author, and commit message.
- Each filter has its specifications.

## 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
- To limit the git log's output, including the -<n> option

Eg:If we want only the last four commit, then we can pass the argument -4 in the git log command.



```
krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log -4
commit a52637e213fdda489f027d222f825ff43af4be74 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:48:20 2021 +0530

    change doc2

commit 73a72caf837d864a6d08e545b3b28d1cfec580c8
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:47:28 2021 +0530

    change doc1

commit ed5f5c55deffaa11d4c6a2a83229fe20b6dd3206
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:42:34 2021 +0530

    commit new doc

commit 1998f014238a26b250fee3cb8d24d07910605aca
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 15:58:58 2021 +0530

    first txt

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ |
```

## 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.

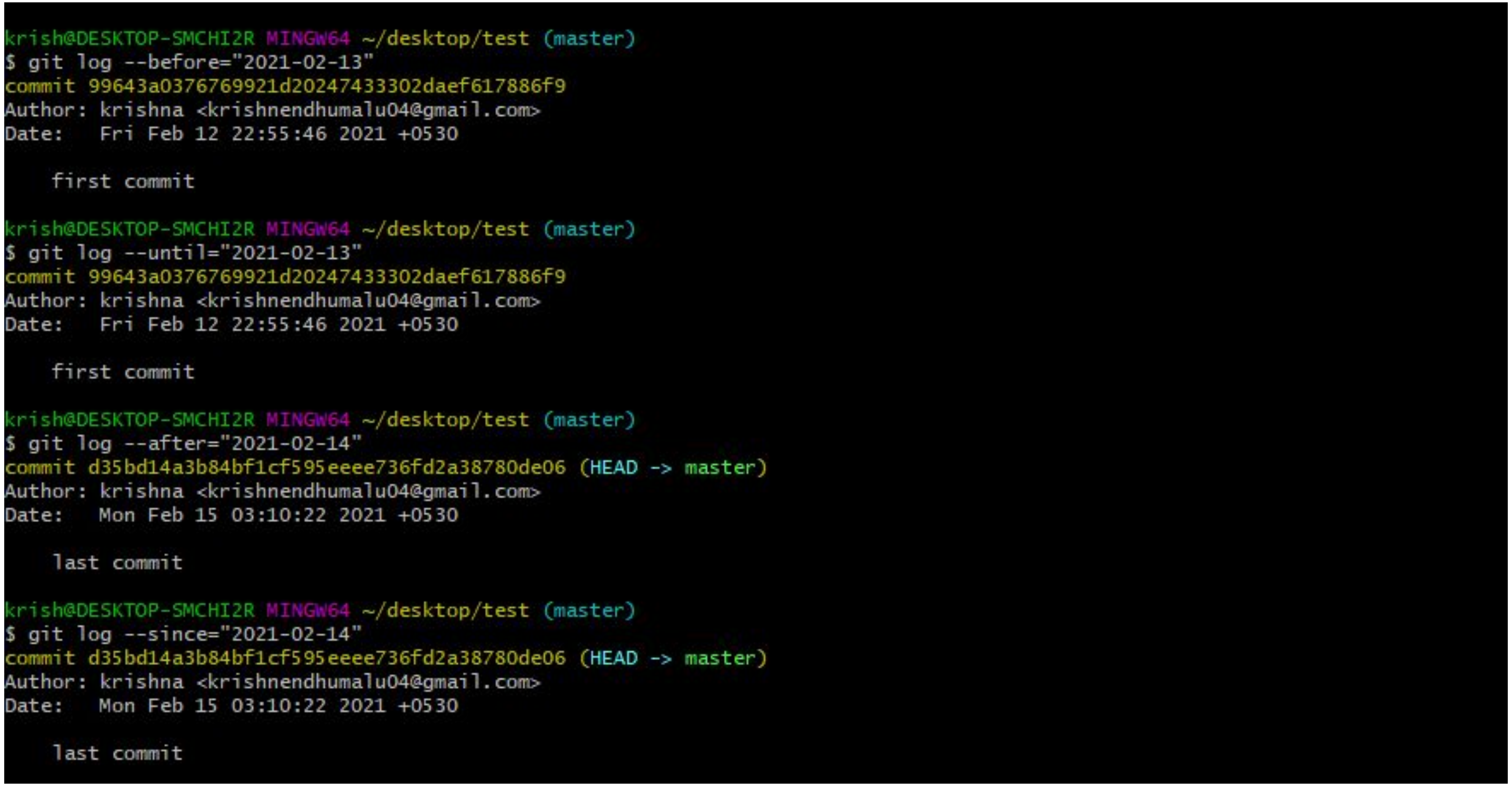
```
$ git log --after="yy-mm-dd"
```

- To track the commits that were created between two dates, pass a statement reference **--before** and **--after** the date".

Eg:`$ git log --after="2019-11-01" --before="2019-11-08 "`

The above command will display the commits made between the dates.

- We can use --since and --until instead of --after and --before.

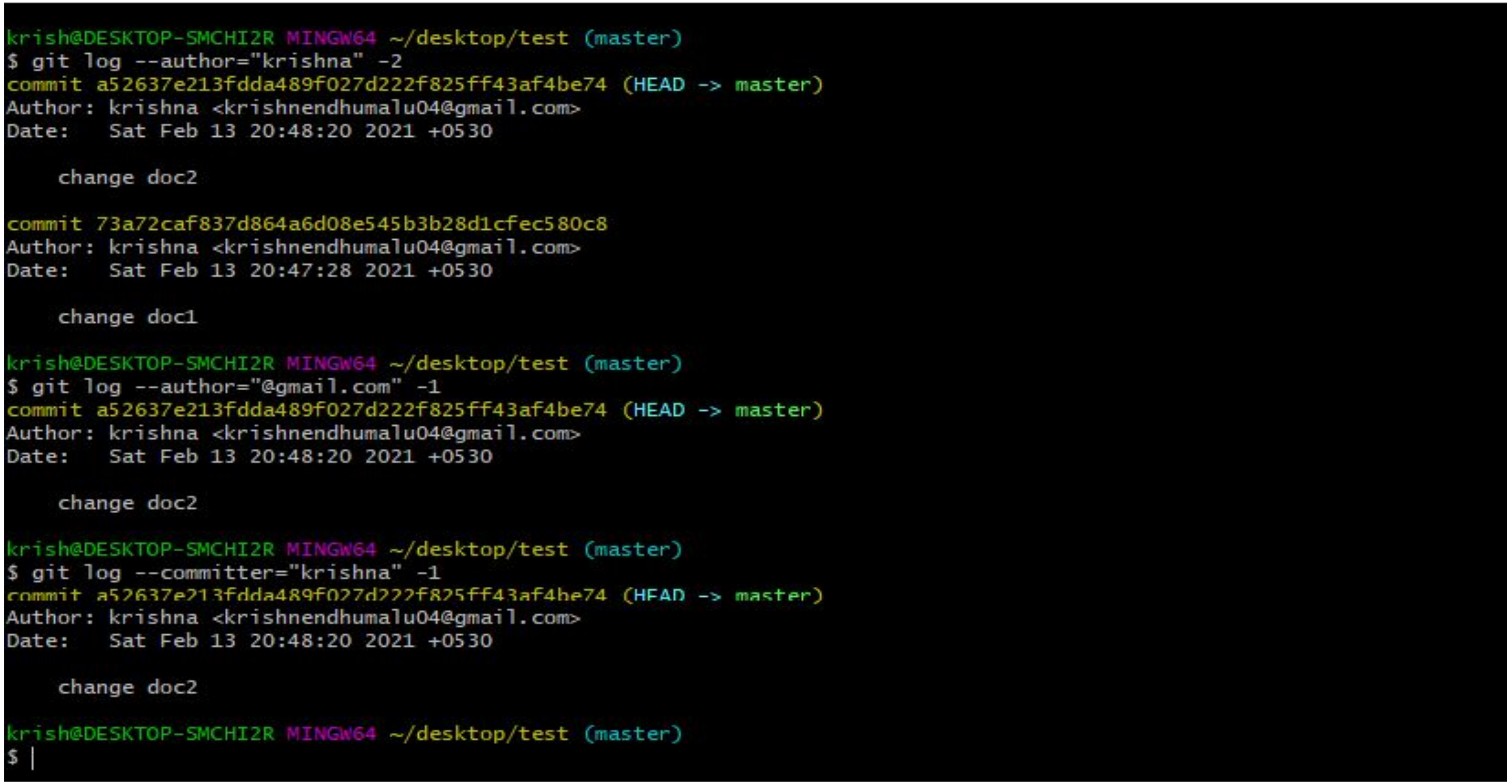


```
krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --before="2021-02-13"
commit 99643a0376769921d20247433302daef617886f9
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Fri Feb 12 22:55:46 2021 +0530

    first commit

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --until="2021-02-13"
commit 99643a0376769921d20247433302daef617886f9
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Fri Feb 12 22:55:46 2021 +0530

    first commit

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --after="2021-02-14"
commit d35bd14a3b84bf1cf595eeee736fd2a38780de06 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Mon Feb 15 03:10:22 2021 +0530

    last commit

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --since="2021-02-14"
commit d35bd14a3b84bf1cf595eeee736fd2a38780de06 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Mon Feb 15 03:10:22 2021 +0530

    last commit
```

## By Author:

- We can filter the commits by a particular user.
- 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.
- **$ git log --author="Author name"**

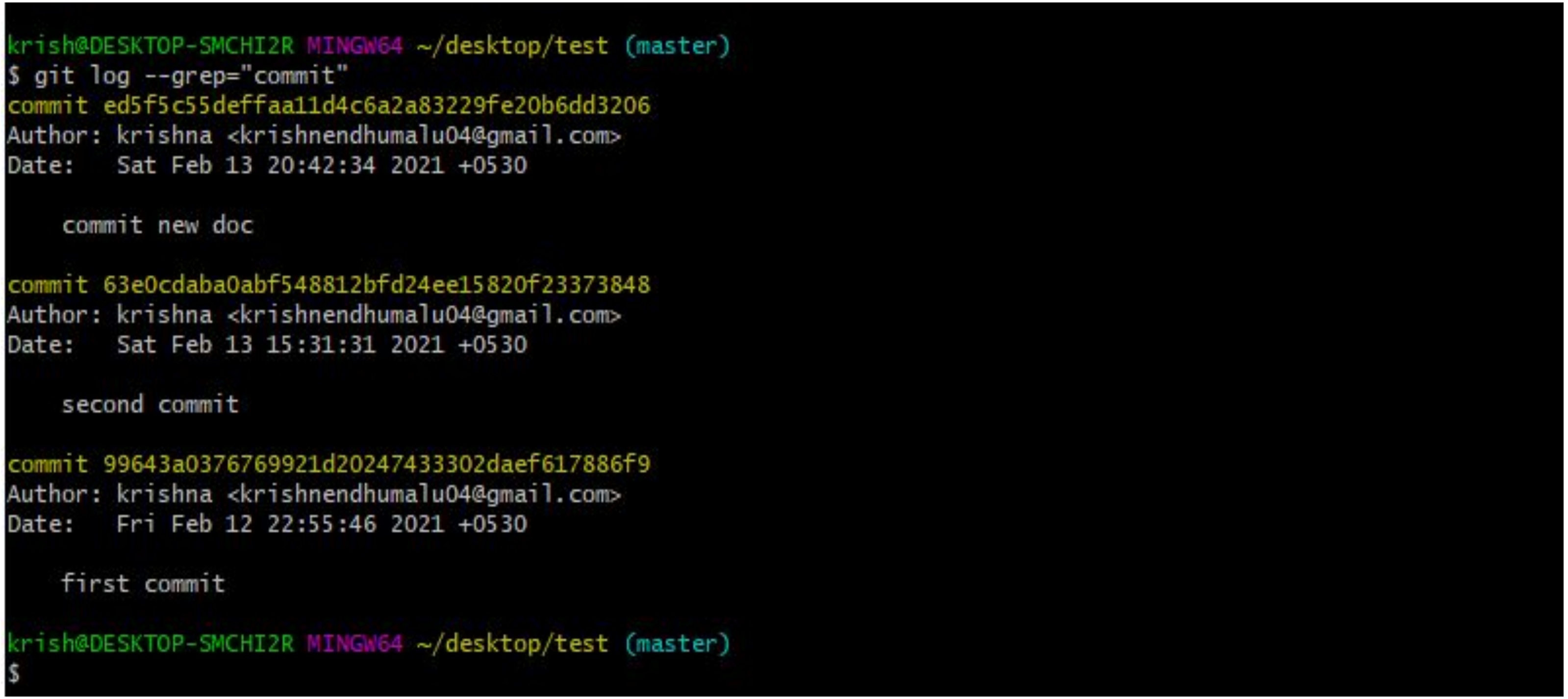
The above command will display all the commits made by the given author.



```
krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --author="krishna" -2
commit a52637e213fdda489f027d222f825ff43af4be74 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:48:20 2021 +0530

    change doc2

commit 73a72caf837d864a6d08e545b3b28d1cfec580c8
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:47:28 2021 +0530

    change doc1

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --author="@gmail.com" -1
commit a52637e213fdda489f027d222f825ff43af4be74 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:48:20 2021 +0530

    change doc2

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --committer="krishna" -1
commit a52637e213fdda489f027d222f825ff43af4be74 (HEAD -> master)
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:48:20 2021 +0530

    change doc2

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ |
```

## 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.
- **$ git log --grep=" Commit message."**
- We can use the short form of commit message instead of a complete message.
- The above output is displaying all the commits that contain the word commit in its commit message.



```
krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$ git log --grep="commit"
commit ed5f5c55deffaa11d4c6a2a83229fe20b6dd3206
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 20:42:34 2021 +0530

    commit new doc

commit 63e0cdaba0abf548812bfd24ee15820f23373848
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Sat Feb 13 15:31:31 2021 +0530

    second commit

commit 99643a0376769921d20247433302daef617886f9
Author: krishna <krishnendhumalu04@gmail.com>
Date:   Fri Feb 12 22:55:46 2021 +0530

    first commit

krish@DESKTOP-SMCHI2R MINGW64 ~/desktop/test (master)
$
```

# Diff Commands

- **Diff command** is used in git to track the difference between the changes made on a file.
- Since Git is a version control system, tracking changes are something very vital to it.
- When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more.
- **Diff command** takes two inputs and reflects the differences between them.
- It is not necessary that these inputs are files only.

