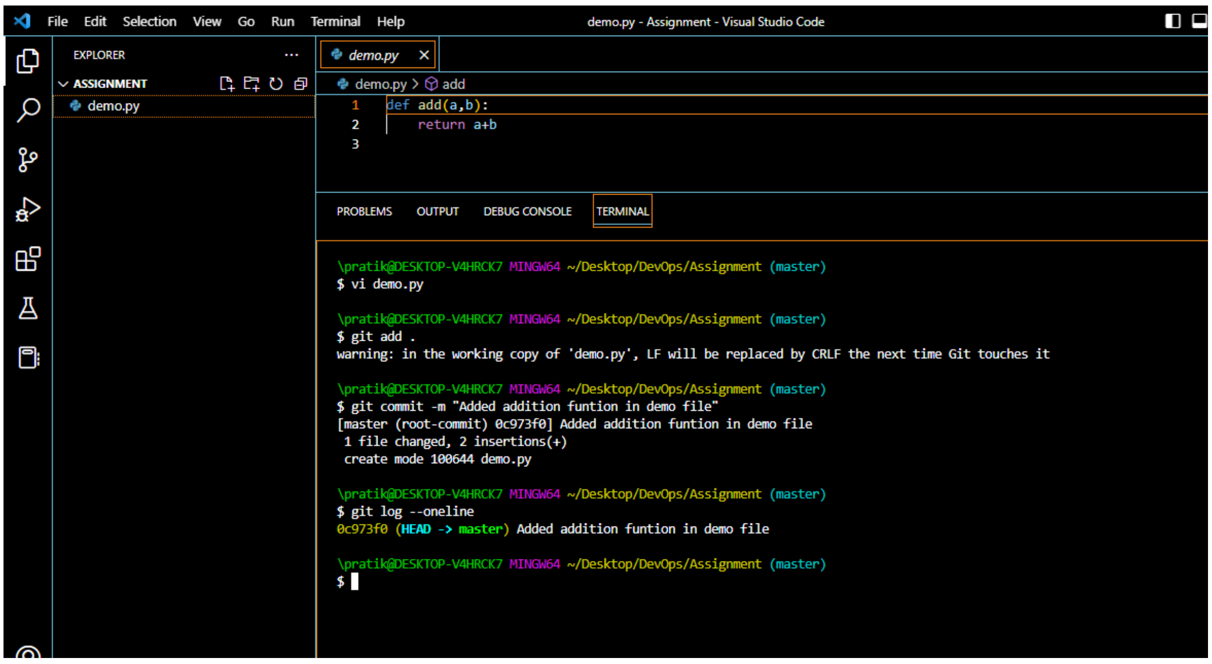
## Q1. Describe the usage of the git stash command by using an example and also state the process by giving the screenshot of all the commands written in git bash.

When working on a project, there may be situations in which we think of a feature to be added, but we don't want to commit it to our project right away. If we are unsure about that addition, we can use git stash to store it.

The git stash command stores modifications that we make to a file, and reverts the file back to the HEAD commit. These modifications are stored in a stack structure.

Use command "git stash" after making a modification. It creates a stash.

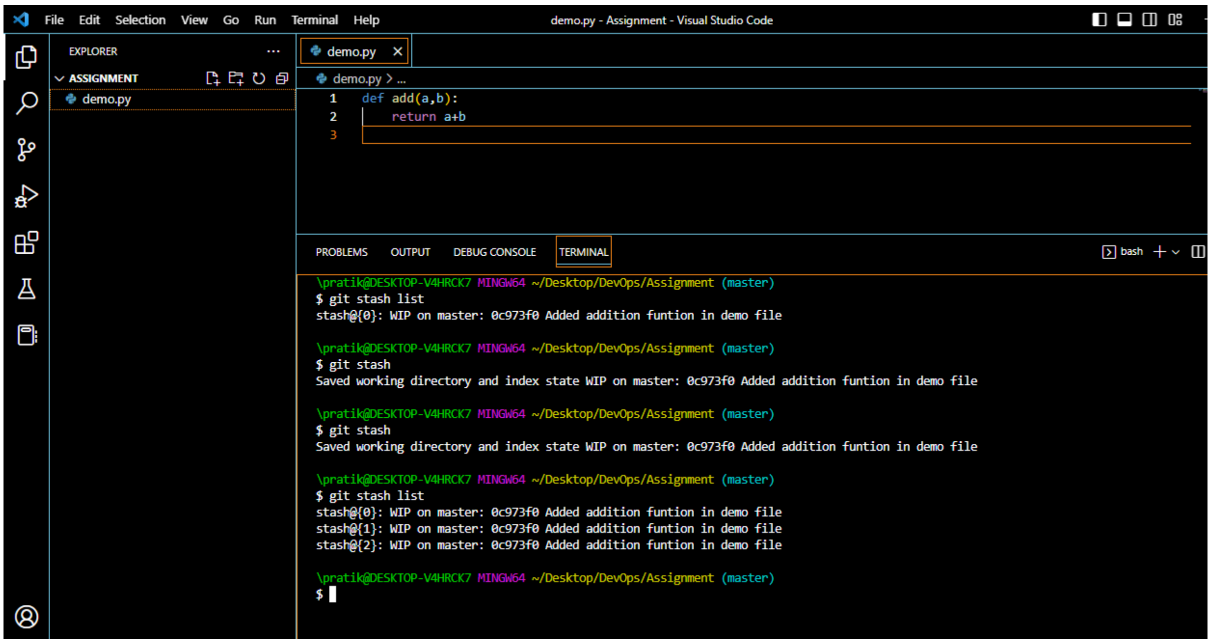
First we make a commit after adding an addition function to demo file.


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py
demo.py > add
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
def add(a,b):
    return a+b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ vi demo.py

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git add .
warning: in the working copy of 'demo.py', LF will be replaced by CRLF the next time Git touches it

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git commit -m "Added addition funtion in demo file"
[master (root-commit) 0c973f0] Added addition funtion in demo file
 1 file changed, 2 insertions(+)
 create mode 100644 demo.py

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git log --oneline
0c973f0 (HEAD -> master) Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```

After this, we add a subtraction function and stash it. We also create multiplication and exponent functions and stash them respectively.

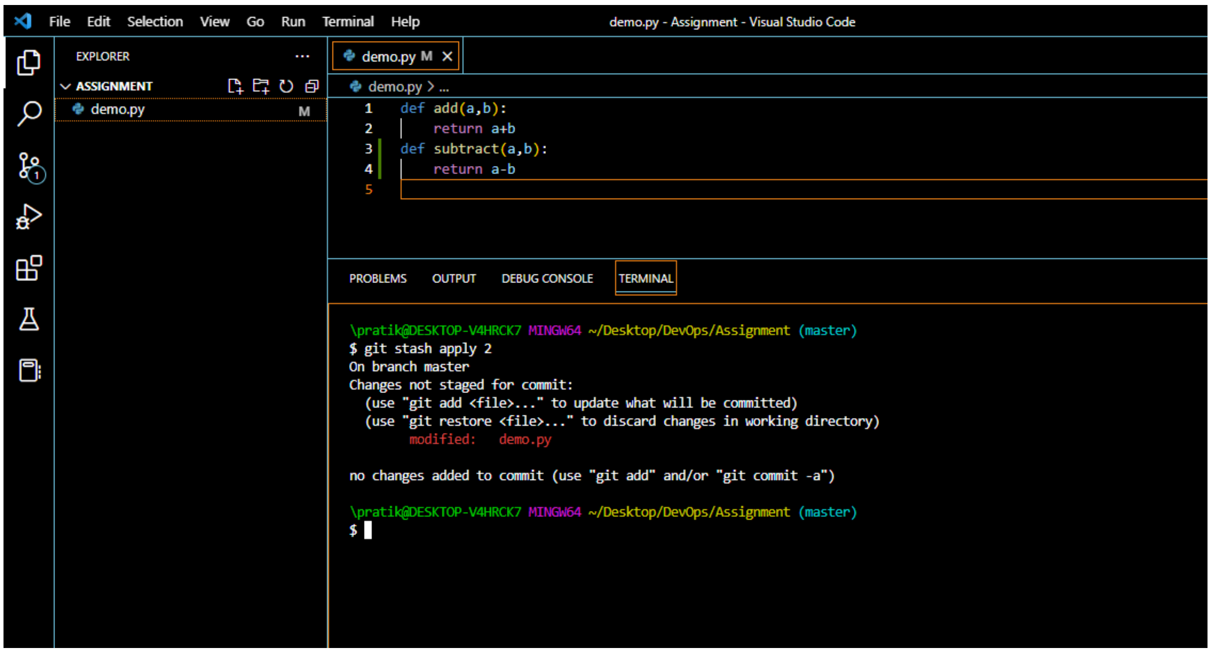

File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash

```python
def add(a,b):
    return a+b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash list
stash@{0}: WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash
Saved working directory and index state WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash
Saved working directory and index state WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash list
stash@{0}: WIP on master: 0c973f0 Added addition funtion in demo file
stash@{1}: WIP on master: 0c973f0 Added addition funtion in demo file
stash@{2}: WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```

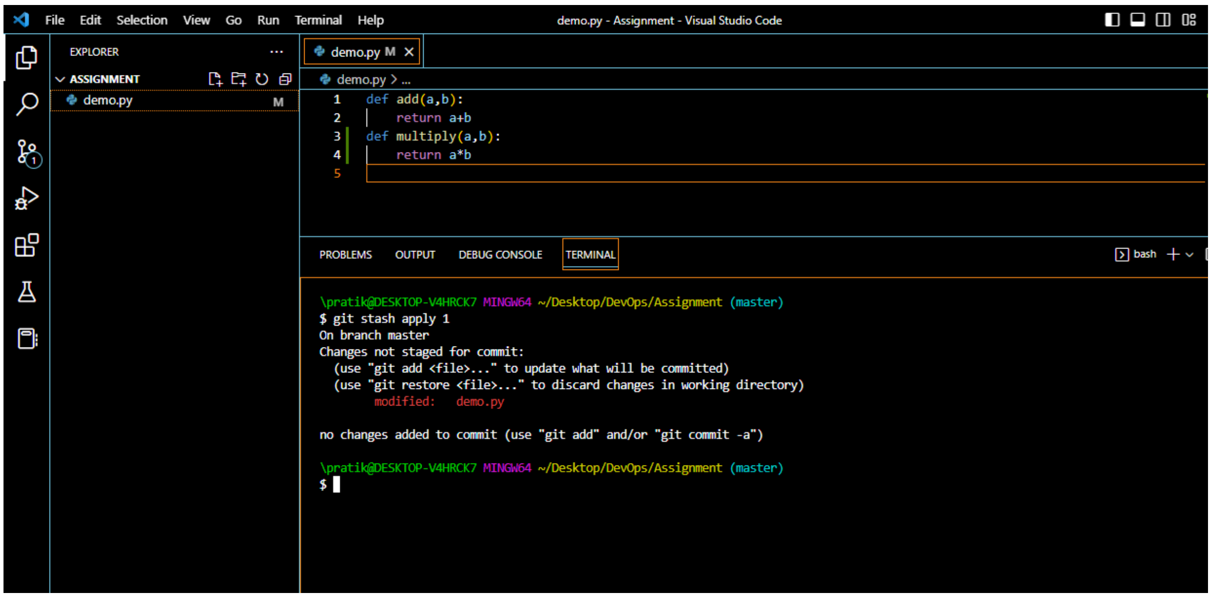
We can use the command “git stash list” to see our stashes with their indices.

Now, we can use the command “git stash apply [index]” to apply the modifications of a stash to our working directory. From there, we can commit the changes if we like.


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

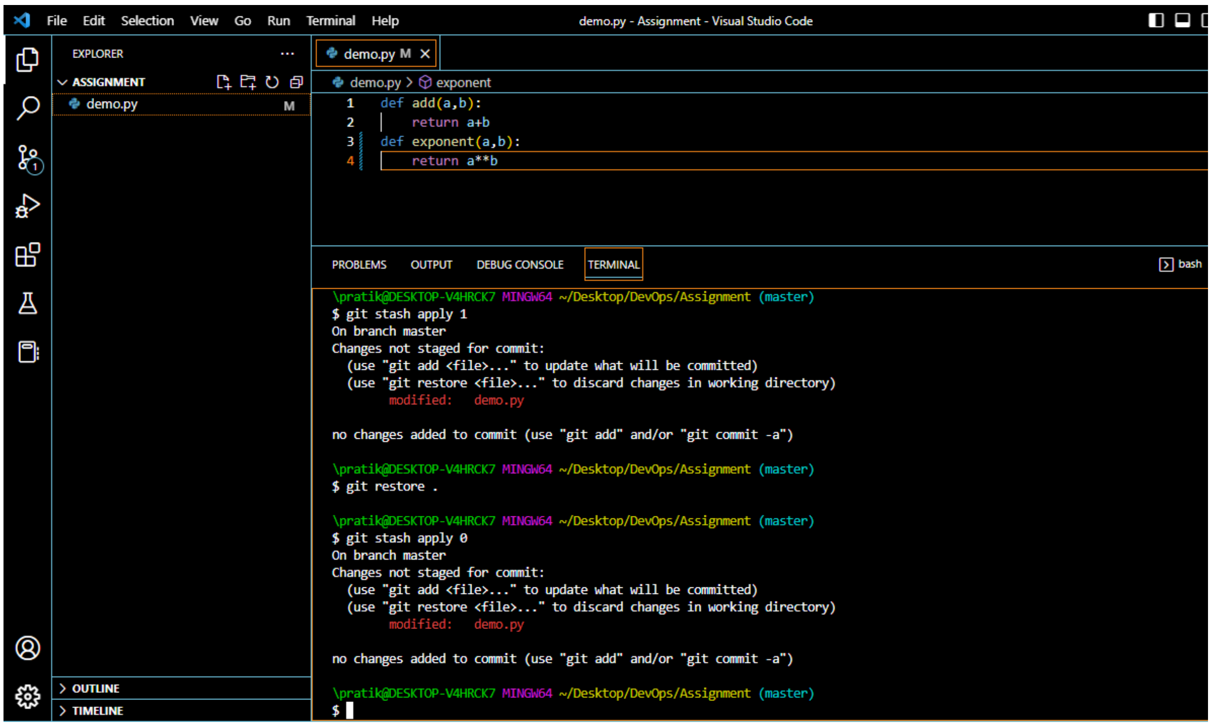
```python
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash apply 2
On branch master
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:   demo.py

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

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash

```
def add(a,b):
    return a+b
def multiply(a,b):
    return a*b
```

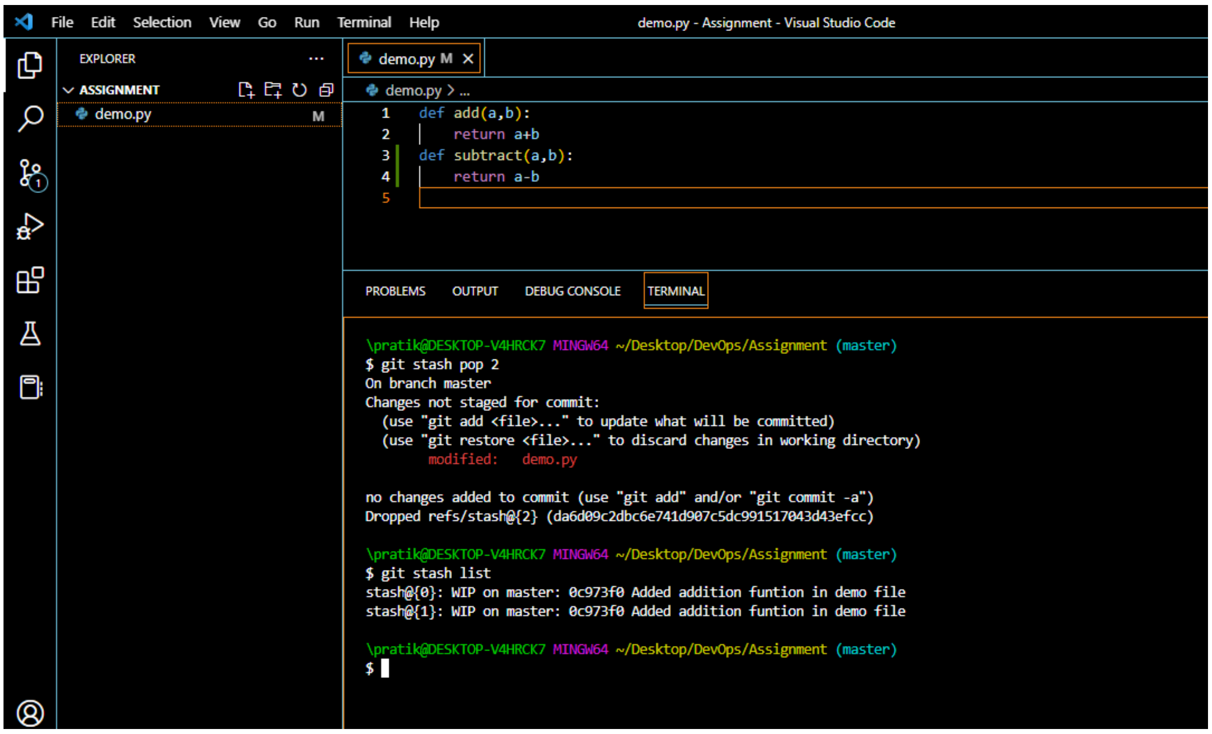
```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash apply 1
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   demo.py

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

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ 
```


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py M
demo.py > exponent
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash
OUTLINE
TIMELINE

```
def add(a,b):
    return a+b
def exponent(a,b):
    return a**b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash apply 1
On branch master
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:   demo.py

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

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git restore .

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash apply 0
On branch master
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:   demo.py

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

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ 
```

We can use the “git pop [index]” command to apply the changes of a stash while at the same time, removing that stash from the stack we have created.

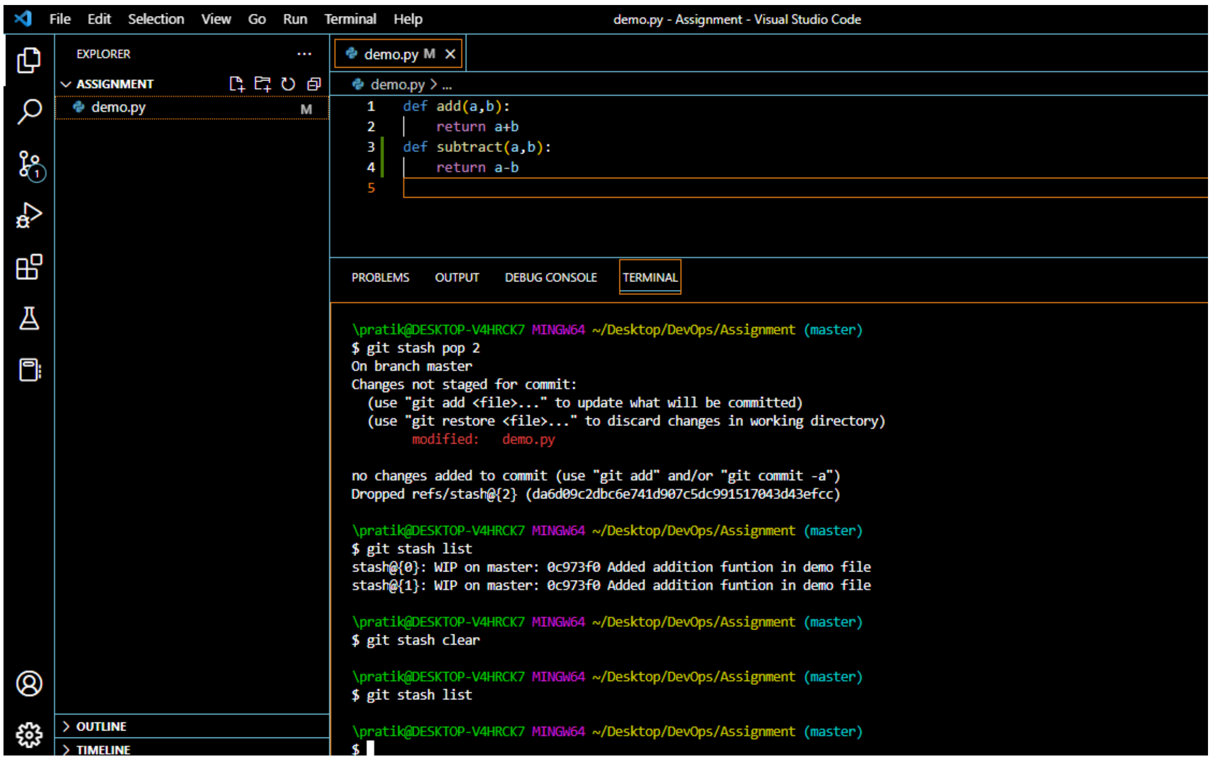

File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```python
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash pop 2
On branch master
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:   demo.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{2} (da6d09c2dbc6e741d907c5dc991517043d43efcc)