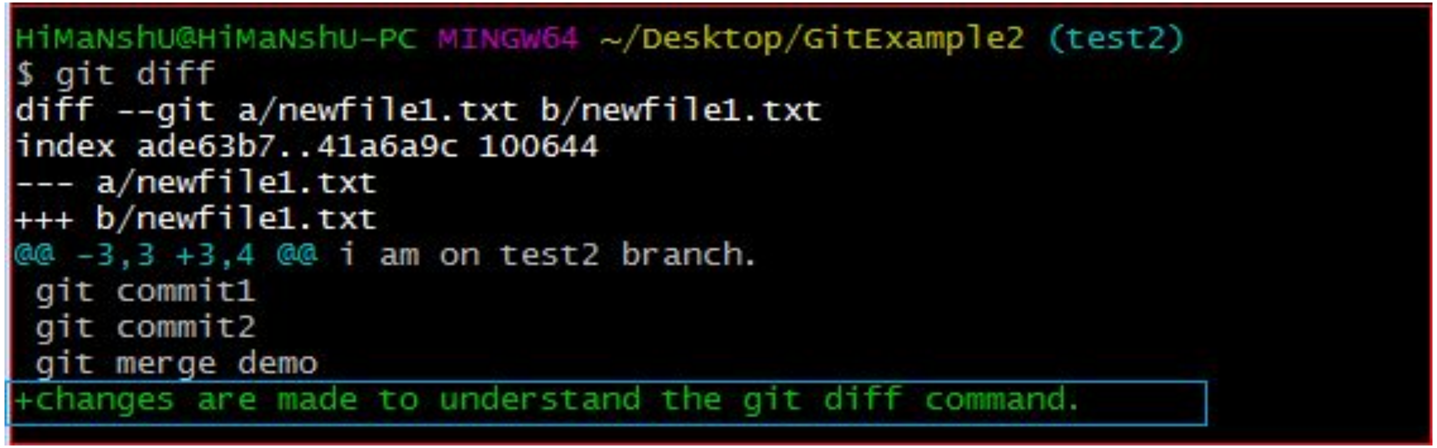
## Different scenarios where we can utilize the git Diff command

- Scenerio1: Track the changes that have not been staged.
- Scenerio2: Track the changes that have staged but not committed.
- Scenerio3: Track the changes after committing a file
- Scenario4: Track the changes between two commits
- To get the difference between branches

## Scenerio1: 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:



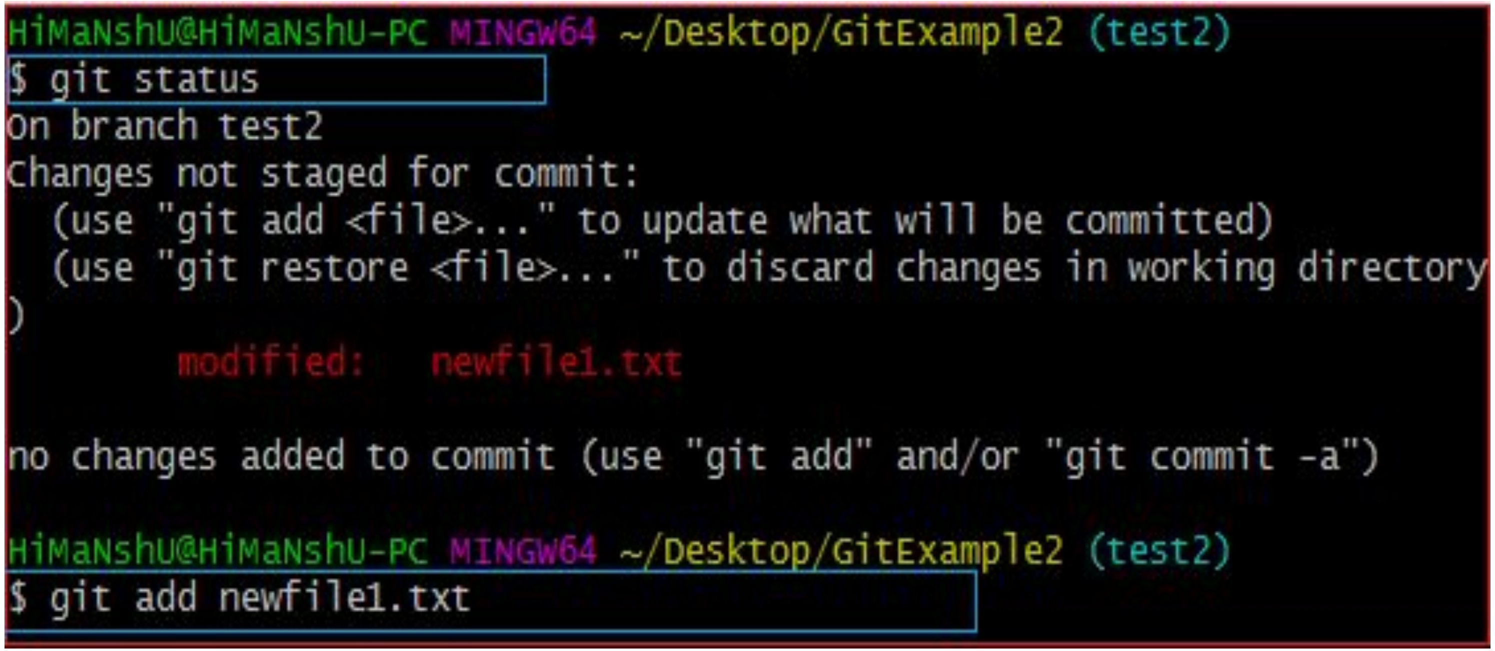
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git diff
diff --git a/newfile1.txt b/newfile1.txt
index ade63b7..41a6a9c 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,3 +3,4 @@ i am on test2 branch.
 git commit1
 git commit2
 git merge demo
+changes are made to understand the git diff command.
```

- 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

- 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:
    - **$ git status**
    - The above command will display the untracked file from the repository.
  - To add the file in the staging area, run the git add command as:
    - **$ git add < file name>**
    - The above command will add the file in the staging area.

## Consider the below output:



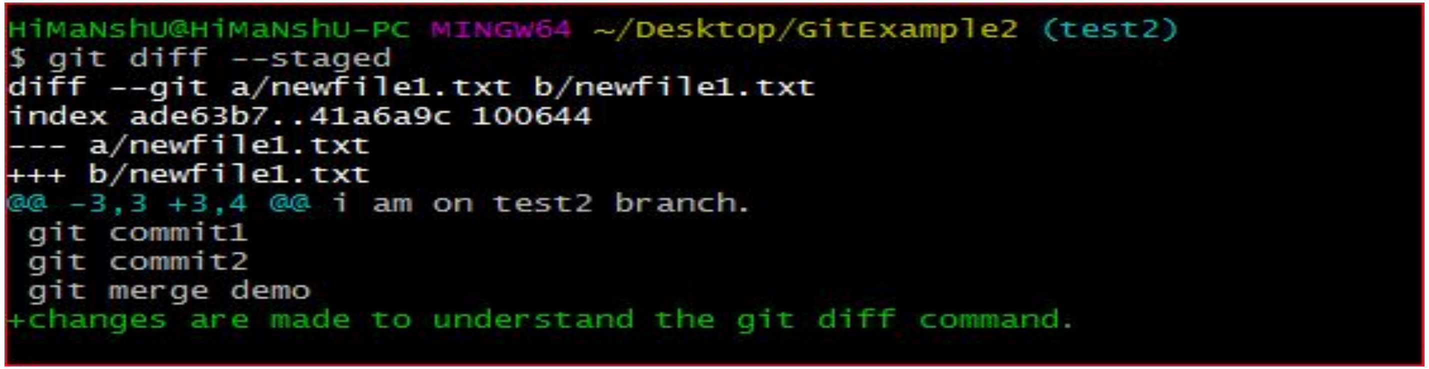
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git status
On branch test2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory
)
        modified:   newfile1.txt

no changes added to commit (use "git add" and/or "git commit -a")

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git add newfile1.txt
```

- 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:

```
$ git diff --staged
```

- The above command will display the changes of already staged files.

## Consider the below output:



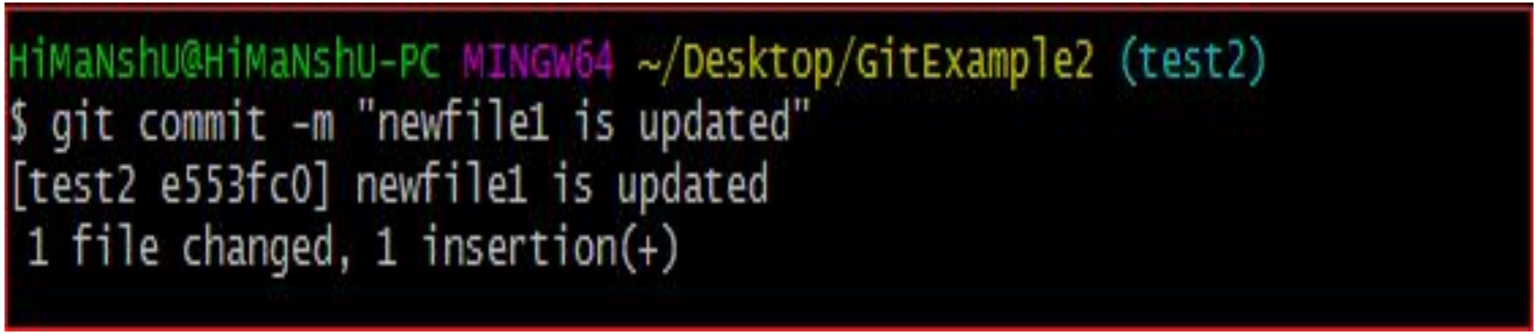
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git diff --staged
diff --git a/newfile1.txt b/newfile1.txt
index ade63b7..41a6a9c 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,3 +3,4 @@ i am on test2 branch.
 git commit1
 git commit2
 git merge demo
+changes are made to understand the git diff command.
```

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.

Consider the below output:

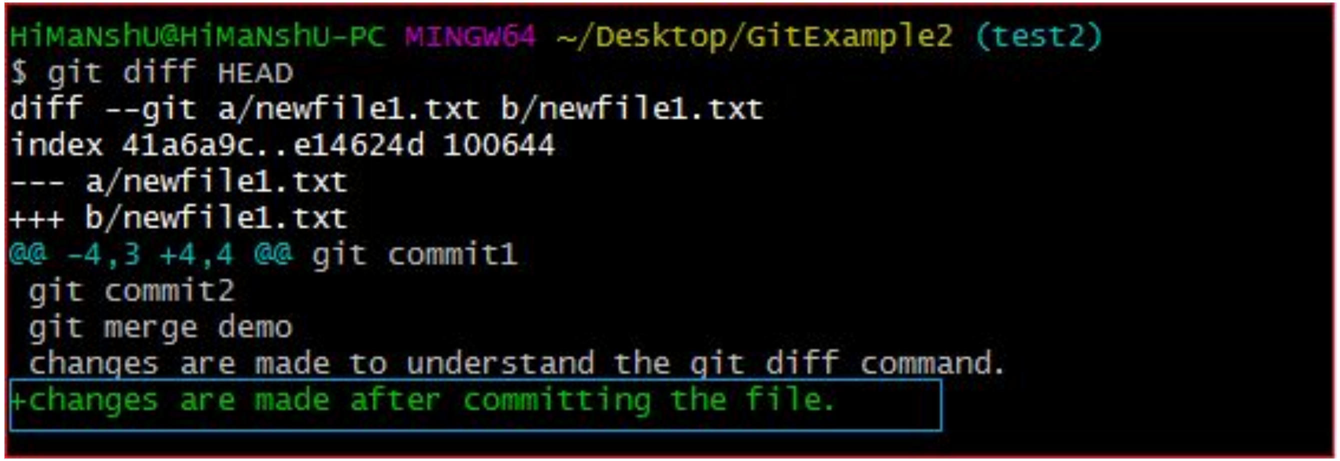


```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git commit -m "newfile1 is updated"
[test2 e553fc0] newfile1 is updated
 1 file changed, 1 insertion(+)
```

- **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:**

```
$ git diff HEAD
```

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



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git diff HEAD
diff --git a/newfile1.txt b/newfile1.txt
index 41a6a9c..e14624d 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -4,3 +4,4 @@ git commit1
 git commit2
 git merge demo
 changes are made to understand the git diff command.
+changes are made after committing the file.
```

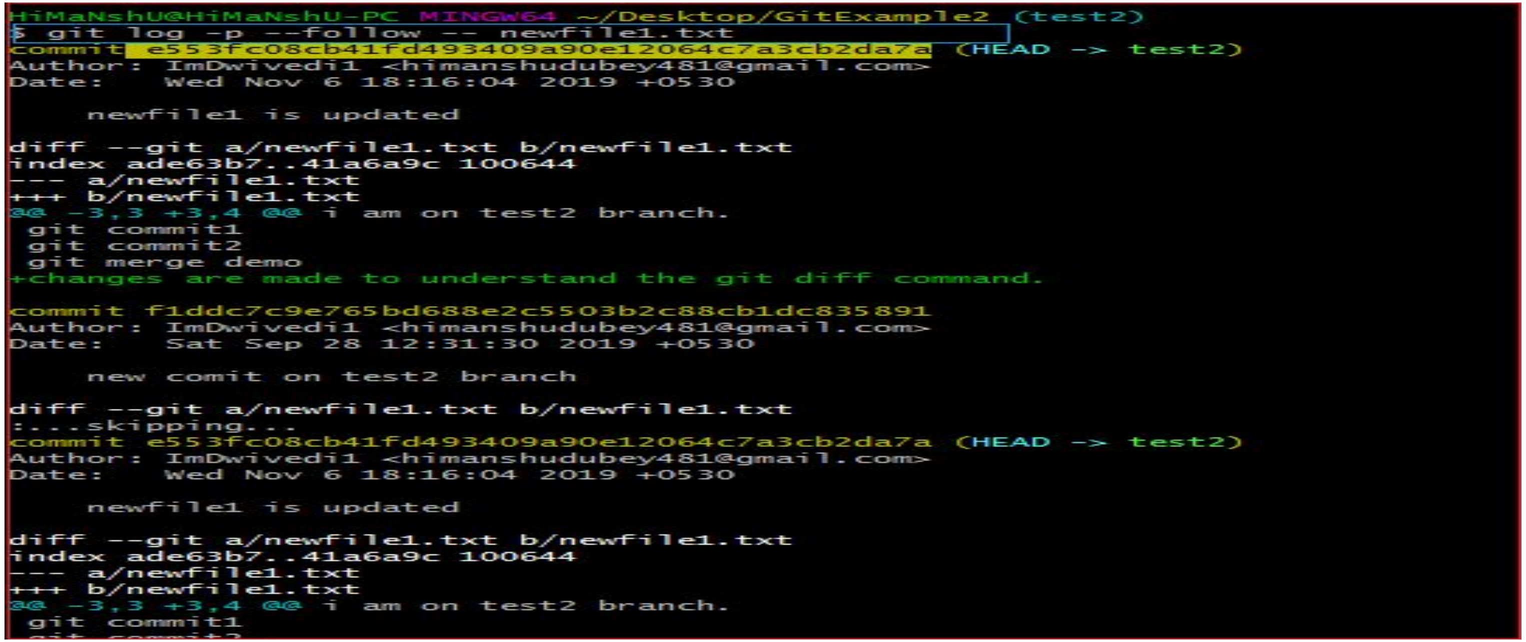
- The above command is displaying the updates of the file newfile1.txt on the highlighted section.

## 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: `$ 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:

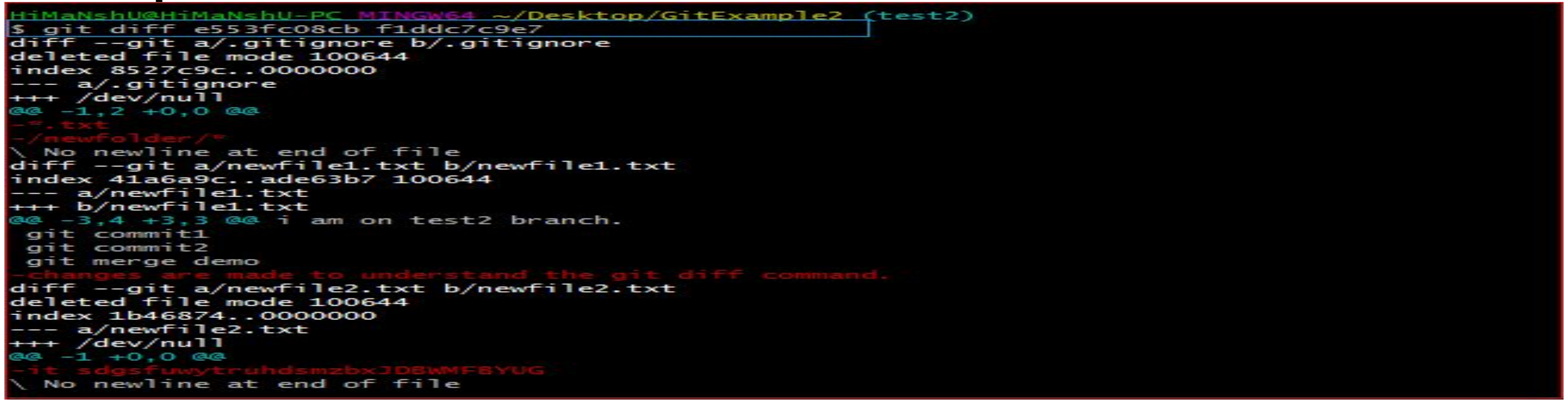
1. **`$ git log -p --follow -- filename`**

## Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git log -p --follow -- newfile1.txt
commit e553fc08cb41fd493409a90e12064c7a3cb2da7a (HEAD -> test2)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Nov 6 18:16:04 2019 +0530

    newfile1 is updated

diff --git a/newfile1.txt b/newfile1.txt
index ade63b7..41a6a9c 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,3 +3,4 @@ i am on test2 branch.
 git commit1
 git commit2
 git merge demo
+changes are made to understand the git diff command.

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

    new comit on test2 branch

diff --git a/newfile1.txt b/newfile1.txt
:...skipping...
commit e553fc08cb41fd493409a90e12064c7a3cb2da7a (HEAD -> test2)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Nov 6 18:16:04 2019 +0530

    newfile1 is updated

diff --git a/newfile1.txt b/newfile1.txt
index ade63b7..41a6a9c 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,3 +3,4 @@ i am on test2 branch.
 git commit1
```

- **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:$ git diff <commit1-sha> <commit2-sha> .The output is:**



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git diff e553fc08cb f1ddc7c9e7
diff --git a/.gitignore b/.gitignore
deleted file mode 100644
index 8527c9c..0000000
--- a/.gitignore
+++ /dev/null
@@ -1,2 +0,0 @@
-*.txt
-/newfolder/*
\ No newline at end of file
diff --git a/newfile1.txt b/newfile1.txt
index 41a6a9c..ade63b7 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,4 +3,3 @@ i am on test2 branch.
 git commit1
 git commit2
 git merge demo
-changes are made to understand the git diff command.
diff --git a/newfile2.txt b/newfile2.txt
deleted file mode 100644
index 1b46874..0000000
--- a/newfile2.txt
+++ /dev/null
@@ -1 +0,0 @@
-it sdgsfuwytruhdsmzbxJDBWMFBYUG
\ No newline at end of file
```