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash list
stash@{0}: WIP on master: 0c973f0 Added addition funtion in demo file
stash@{1}: WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ 
```

As you can see, one stash was removed and only two remain.

We can use the command “git stash clear” to drop all the stashes we have created.


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
OUTLINE
TIMELINE

```python
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
```

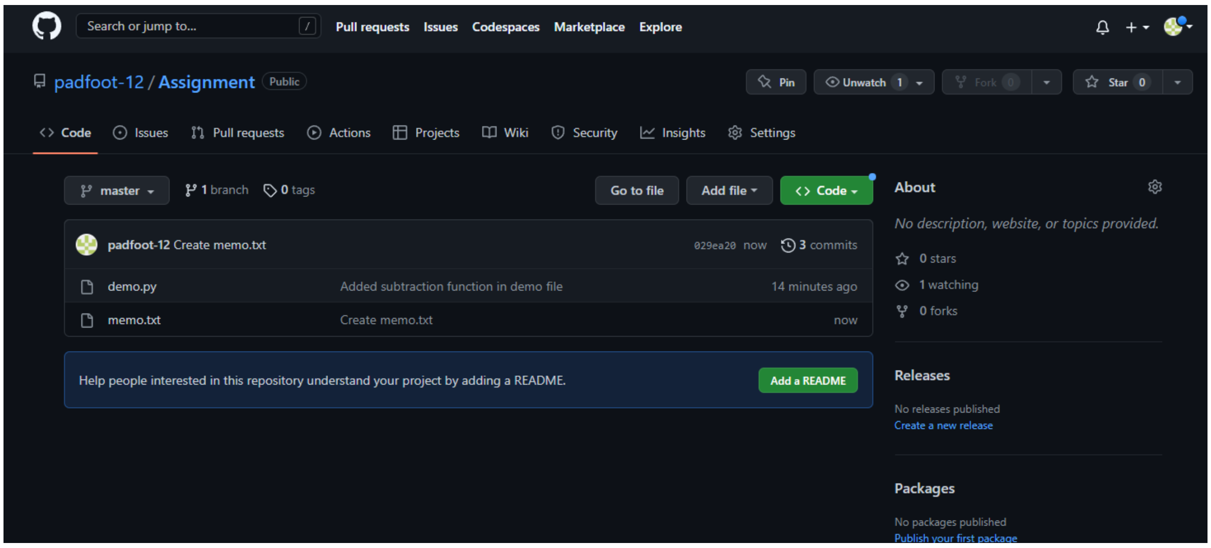
```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash pop 2
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   demo.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{2} (da6d09c2dbc6e741d907c5dc991517043d43efcc)

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash list
stash@{0}: WIP on master: 0c973f0 Added addition funtion in demo file
stash@{1}: WIP on master: 0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash clear

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git stash list

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ 
```

We have no stashes left.

## Q2. By using a sample example of your choice, use the git fetch command and also use the git merge command and describe the whole process through a screenshot with all the commands and their output in git bash.

The git fetch command is used to retrieve the changes made in our remote repository to our local repository. What this means, basically, is downloading additions made to the remote repository to our local repository. It does not apply those changes to our local repo directly.

Here, we are creating a next text file called memo in our remote repository.



padfoot-12 / Assignment Public

master · 1 branch · 0 tags

padfoot-12 Create memo.txt   029ea20 now   3 commits

| | | |
|---|---|---|
| demo.py | Added subtraction function in demo file | 14 minutes ago |
| memo.txt | Create memo.txt | now |

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

0 stars
1 watching
0 forks

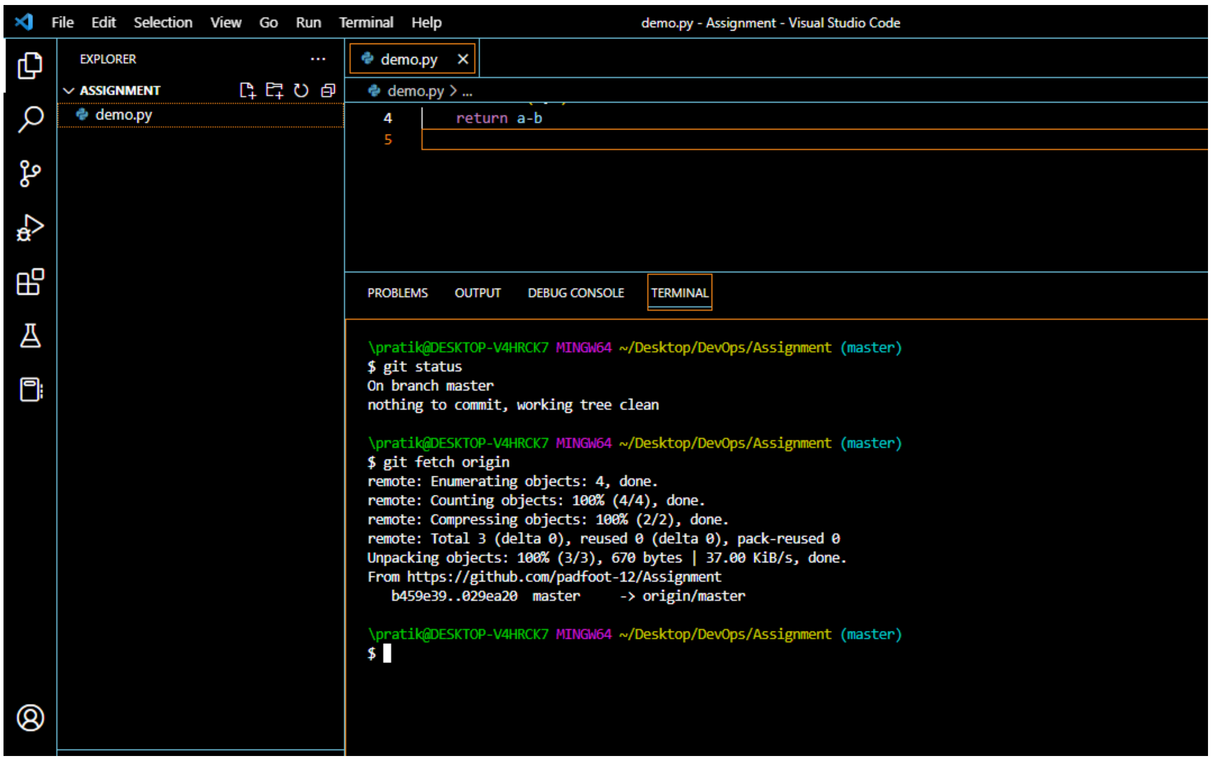
Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

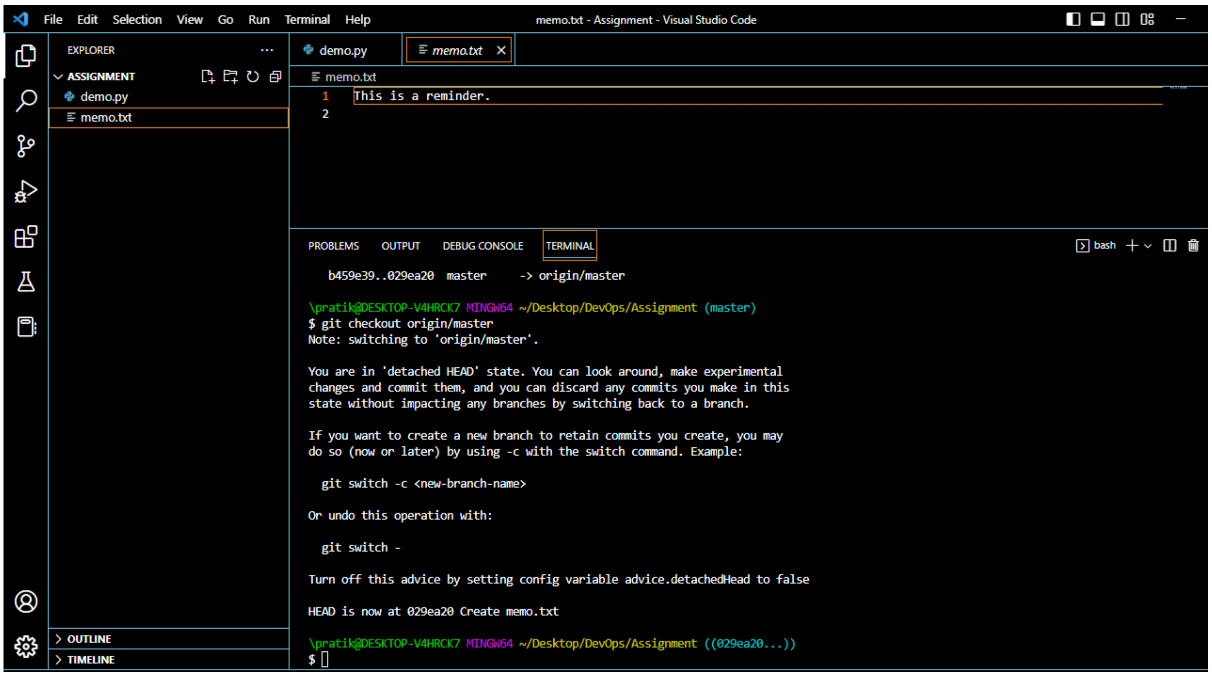
Now in our local repository, we go ahead and perform the "git fetch origin" command. This will fetch all changes through our remote "origin".



```
    return a-b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git status
On branch master
nothing to commit, working tree clean

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 670 bytes | 37.00 KiB/s, done.
From https://github.com/padfoot-12/Assignment
   b459e39..029ea20  master     -> origin/master

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```

To see the changes, we can checkout the remote branch using command “git checkout origin/master”.