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

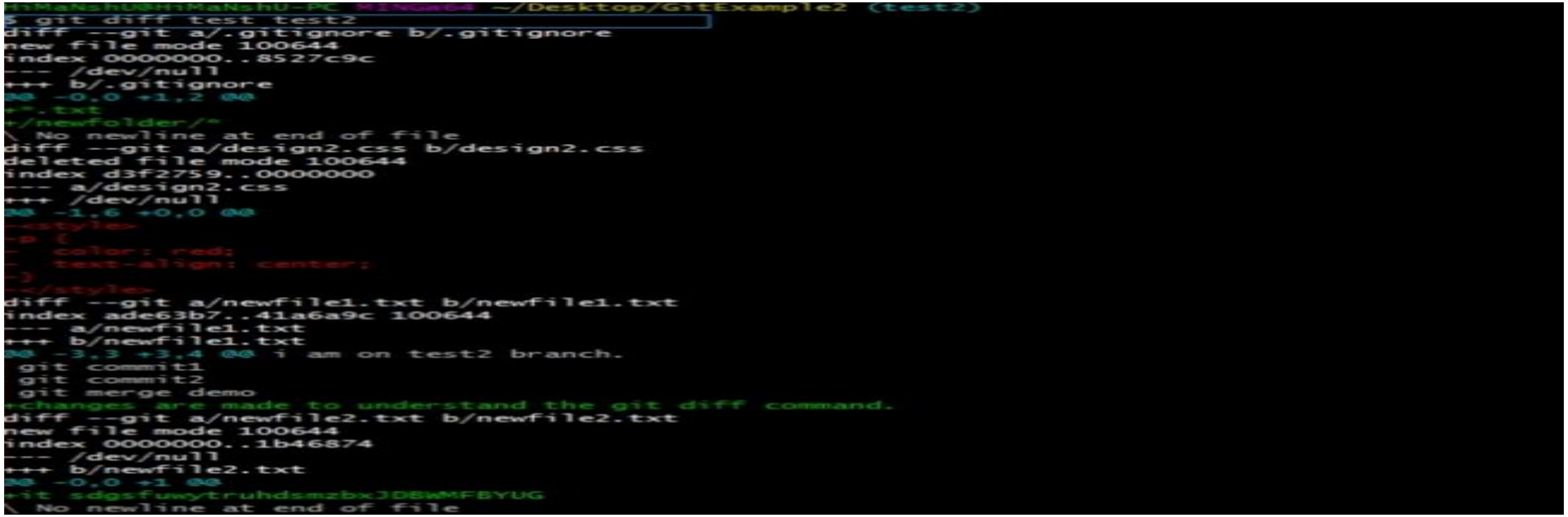
## Git Diff Branches

- Git allows comparing the branches
- Many conflicts can arise if you merge the branch without comparing it.
- To avoid these conflicts, Git allows many handy commands to preview, compare, and edit the changes.
- To avoid these conflicts, Git allows many handy commands to preview, compare, and edit 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:

```
$ 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:



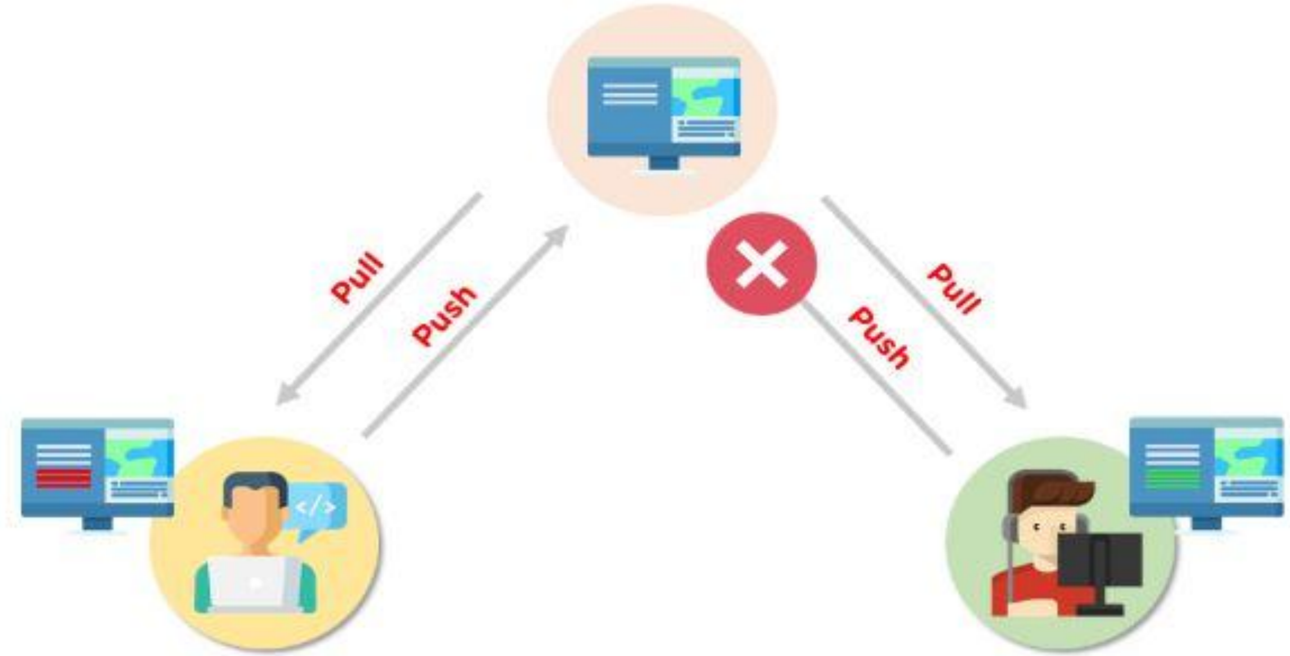
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git diff test test2
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..8527c9c
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,2 @@
+*.txt
+/newfolder/*
\ No newline at end of file
diff --git a/design2.css b/design2.css
deleted file mode 100644
index d3f2759..0000000
--- a/design2.css
+++ /dev/null
@@ -1,6 +0,0 @@
-<style>
-p {
-   color: red;
-   text-align: center;
-}
-</style>
diff --git a/newfile1.txt b/newfile1.txt
index ade63b7..41a6a9c 100644
--- a/newfile1.txt
+++ b/newfile1.txt
@@ -3,3 +3,4 @@ i am on test2 branch.
 git commit1
 git commit2
 git merge demo
+changes are made to understand the git diff command.
diff --git a/newfile2.txt b/newfile2.txt
new file mode 100644
index 0000000..1b46874
--- /dev/null
+++ b/newfile2.txt
@@ -0,0 +1 @@
+it sdgsfuwytruhdsmzbxJDBWMFBYUG
\ No newline at end of file
```

- The above output is displaying the differences between my repository  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

## Git Conflict Resolution

## What is git merge conflict?

- A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits.
- Git can merge the changes automatically only if the commits are on different lines or branches.


Pull
Push
Pull
Push

- Let’s assume there are two developers: Developer A and Developer B.
- Both of them pull the same code file from the remote repository and try to make various amendments in that file.
- After making the changes, Developer A pushes the file back to the remote repository from his local repository.
- Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repositoryTo prevent such conflicts, developers work in separate isolated branches.
- The Git merge command combines separate branches and resolves any conflicting edits.

:

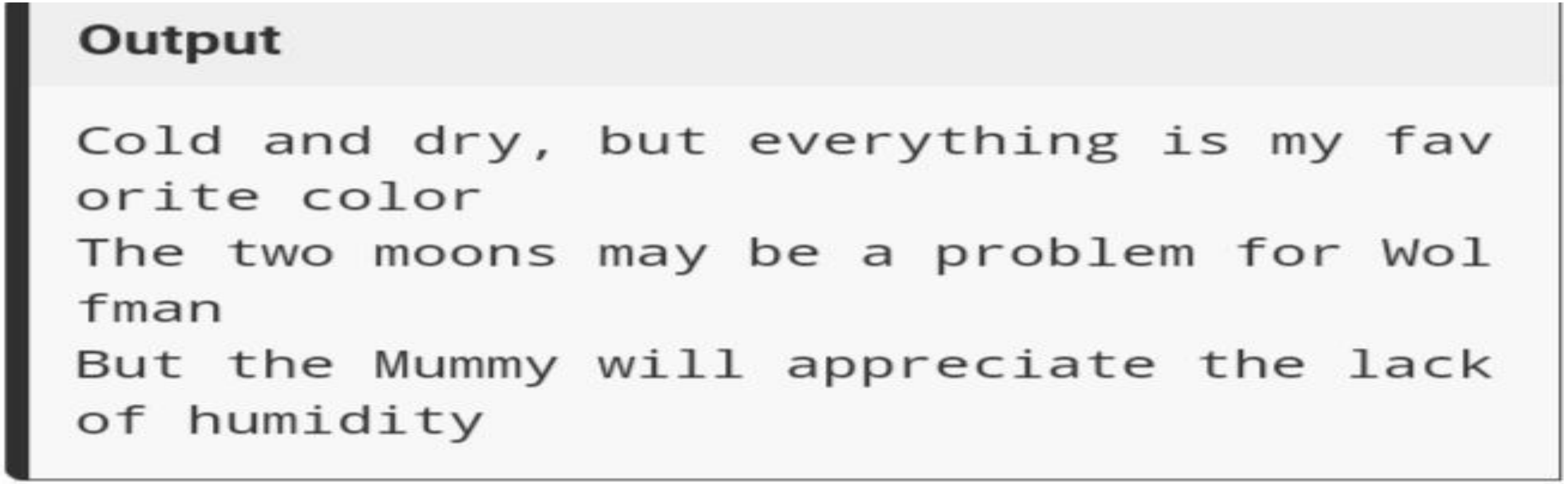
## How to resolve merge conflicts?

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

1. The easiest way to resolve a conflicted file is to open it and make any necessary changes
2. After editing the file, we can use the git add a command to stage the new merged content
3. The final step is to create a new commit with the help of the git commit command
4. Git will create a new merge commit to finalize the merge

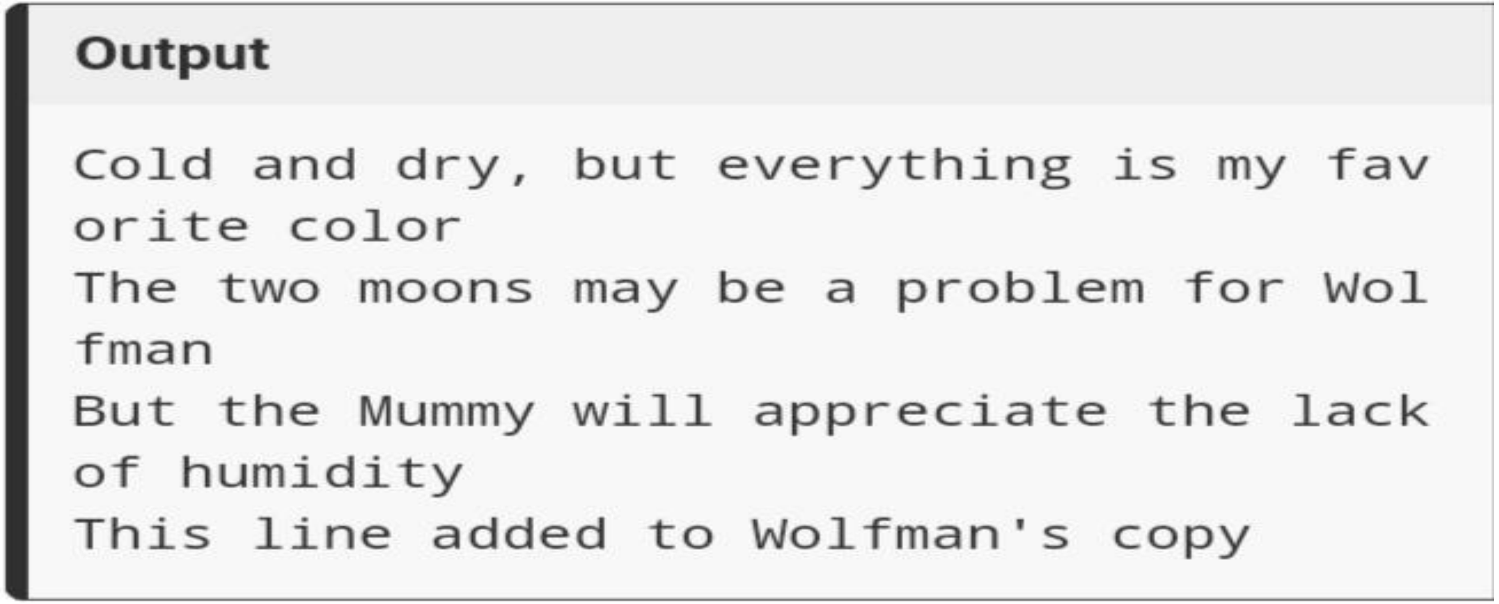
Let us now look into the Git commands that may play a significant role in resolving conflicts

- The file mars.txt currently looks like this in both partners' copies of our planets repository:


**Output**

```
Cold and dry, but everything is my fav
orite color
The two moons may be a problem for Wol
fman
But the Mummy will appreciate the lack
 of humidity
```

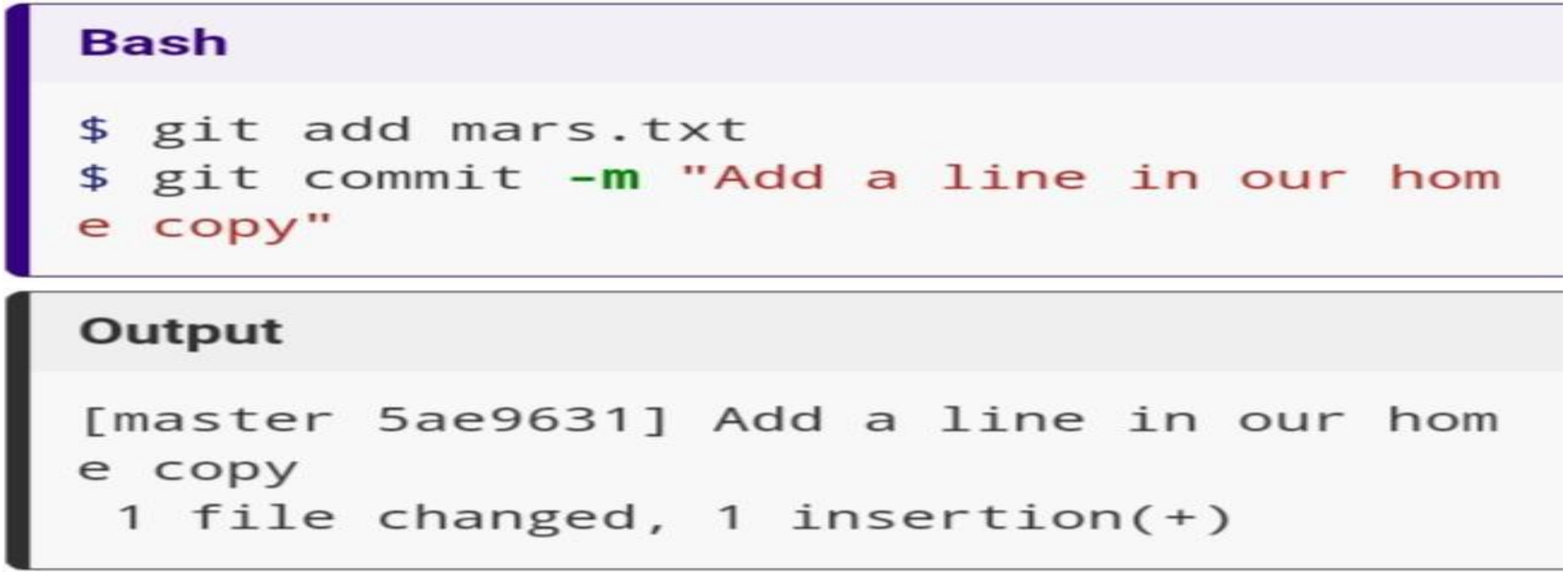
- Let’s add a line to the collaborator’s copy only:



**Output**

```
Cold and dry, but everything is my fav
orite color
The two moons may be a problem for Wol
fman
But the Mummy will appreciate the lack
of humidity
This line added to Wolfman's copy
```

- And then push the change to GitHub:

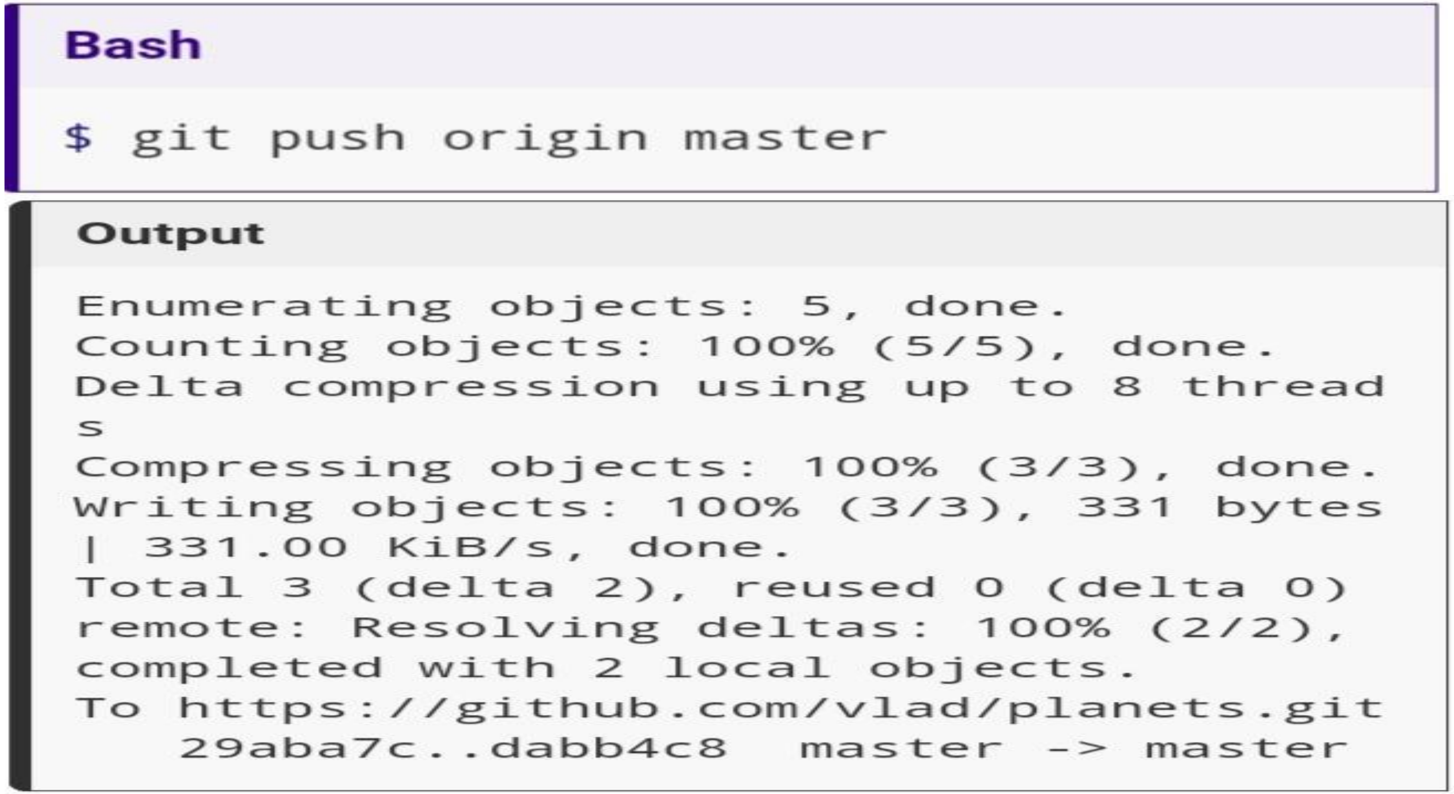


**Bash**

```
$ git add mars.txt
$ git commit -m "Add a line in our hom
e copy"
```

**Output**

```
[master 5ae9631] Add a line in our hom
e copy
 1 file changed, 1 insertion(+)
```



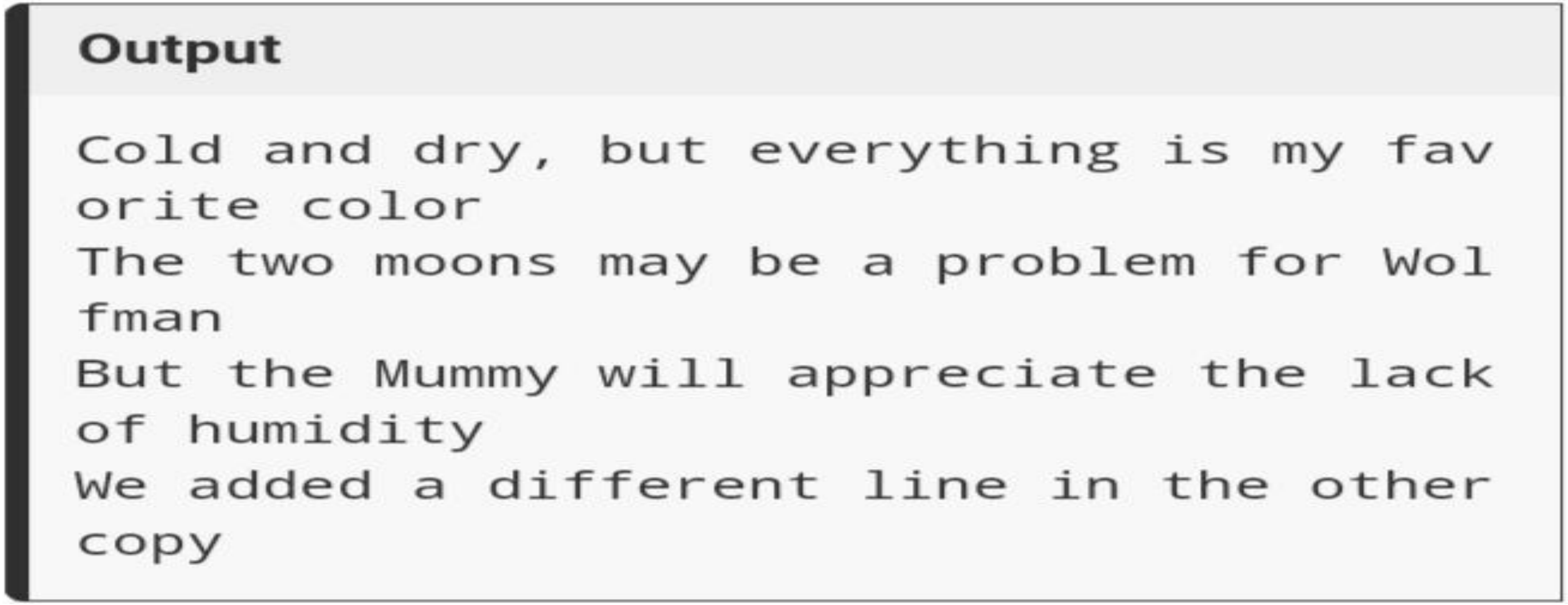
**Bash**

```
$ git push origin master
```

**Output**

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 thread
s
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 331 bytes
| 331.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2),
completed with 2 local objects.
To https://github.com/vlad/planets.git
   29aba7c..dabb4c8  master -> master
```

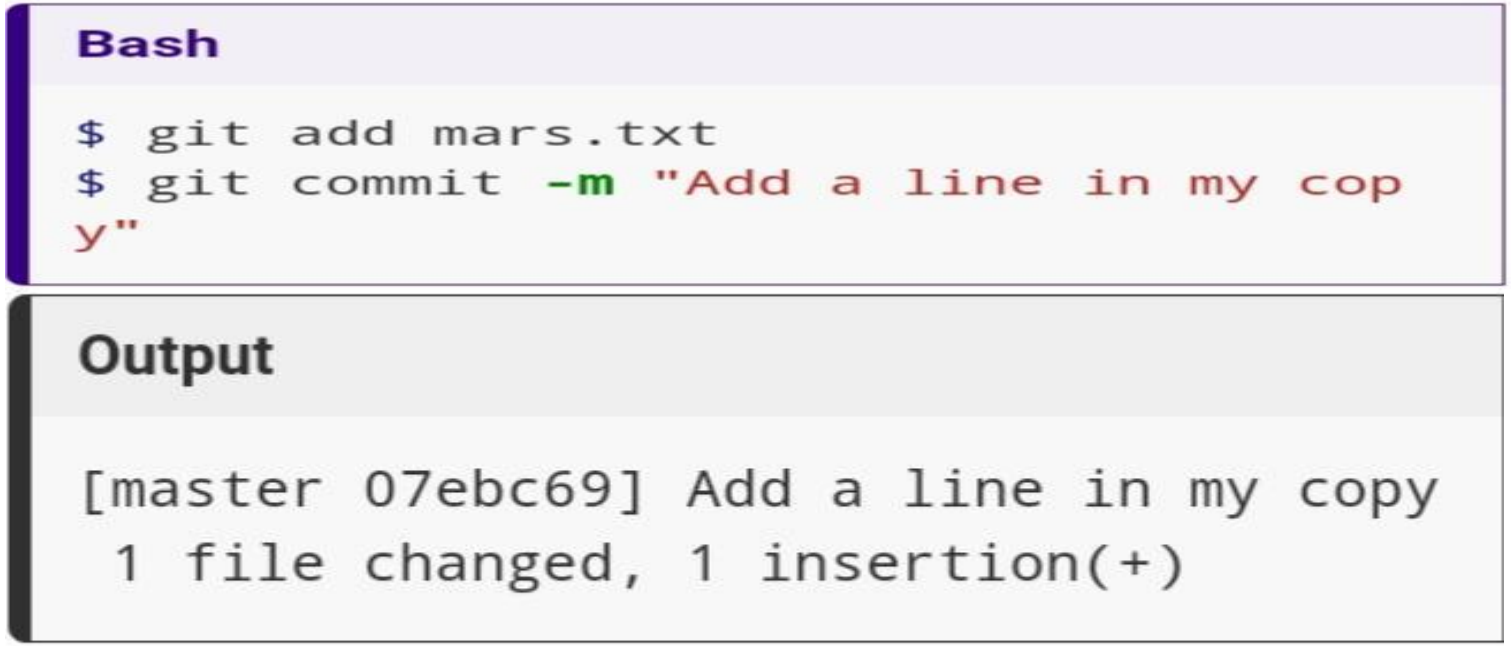
- Now let’s have the owner make a different change to their copy without updating from GitHub:



**Output**

```
Cold and dry, but everything is my fav
orite color
The two moons may be a problem for Wol
fman
But the Mummy will appreciate the lack
 of humidity
We added a different line in the other
 copy
```

- We can commit the change locally:



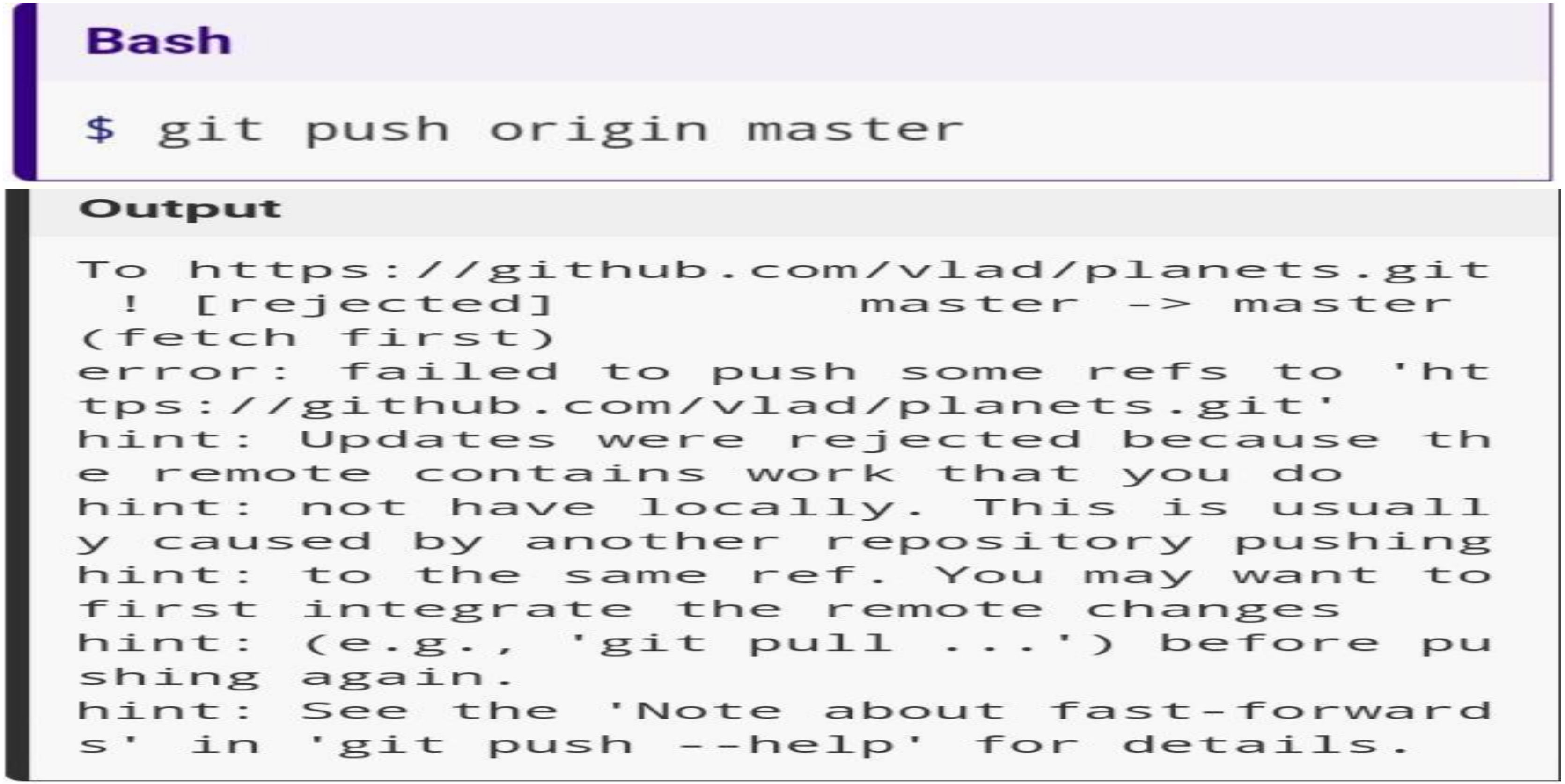
Bash

```
$ git add mars.txt
$ git commit -m "Add a line in my cop
y"
```

Output

```
[master 07ebc69] Add a line in my copy
 1 file changed, 1 insertion(+)
```

- But Git won't let us push it to GitHub:



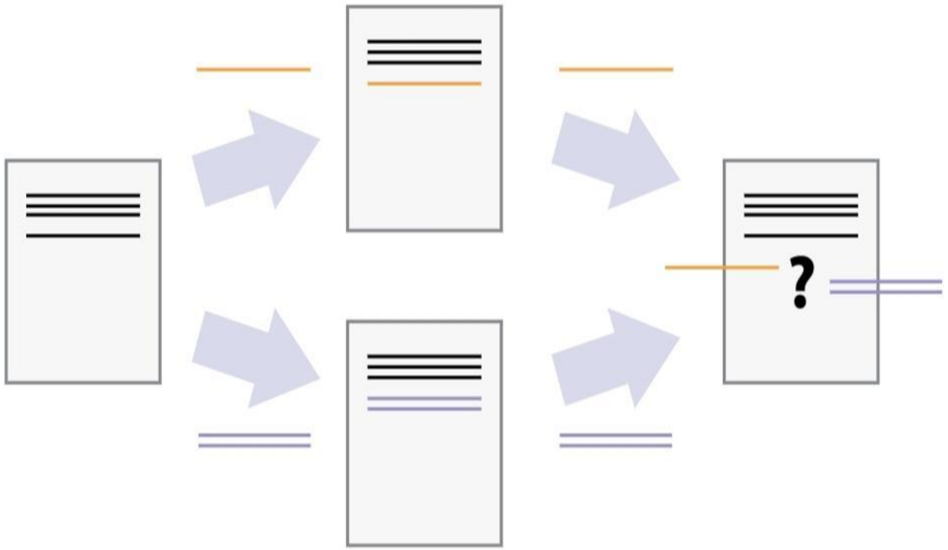
Bash

```
$ git push origin master
```

Output

```
To https://github.com/vlad/planets.git
 ! [rejected]        master -> master
(fetch first)
error: failed to push some refs to 'ht
tps://github.com/vlad/planets.git'
hint: Updates were rejected because th
e remote contains work that you do
hint: not have locally. This is usuall
y caused by another repository pushing
hint: to the same ref. You may want to
first integrate the remote changes
hint: (e.g., 'git pull ...') before pu
shing again.
hint: See the 'Note about fast-forward
s' in 'git push --help' for details.
```

- This is known as Conflict



- Git rejects the push because it detects that the remote repository has new updates that have not been incorporated into the local branch.

- **What we have to do is pull the changes from GitHub, merge them into the copy we’re currently working in, and then push that.**

- Let’s start by pulling:

**Bash**

```
$ git pull origin master
```

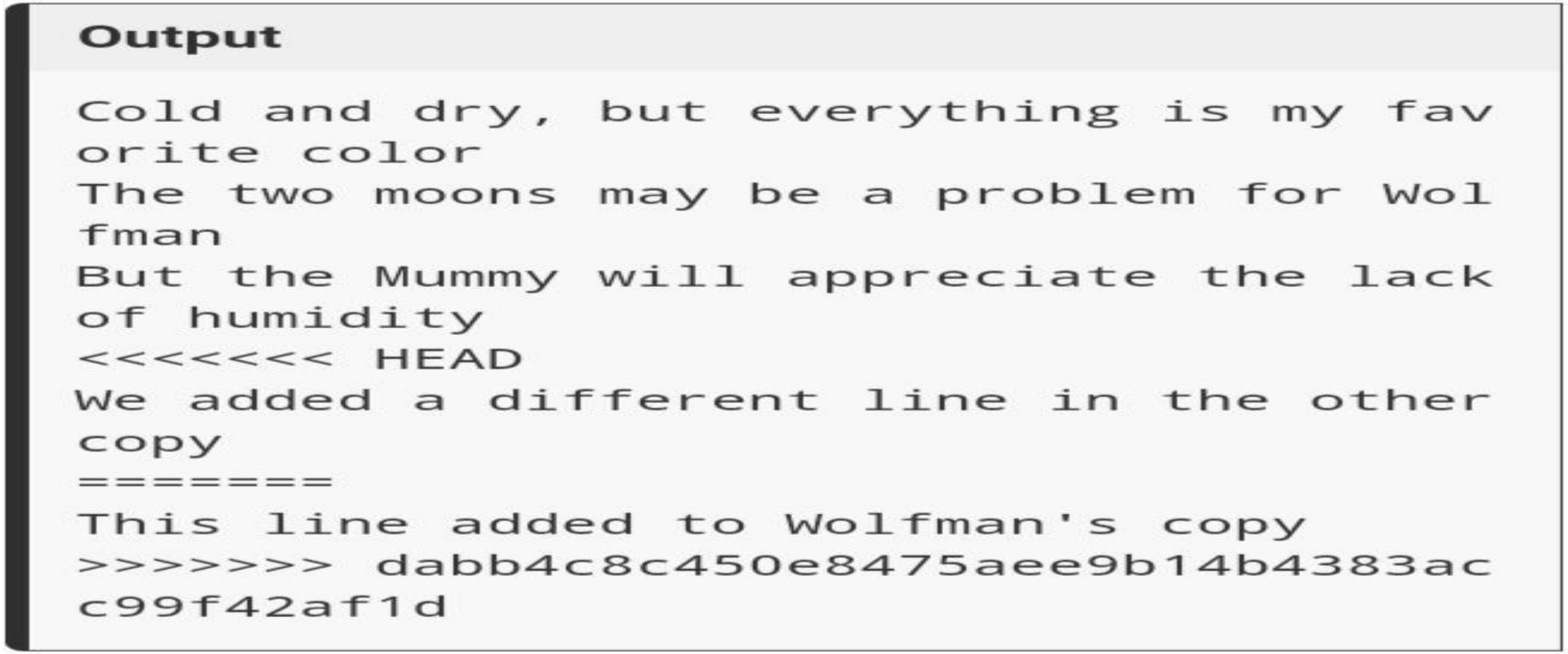
**Output**

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5),
done.
remote: Compressing objects: 100% (1/
1), done.
remote: Total 3 (delta 2), reused 3 (d
elta 2), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/vlad/planets
 * branch            master     -> FET
CH_HEAD
    29aba7c..dabb4c8  master     -> or
igin/master
Auto-merging mars.txt
CONFLICT (content): Merge conflict in
mars.txt
Automatic merge failed; fix conflicts
and then commit the result.
```

- The git pull command updates the local repository to include those changes already included in the remote repository.

- After the changes from remote branch have been fetched, Git detects that changes made to the local copy overlap with those made to the remote repository, and therefore refuses to merge the two versions to stop us from trampling on our previous work.

- The conflict is marked in in the affected file:



**Output**

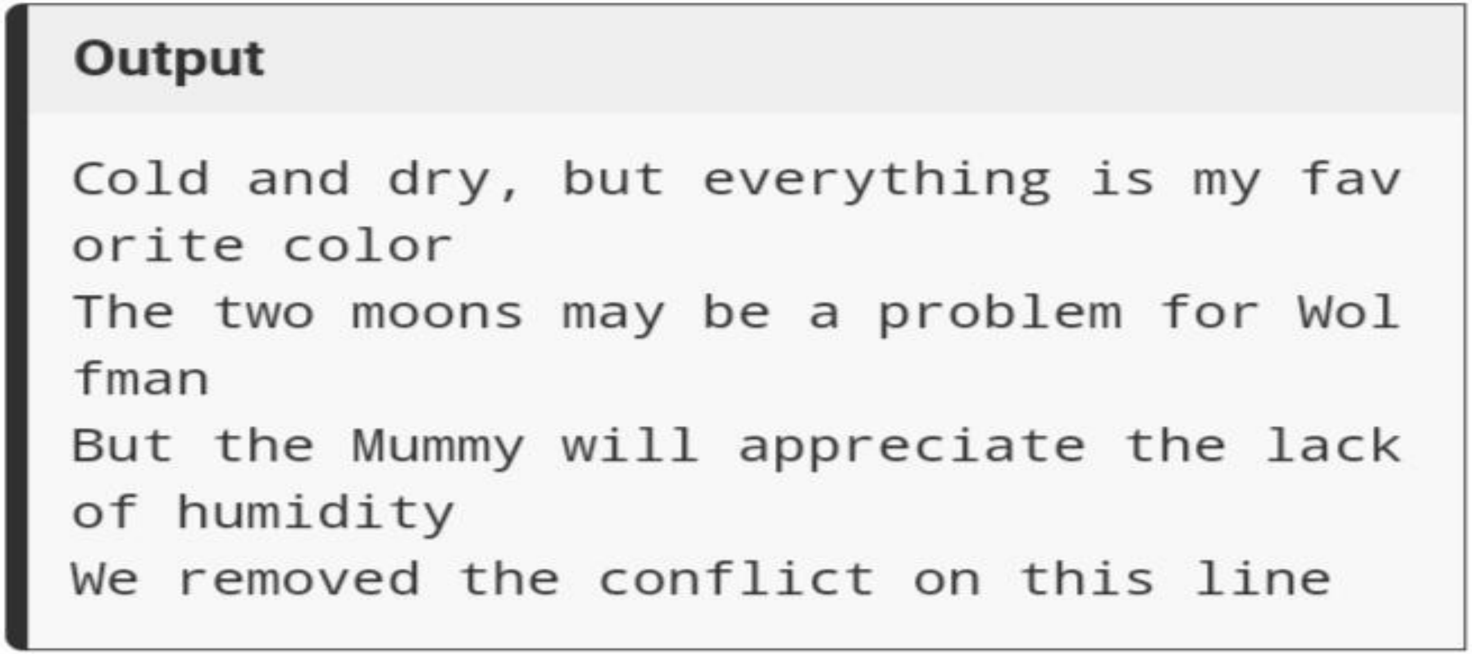
```
Cold and dry, but everything is my fav
orite color
The two moons may be a problem for Wol
fman
But the Mummy will appreciate the lack
of humidity
<<<<<<< HEAD
We added a different line in the other
copy
=======
This line added to Wolfman's copy
>>>>>>> dabb4c8c450e8475aee9b14b4383ac
c99f42af1d
```

- Our change is preceded by <<<<<<< HEAD.
- Git has then inserted ======= as a separator between the conflicting changes and marked the end of the content downloaded from GitHub with >>>>>>>.
- (The string of letters and digits after that marker identifies the commit we've just downloaded.)

- It is now up to us to edit this file to remove these markers and reconcile the changes.

- We can do anything we want:
    - keep the change made in the local repository
    - keep the change made in the remote repository
    - write something new to replace both
    - get rid of the change entirely.