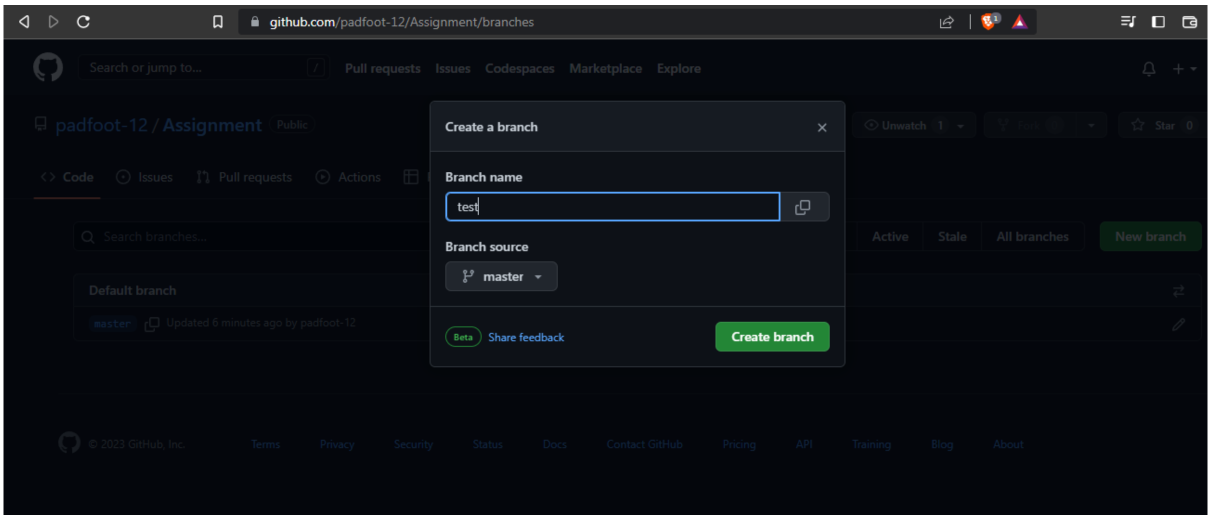
```
This is a reminder.
```

```
  b459e39..029ea20  master     -> origin/master

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git checkout origin/master
Note: switching to 'origin/master'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 029ea20 Create memo.txt

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment ((029ea20...))
$
```

As you can see, now we can see the memo file in our local repository.

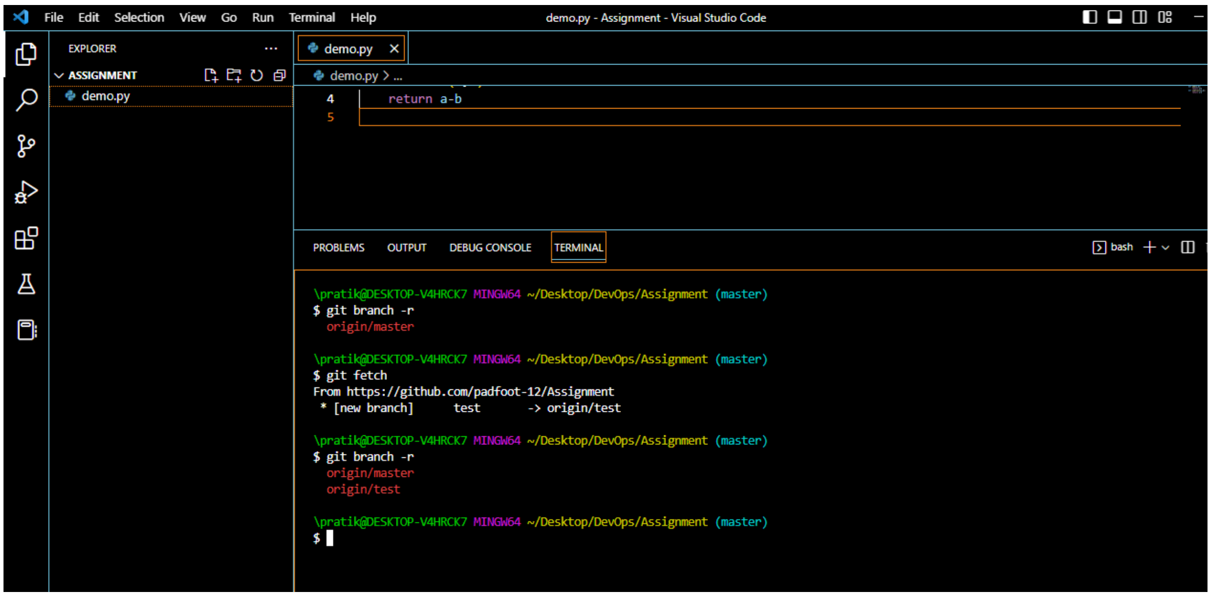
Not only can fetch command fetch changes in a branch, it can fetch a whole new branch in its entirety.

Suppose we create a new branch called “test” in our local repository. Now, this branch will not be available in our local repository.



When we perform the command “git branch –r” it won’t show us the new branch.

Use the command “git fetch” to fetch the new branch.



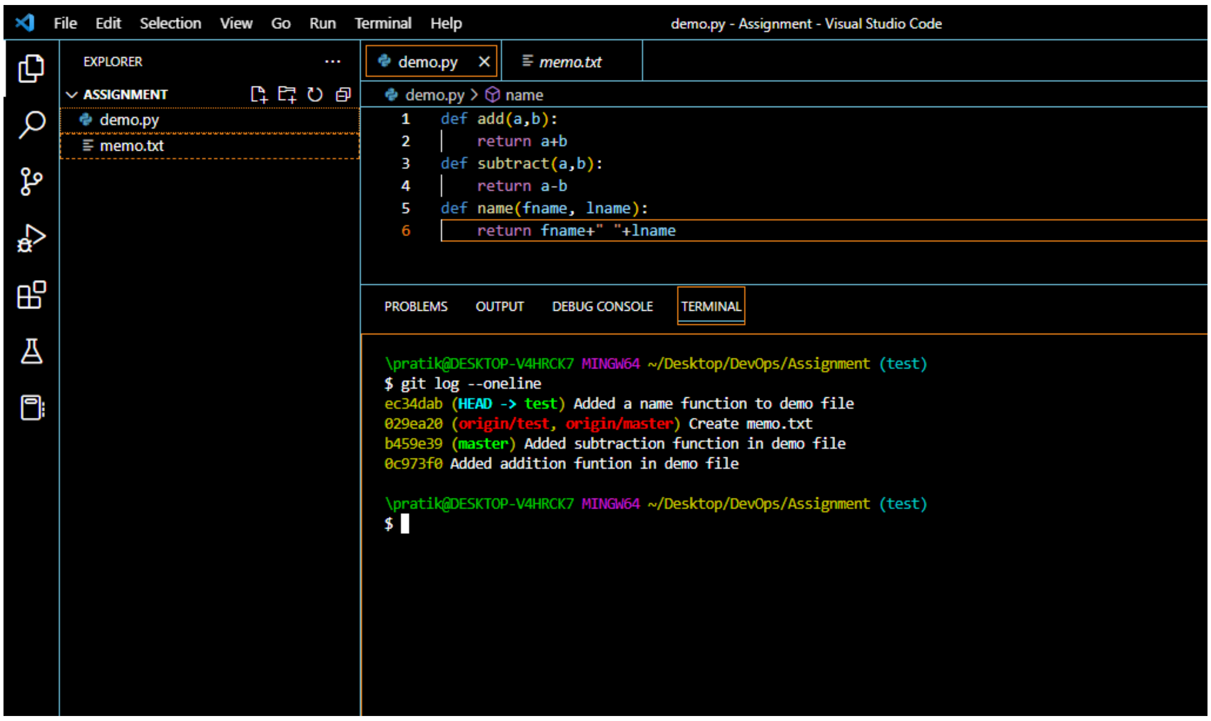
```
    return a-b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git branch -r
  origin/master

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git fetch
From https://github.com/padfoot-12/Assignment
 * [new branch]      test       -> origin/test

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git branch -r
  origin/master
  origin/test

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```

As you can see, the new branch is now visible in our local repository.

Git “merge” command is used to bring one branch up to date with the commits made on another branch.

Suppose we add a name function in our new branch test.

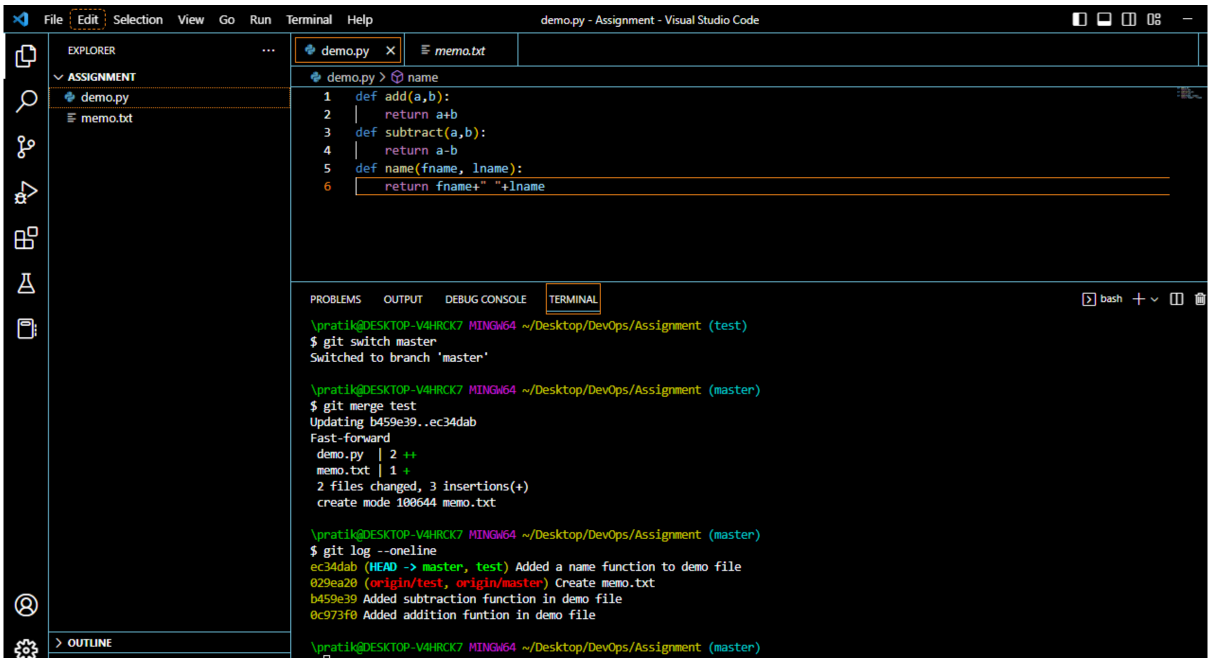


```python
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
def name(fname, lname):
    return fname+" "+lname
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (test)
$ git log --oneline
ec34dab (HEAD -> test) Added a name function to demo file
029ea20 (origin/test, origin/master) Create memo.txt
b459e39 (master) Added subtraction function in demo file
0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (test)
$
```

As we can see in the logs, the branch master is behind the test branch. If we want to include this new function in our master branch, we need to bring it up to date with our test branch, where the HEAD is pointing.

Switch to master branch using “git switch master”. Use command “git merge test” to bring our master branch up to date with the test branch.