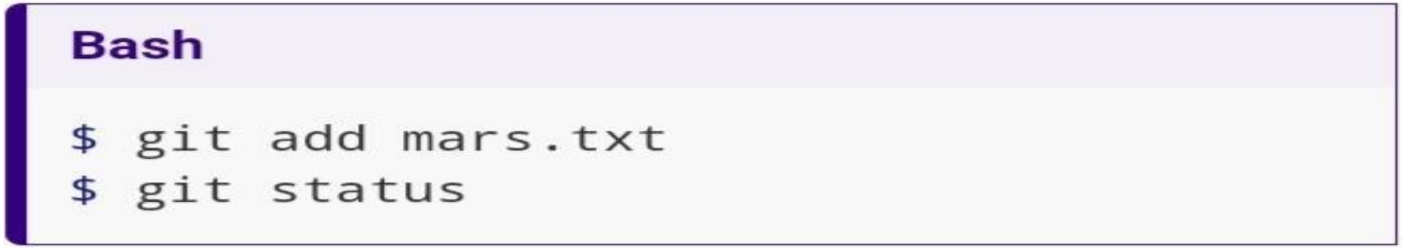
- Let’s replace both so that the file looks like this:



**Output**

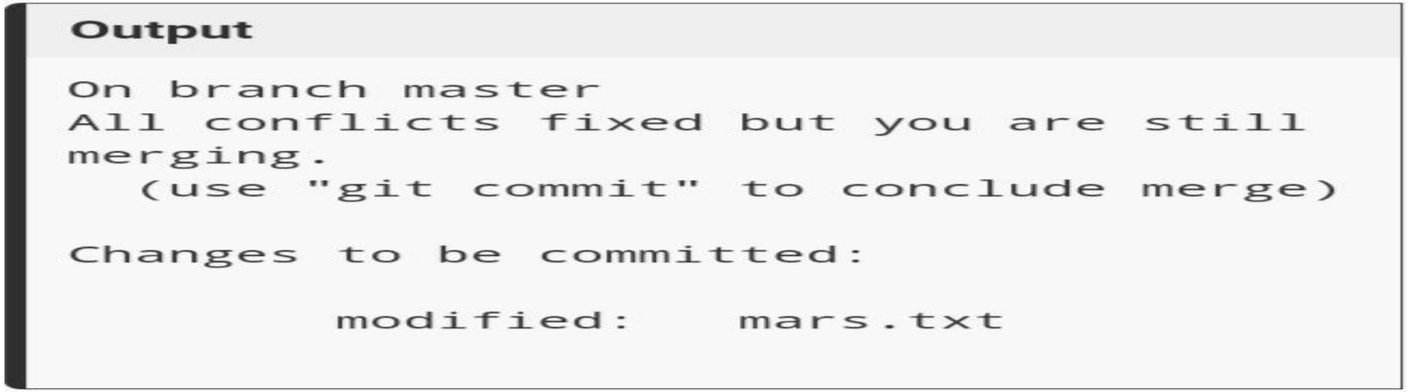
```
Cold and dry, but everything is my fav
orite color
The two moons may be a problem for Wol
fman
But the Mummy will appreciate the lack
 of humidity
We removed the conflict on this line
```

- To finish merging, we add mars.txt to the changes being made by the merge and then commit:



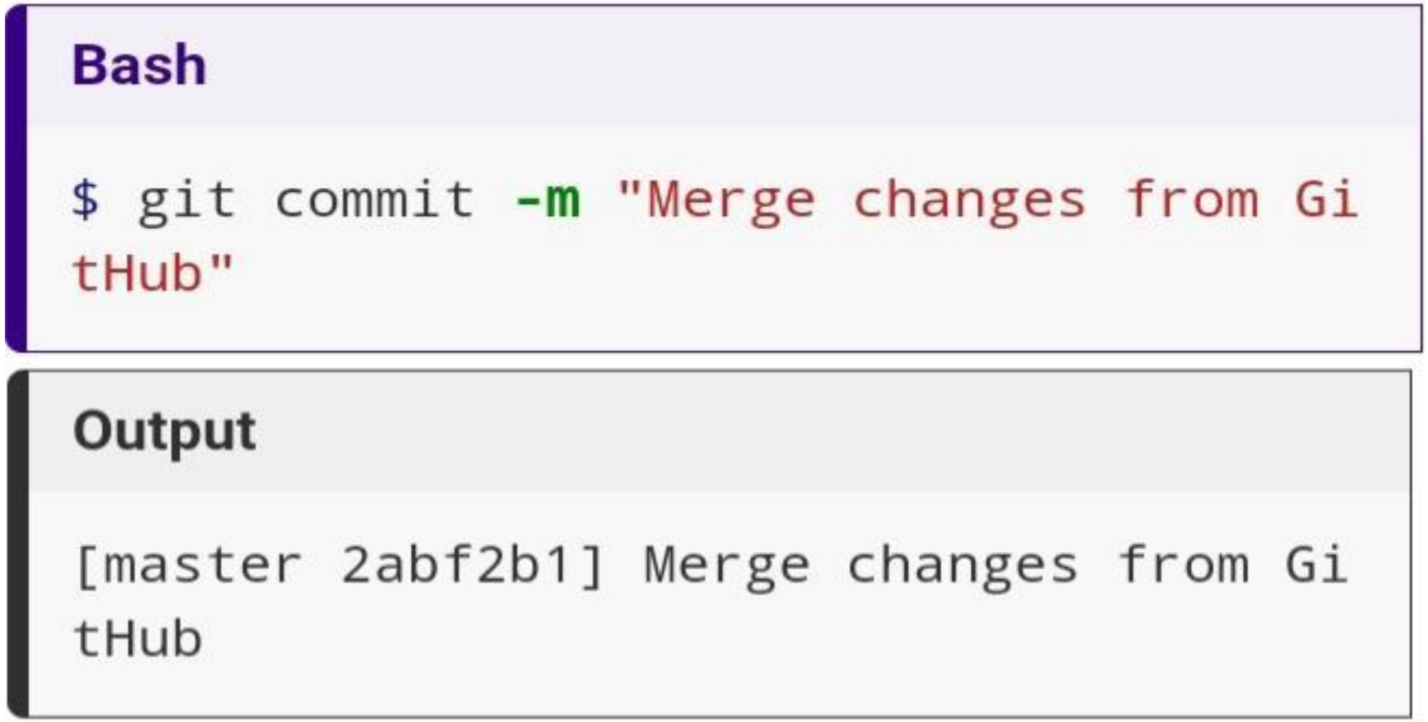
Bash

```
$ git add mars.txt
$ git status
```



Output

```
On branch master
All conflicts fixed but you are still
merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   mars.txt
```



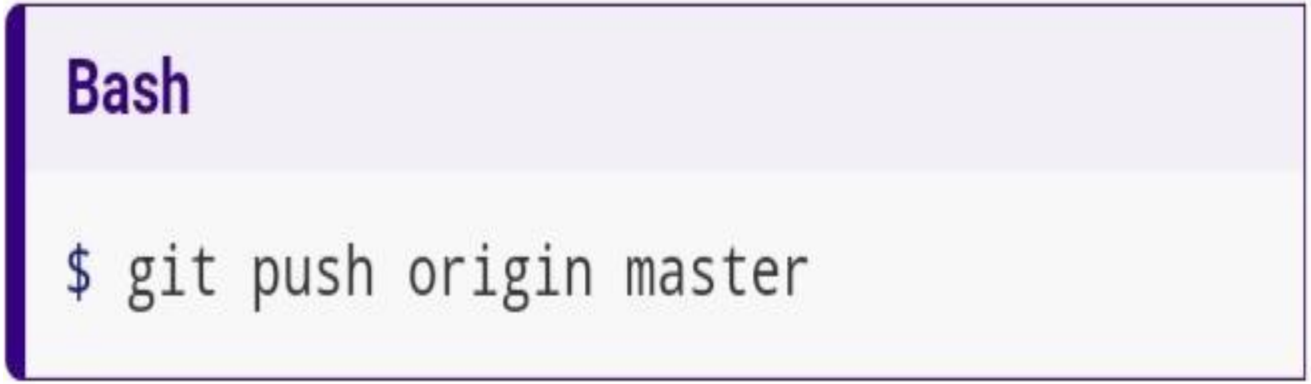
Bash

```
$ git commit -m "Merge changes from Gi
tHub"
```

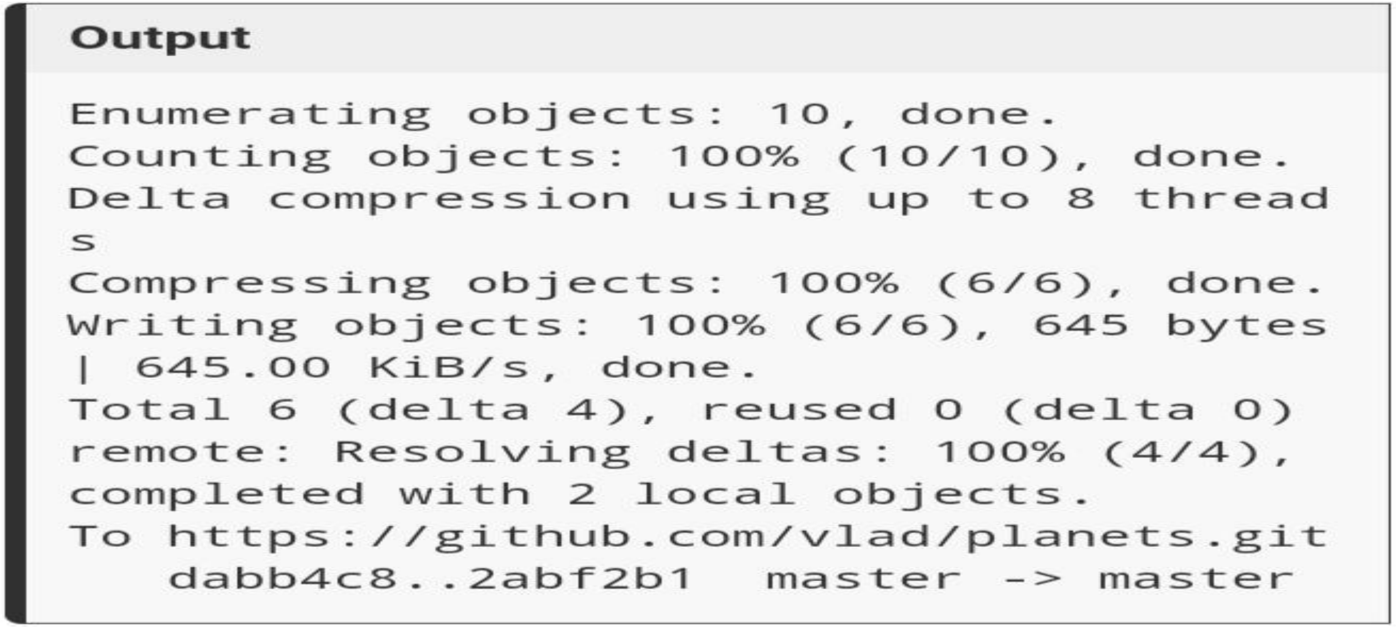
Output

```
[master 2abf2b1] Merge changes from Gi
tHub
```

- Now we can push our changes to GitHub:



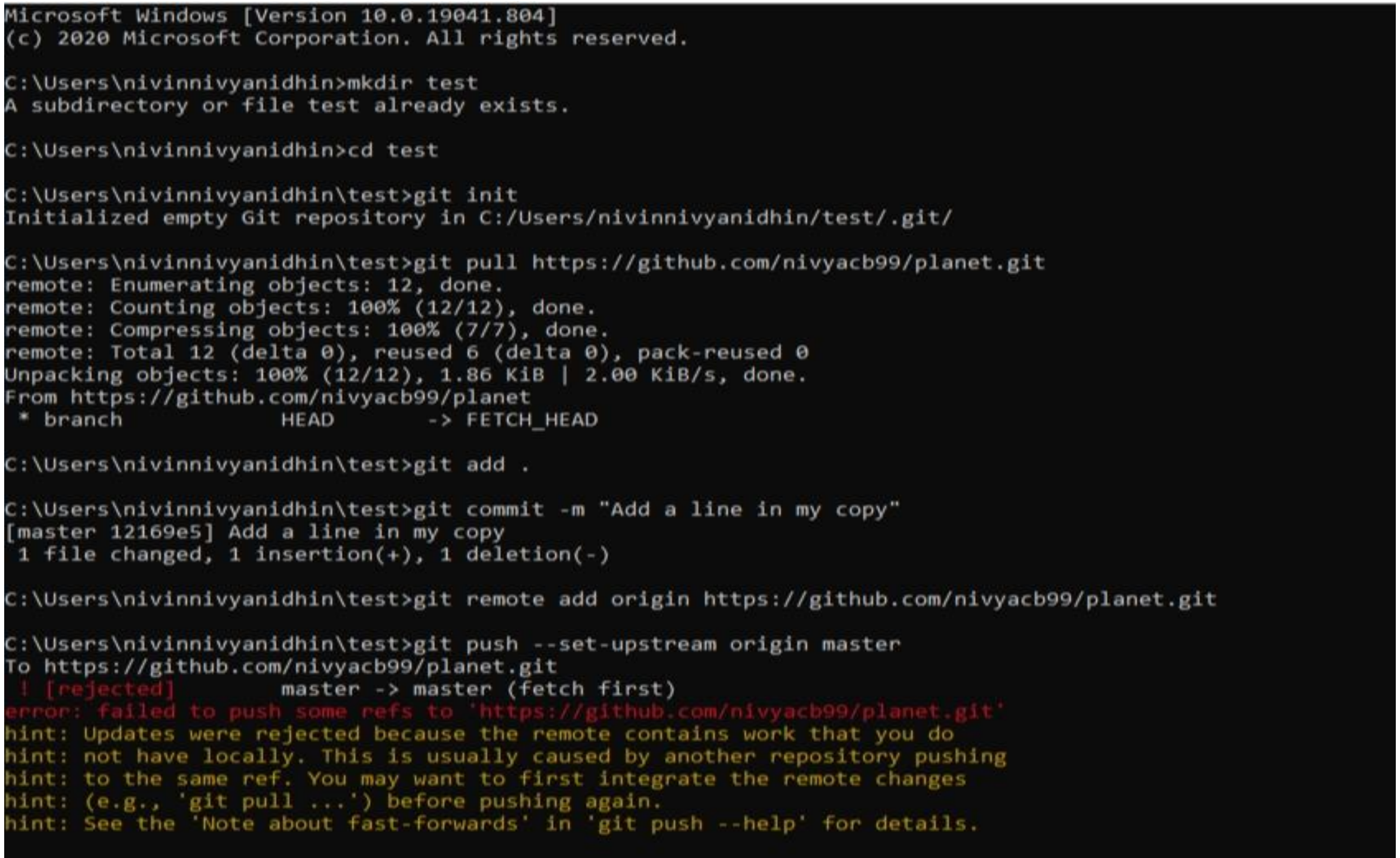
```
$ git push origin master
```



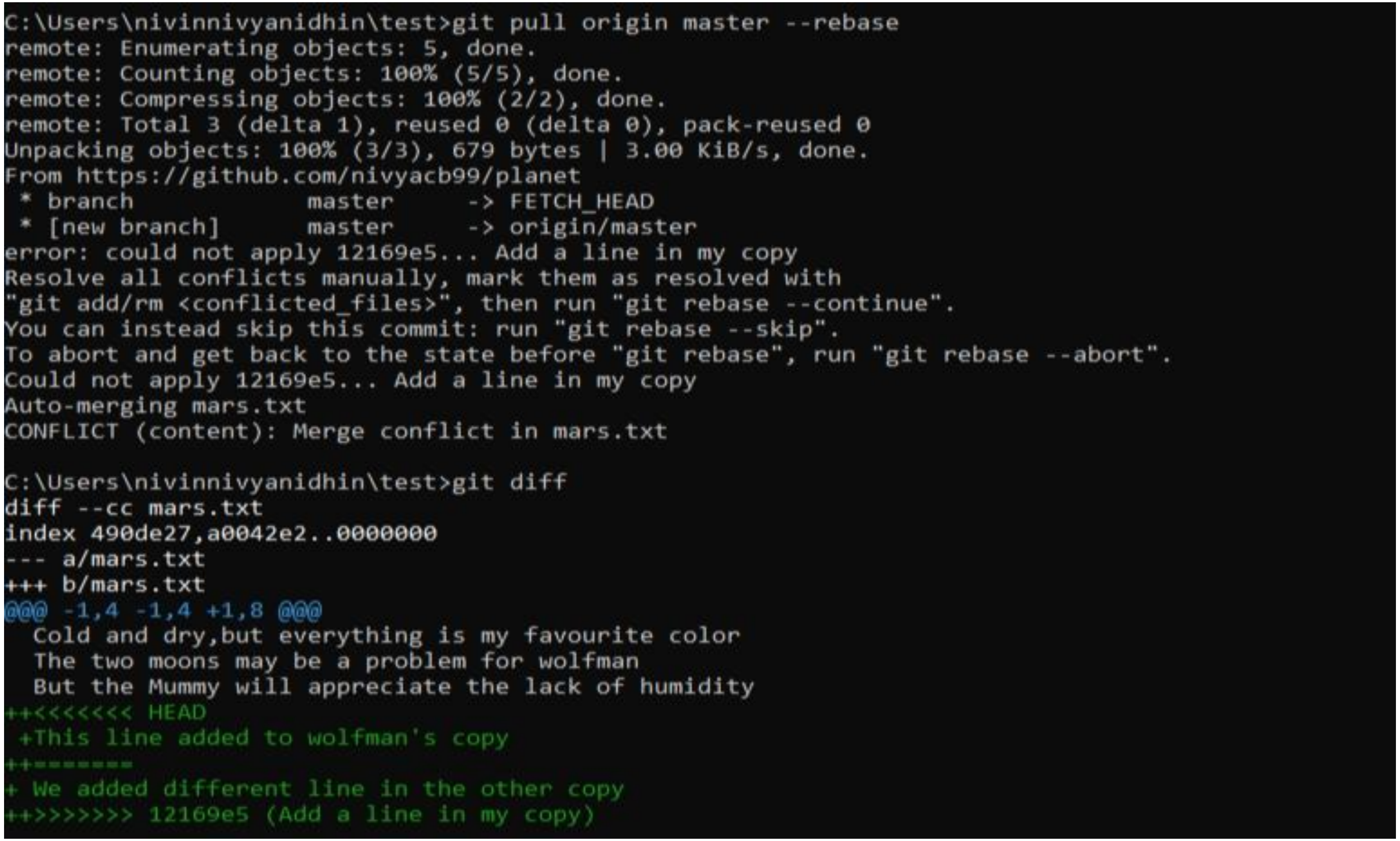
**Output**

```
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 thread
s
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 645 bytes
| 645.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4),
completed with 2 local objects.
To https://github.com/vlad/planets.git
   dabb4c8..2abf2b1  master -> master
```

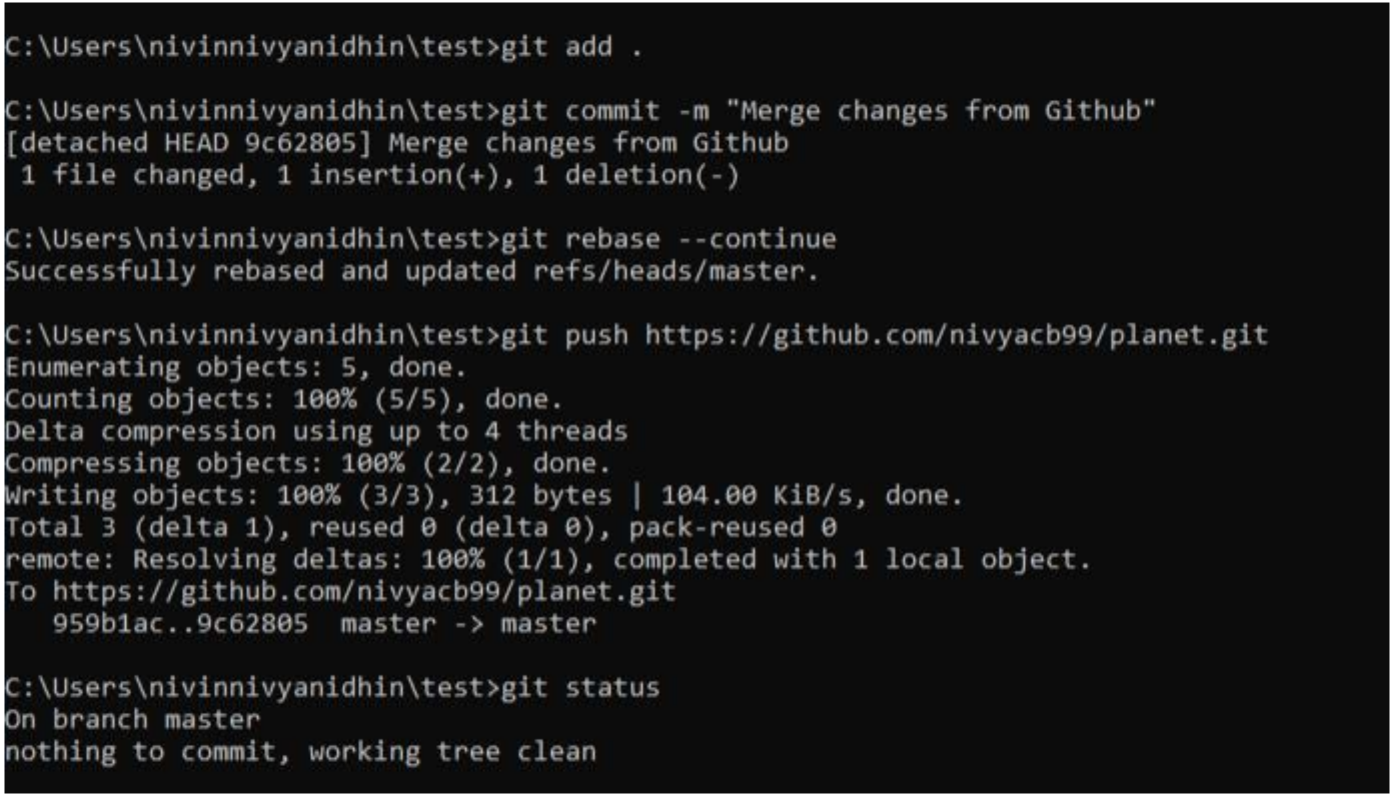
- Now the conflict is resolved



```
Microsoft Windows [Version 10.0.19041.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\nivinnivyanidhin>mkdir test
A subdirectory or file test already exists.

C:\Users\nivinnivyanidhin>cd test

C:\Users\nivinnivyanidhin\test>git init
Initialized empty Git repository in C:/Users/nivinnivyanidhin/test/.git/

C:\Users\nivinnivyanidhin\test>git pull https://github.com/nivyacb99/planet.git
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 12 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (12/12), 1.86 KiB | 2.00 KiB/s, done.
From https://github.com/nivyacb99/planet
 * branch            HEAD       -> FETCH_HEAD

C:\Users\nivinnivyanidhin\test>git add .

C:\Users\nivinnivyanidhin\test>git commit -m "Add a line in my copy"
[master 12169e5] Add a line in my copy
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\Users\nivinnivyanidhin\test>git remote add origin https://github.com/nivyacb99/planet.git

C:\Users\nivinnivyanidhin\test>git push --set-upstream origin master
To https://github.com/nivyacb99/planet.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/nivyacb99/planet.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```



```
C:\Users\nivinnivyanidhin\test>git pull origin master --rebase
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 679 bytes | 3.00 KiB/s, done.
From https://github.com/nivyacb99/planet
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
error: could not apply 12169e5... Add a line in my copy
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 12169e5... Add a line in my copy
Auto-merging mars.txt
CONFLICT (content): Merge conflict in mars.txt

C:\Users\nivinnivyanidhin\test>git diff
diff --cc mars.txt
index 490de27,a0042e2..0000000
--- a/mars.txt
+++ b/mars.txt
@@@ -1,4 -1,4 +1,8 @@@
  Cold and dry,but everything is my favourite color
  The two moons may be a problem for wolfman
  But the Mummy will appreciate the lack of humidity
++<<<<<<< HEAD
 +This line added to wolfman's copy
++=======
+ We added different line in the other copy
++>>>>>>> 12169e5 (Add a line in my copy)
```



```
C:\Users\nivinnivyanidhin\test>git add .

C:\Users\nivinnivyanidhin\test>git commit -m "Merge changes from Github"
[detached HEAD 9c62805] Merge changes from Github
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\Users\nivinnivyanidhin\test>git rebase --continue
Successfully rebased and updated refs/heads/master.

C:\Users\nivinnivyanidhin\test>git push https://github.com/nivyacb99/planet.git
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 312 bytes | 104.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/nivyacb99/planet.git
   959b1ac..9c62805  master -> master

C:\Users\nivinnivyanidhin\test>git status
On branch master
nothing to commit, working tree clean
```

# THANK YOU