```python
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
def name(fname, lname):
    return fname+" "+lname
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (test)
$ git switch master
Switched to branch 'master'

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git merge test
Updating b459e39..ec34dab
Fast-forward
 demo.py  | 2 ++
 memo.txt | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 memo.txt

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git log --oneline
ec34dab (HEAD -> master, test) Added a name function to demo file
029ea20 (origin/test, origin/master) Create memo.txt
b459e39 Added subtraction function in demo file
0c973f0 Added addition funtion in demo file

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
```

As you can see, now our master branch is up to date with the test branch, and has the name function too.

## Q3. State the difference between git fetch and git pull by doing a practical example in your git bash and attach a screenshot of all the processes.

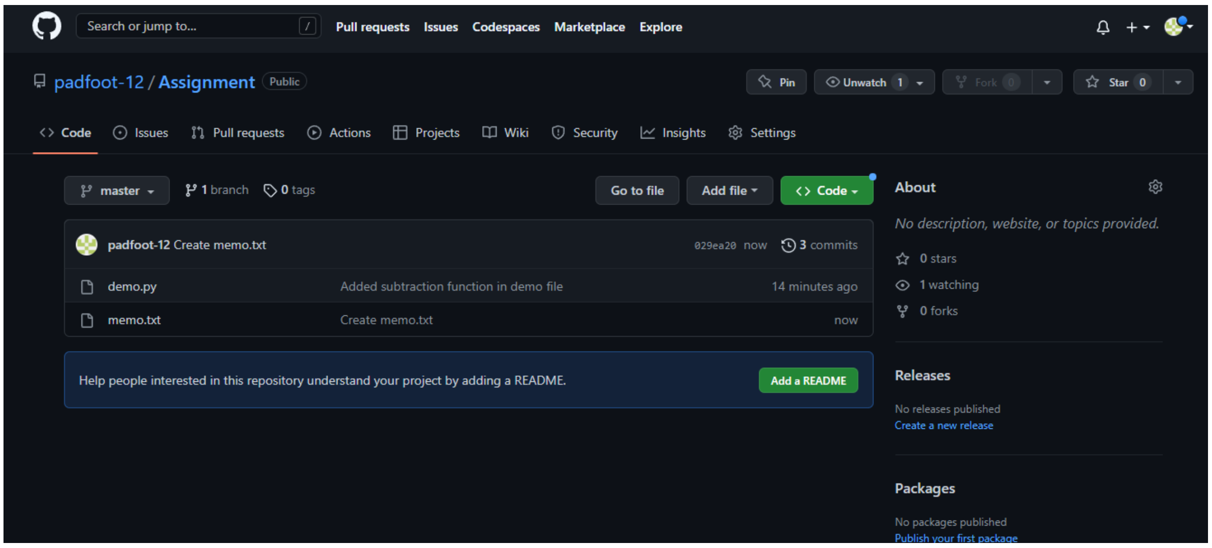
The git fetch command is used to retrieve the changes made in our remote repository to our local repository. What this means, basically, is downloading additions made to the remote repository to our local repository. It does not apply those changes to our local repo directly.

The git pull command is used to apply the changes fetched from a remote repository to our local repository. The pull command downloads and implements the changes as opposed to just downloading them, like the fetch command.

Pull =fetch + merge

**GIT fetch**

Suppose we create a new file memo in our remote repository.



padfoot-12 / Assignment Public

padfoot-12 Create memo.txt 029ea20 now 3 commits

| | | |
|---|---|---|
| demo.py | Added subtraction function in demo file | 14 minutes ago |
| memo.txt | Create memo.txt | now |

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

0 stars

1 watching

0 forks

Releases

No releases published

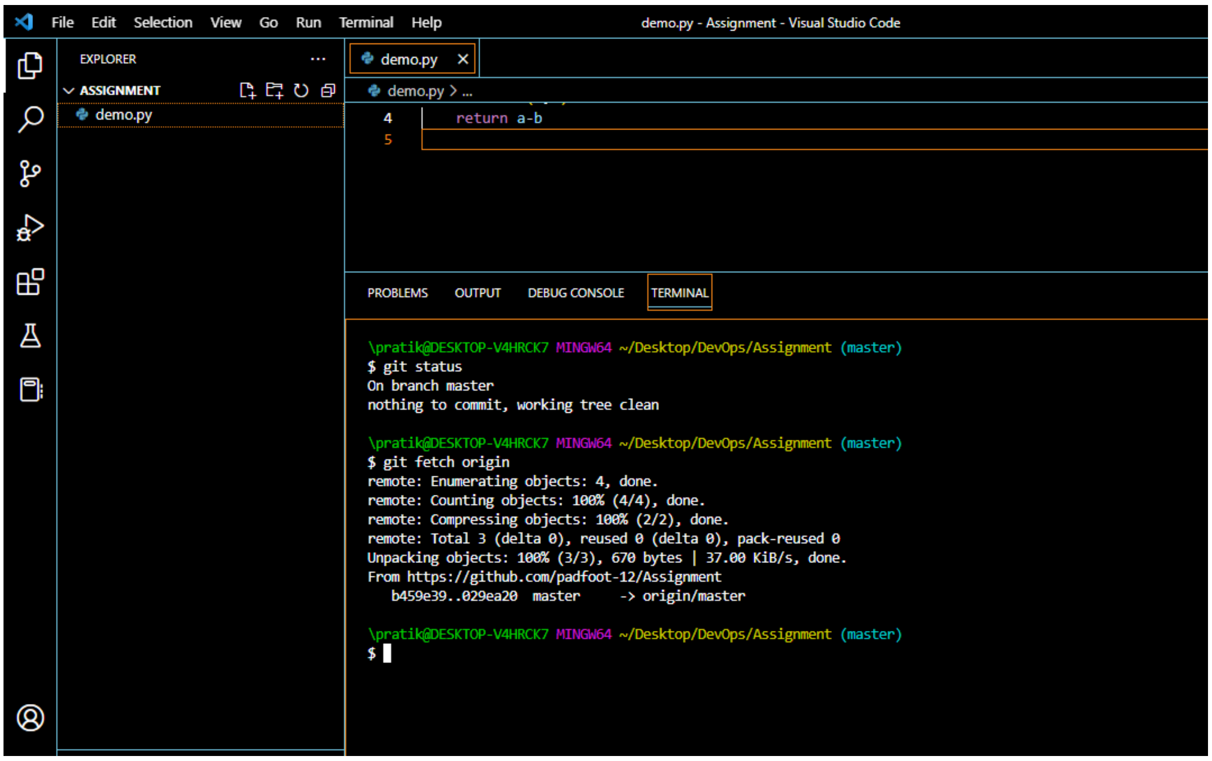
Create a new release

Packages

No packages published

Publish your first package

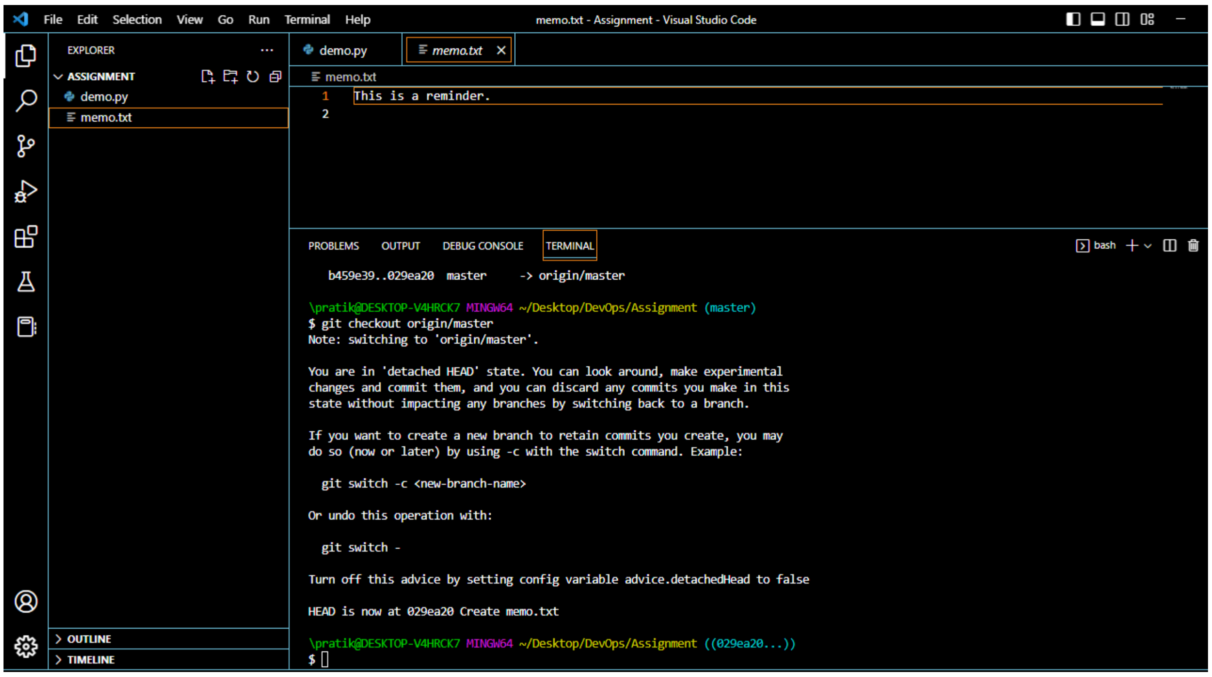
This will not be visible in our local repository. We can fetch it using the command “git fetch origin”.


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL

```
    return a-b
```

```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git status
On branch master
nothing to commit, working tree clean

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 670 bytes | 37.00 KiB/s, done.
From https://github.com/padfoot-12/Assignment
   b459e39..029ea20  master     -> origin/master

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$
```

We can see the fetched changes by using “git checkout origin/master” command. They haven’t been directly applied to our local repository.

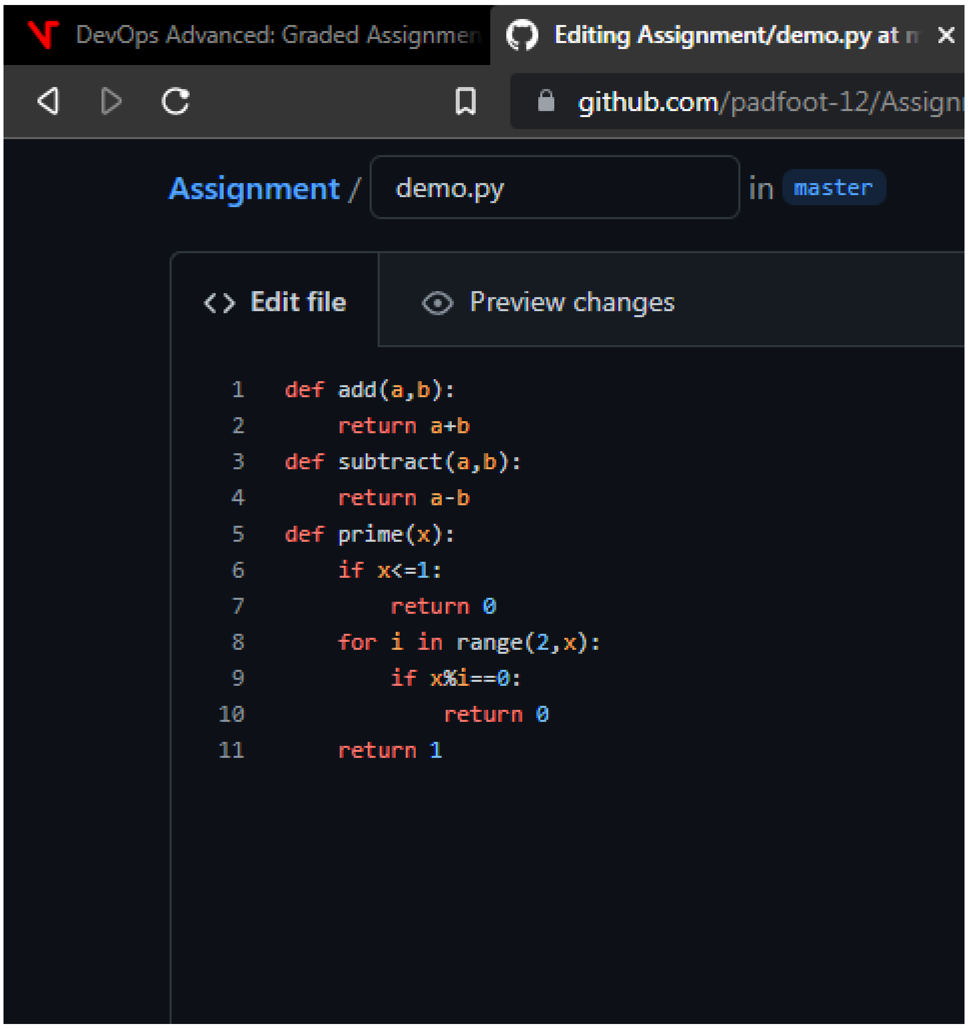

File Edit Selection View Go Run Terminal Help
memo.txt - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py
memo.txt
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bash

```
This is a reminder.
```

```
  b459e39..029ea20  master     -> origin/master

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git checkout origin/master
Note: switching to 'origin/master'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 029ea20 Create memo.txt

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment ((029ea20...))
$
```

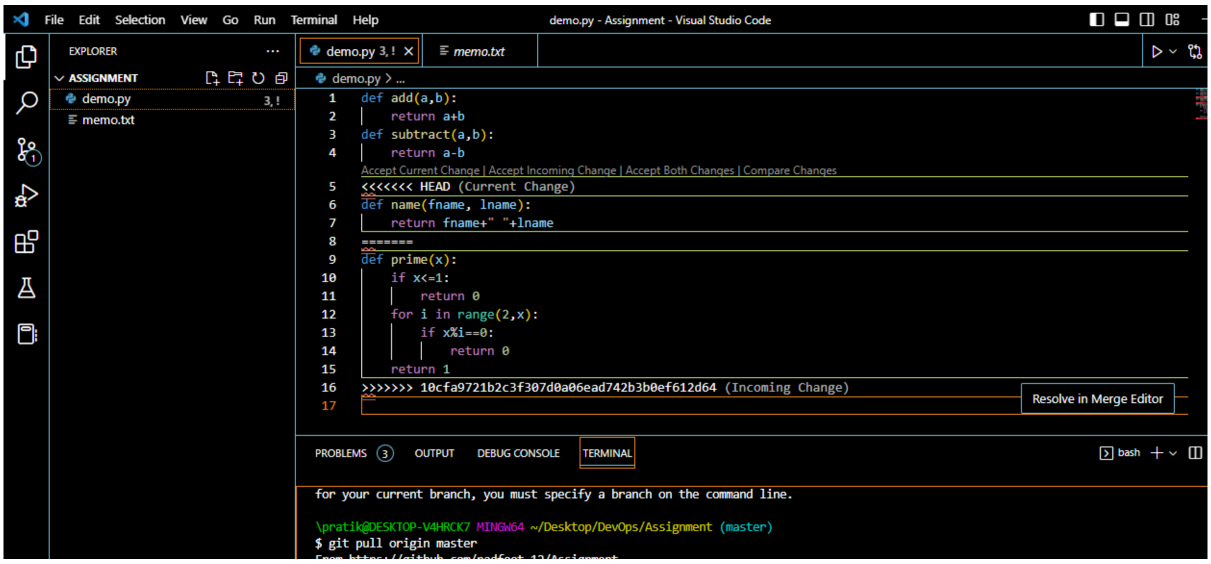
## GIT pull

Suppose we add a prime number function in the demo file in our remote repository, and want to apply those changes to our local repository.


DevOps Advanced: Graded Assignmen
Editing Assignment/demo.py at m
github.com/padfoot-12/Assign
Assignment / demo.py in master
Edit file
Preview changes

```
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
def prime(x):
    if x<=1:
        return 0
    for i in range(2,x):
        if x%i==0:
            return 0
    return 1
```

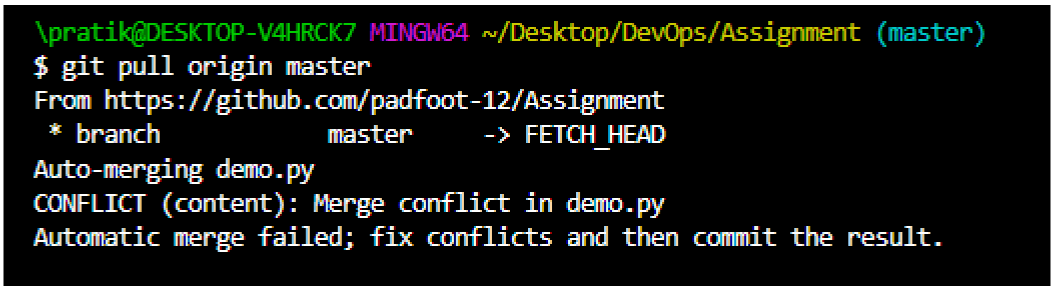
We can use the command “git pull origin master” to apply the changes made in our remote branch origin/master to our local branch master.


File Edit Selection View Go Run Terminal Help
demo.py - Assignment - Visual Studio Code
EXPLORER
ASSIGNMENT
demo.py 3, !
memo.txt
demo.py > ...
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
Resolve in Merge Editor
PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
bash

```
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
<<<<<<< HEAD (Current Change)
def name(fname, lname):
    return fname+" "+lname
=======
def prime(x):
    if x<=1:
        return 0
    for i in range(2,x):
        if x%i==0:
            return 0
    return 1
>>>>>>> 10cfa9721b2c3f307d0a06ead742b3b0ef612d64 (Incoming Change)
```

```
for your current branch, you must specify a branch on the command line.

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git pull origin master
```

After resolving the merge conflicts, we get the following message.



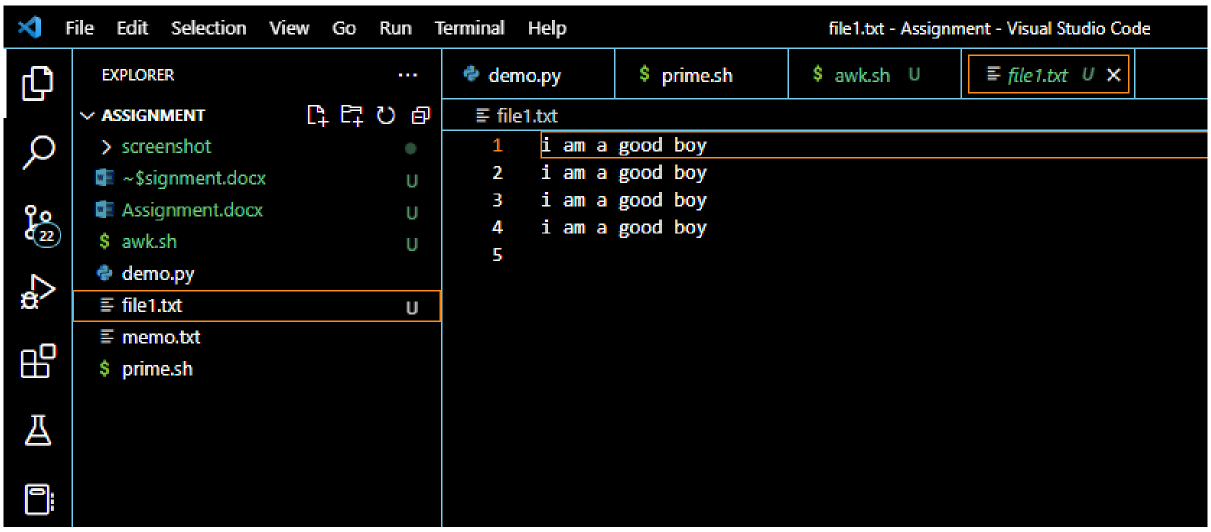
```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ git pull origin master
From https://github.com/padfoot-12/Assignment
 * branch            master     -> FETCH_HEAD
Auto-merging demo.py
CONFLICT (content): Merge conflict in demo.py
Automatic merge failed; fix conflicts and then commit the result.
```

## Q4. Try to find out about the awk command and use it while reading a file created by yourself. Also, make a bash script file and try to find out the prime number from the range 1 to 20.

## The whole process should be carried out and by using the history command, give the screenshot of all the processes being carried out.

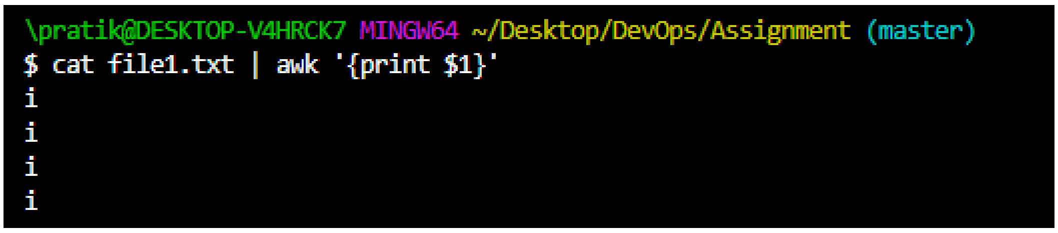
awk command searches files for text containing a pattern. When a line or text matches, awk performs a specific action on that line/text. The Program statement tells awk what operation to do.

Here, we are creating a file called file1.txt and writing some lines in it.



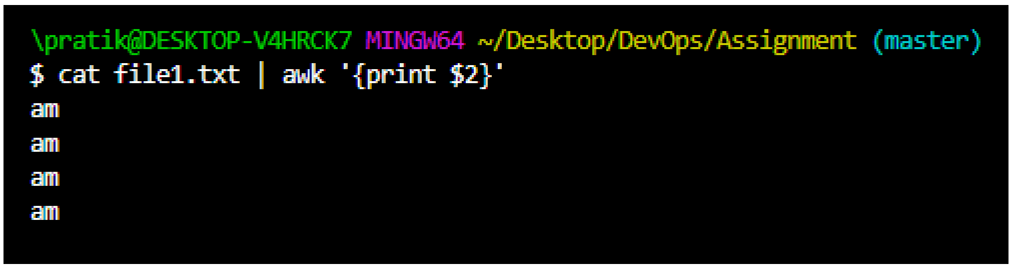
```
i am a good boy
i am a good boy
i am a good boy
i am a good boy
```

We execute the command “cat file1.txt | awk ‘{print $1}’” to print the first column of every line.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ cat file1.txt | awk '{print $1}'
i
i
i
i
```

We use the command “cat file1.txt | awk ‘{print $2}’” to print the second column of every line.

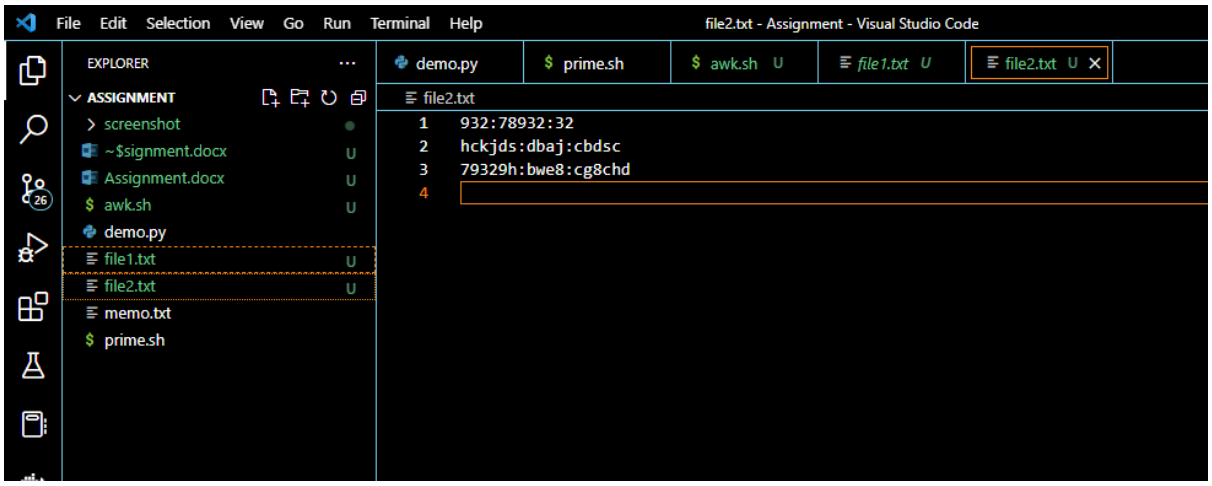


```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ cat file1.txt | awk '{print $2}'
am
am
am
am
```

So on and so forth.

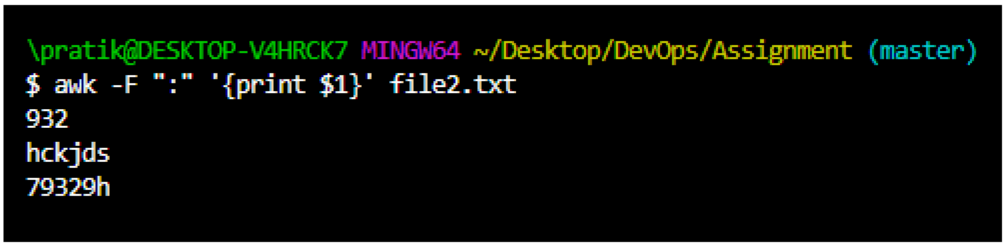
The default separator used is “ “ (space). We can change it to Our preferred separator using the –F option.

Here, we are creating a file called file2.txt and writing some lines in it.



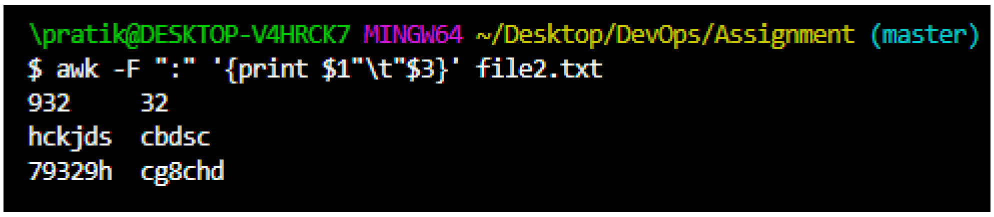
```
932:78932:32
hckjds:dbaj:cbdsc
79329h:bwe8:cg8chd
```

Use the command “awk –F “:” ‘{print $1}’ file2.txt” to print the first column of every line separated by “:”.



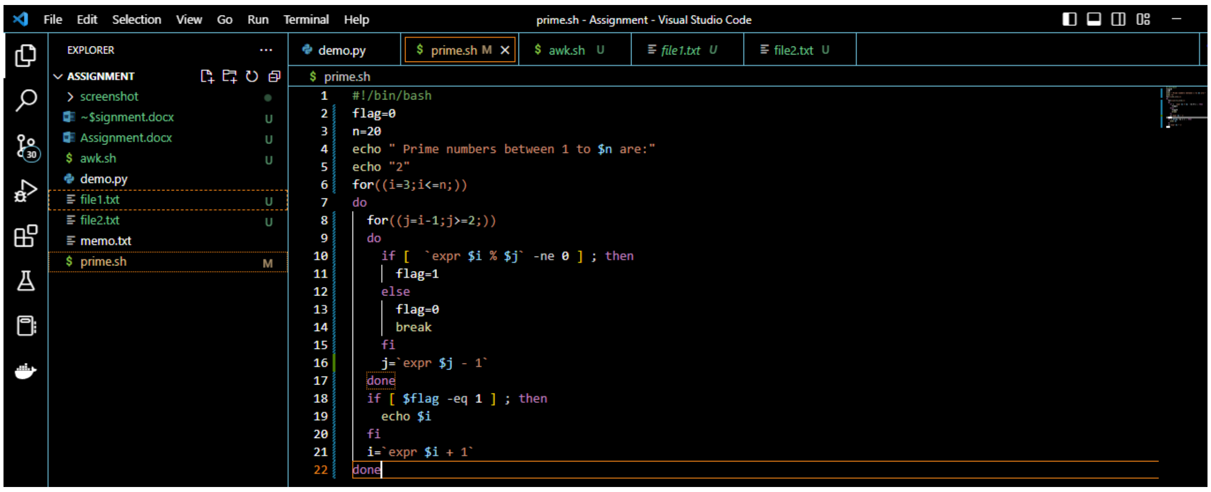
```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ awk -F ":" '{print $1}' file2.txt
932
hckjds
79329h
```

We can also print multiple columns at a time. Use command “awk –F “:” ‘{print $1”\t”$3}’ file2.txt”. In this command, we will print the first and third columns, that are separated by colon, and we will print a tab space between them, as specified by “\t”.

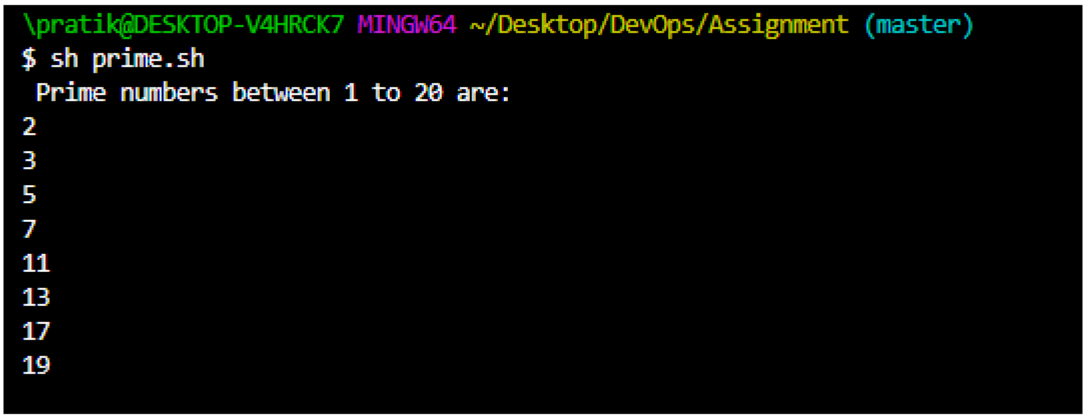


```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ awk -F ":" '{print $1"\t"$3}' file2.txt
932     32
hckjds  cbdsc
79329h  cg8chd
```

**Prime numbers between 1 to 20:**



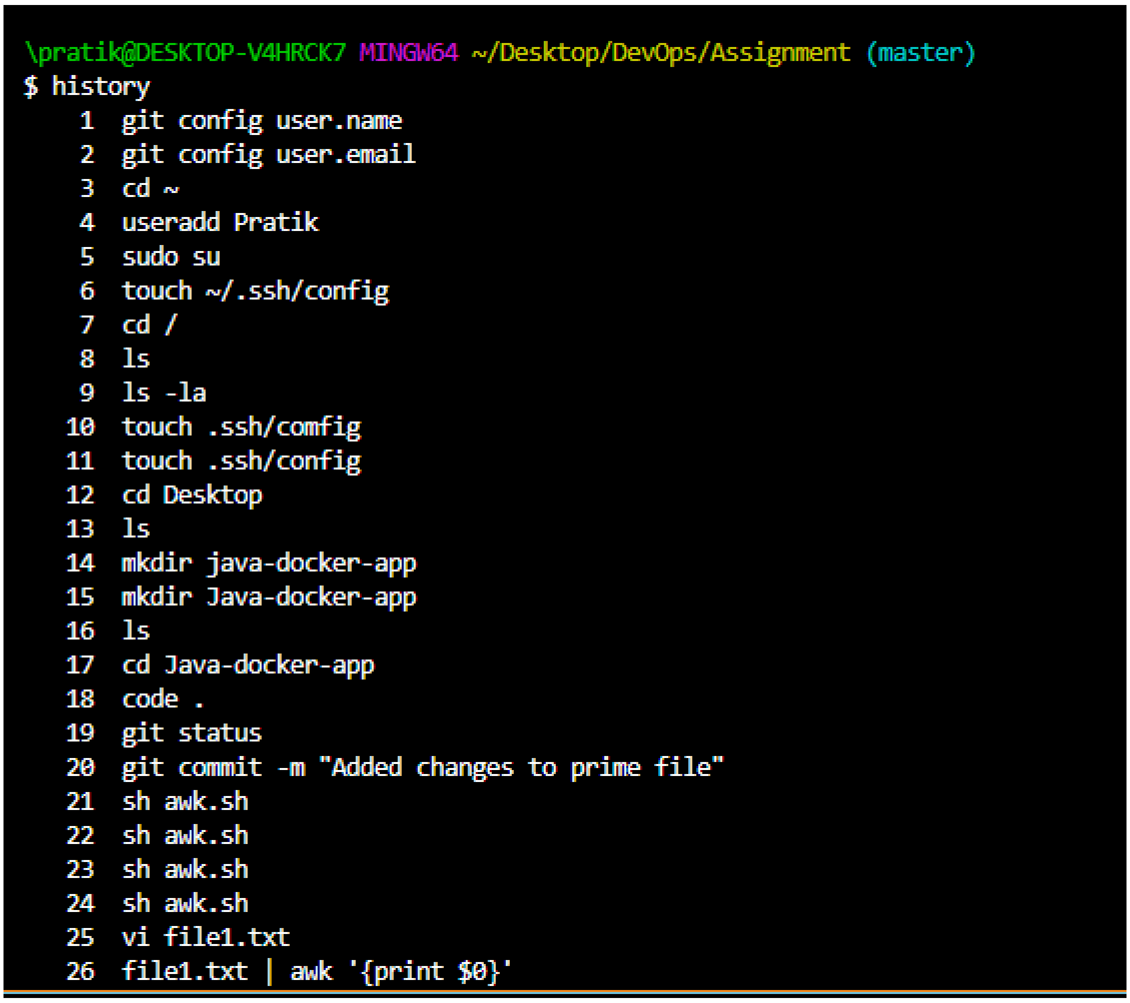
```
#!/bin/bash
flag=0
n=20
echo " Prime numbers between 1 to $n are:"
echo "2"
for((i=3;i<=n;))
do
  for((j=i-1;j>=2;))
  do
    if [  `expr $i % $j` -ne 0 ] ; then
      flag=1
    else
      flag=0
      break
    fi
    j=`expr $j - 1`
  done
  if [ $flag -eq 1 ] ; then
    echo $i
  fi
  i=`expr $i + 1`
done
```



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ sh prime.sh
 Prime numbers between 1 to 20 are:
2
3
5
7
11
13
17
19
```

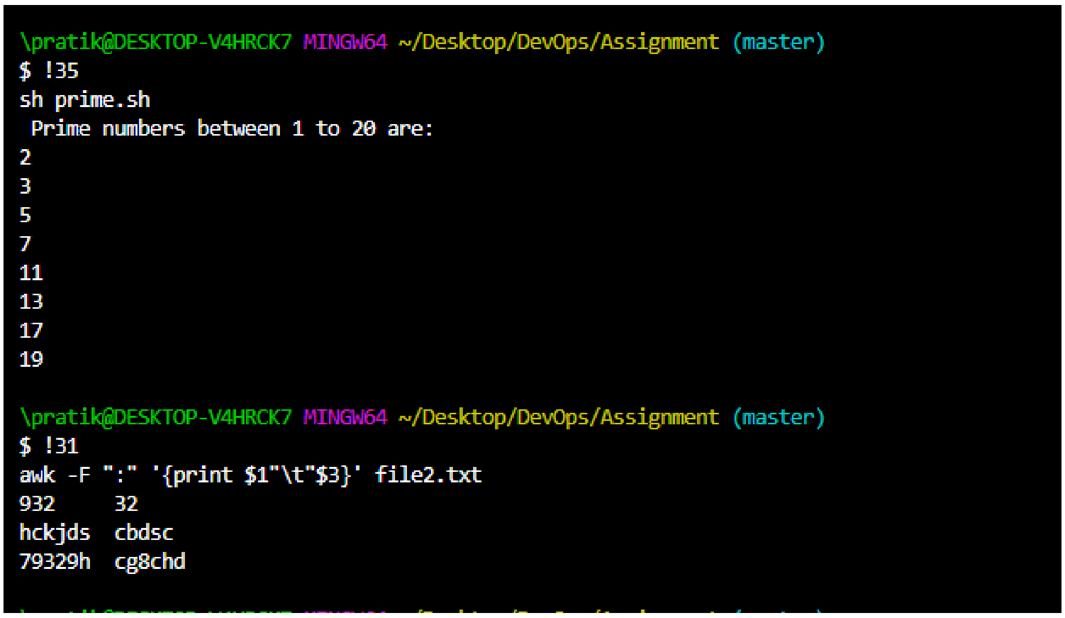
**History command**

The history command lists the history of commands executed in the terminal in a chronological order, assigning a number to each command.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ history
    1  git config user.name
    2  git config user.email
    3  cd ~
    4  useradd Pratik
    5  sudo su
    6  touch ~/.ssh/config
    7  cd /
    8  ls
    9  ls -la
   10  touch .ssh/comfig
   11  touch .ssh/config
   12  cd Desktop
   13  ls
   14  mkdir java-docker-app
   15  mkdir Java-docker-app
   16  ls
   17  cd Java-docker-app
   18  code .
   19  git status
   20  git commit -m "Added changes to prime file"
   21  sh awk.sh
   22  sh awk.sh
   23  sh awk.sh
   24  sh awk.sh
   25  vi file1.txt
   26  file1.txt | awk '{print $0}'
```

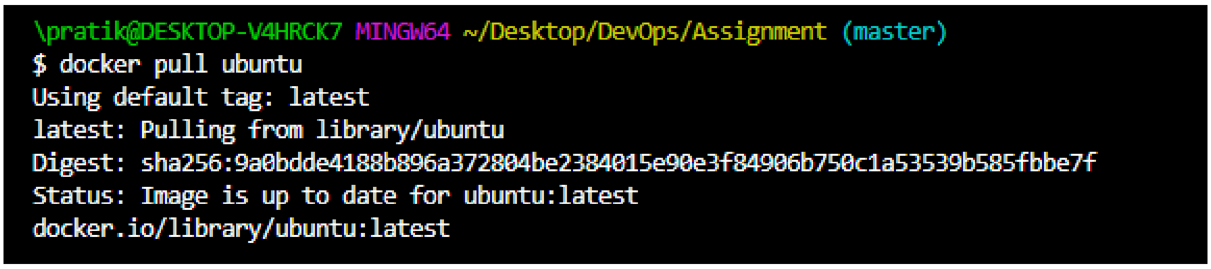
We can execute these commands again using their number. Use the "!" sign before the command number.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ !35
sh prime.sh
 Prime numbers between 1 to 20 are:
2
3
5
7
11
13
17
19

\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ !31
awk -F ":" '{print $1"\t"$3}' file2.txt
932     32
hckjds  cbdsc
79329h  cg8chd
```

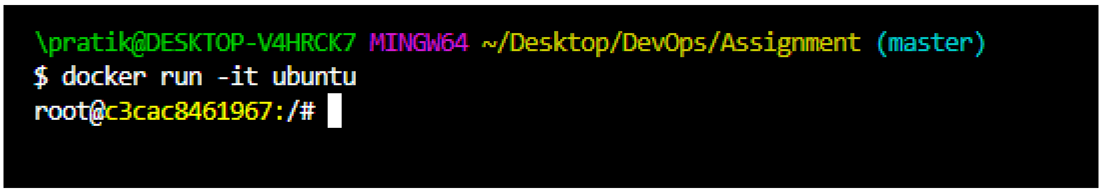
## Q5. Set up a container and run a Ubuntu operating system. For this purpose, you can make use of the docker hub and run the container in interactive mode.

First, we pull the Ubuntu image from dockerhub using the “docker pull ubuntu” command.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
```

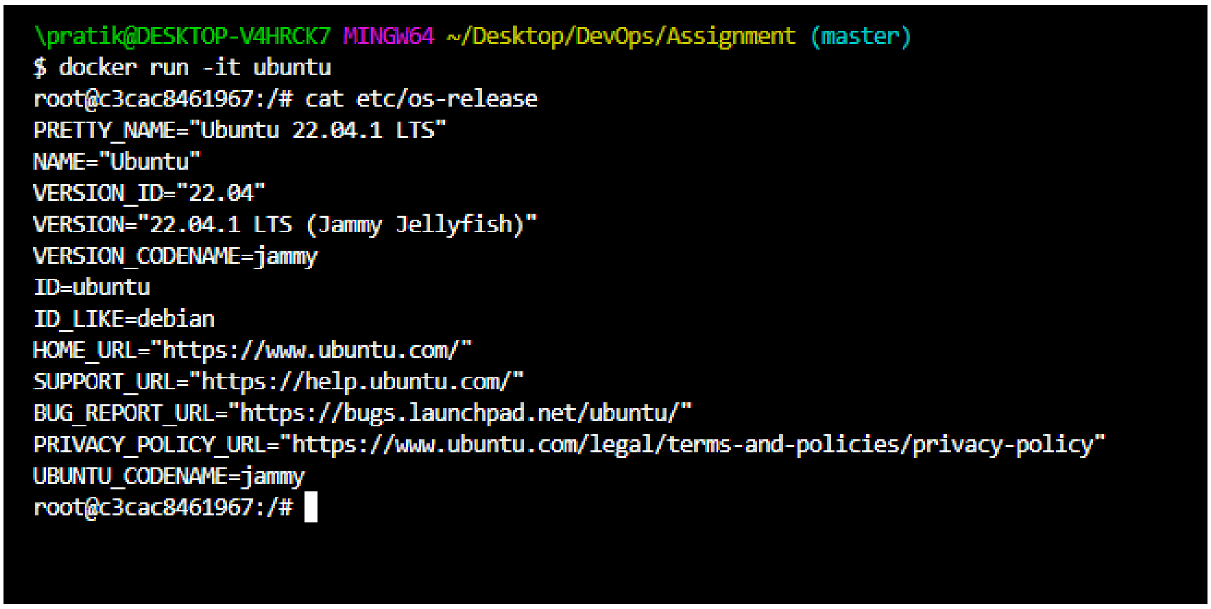
Next, we run this image using the command “docker run –it ubuntu”. The option –i means we are running the container in interactive mode. The option –t means we are running this container with terminal support. When we execute this command, a container is formed for this Ubuntu image and it is executed.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ docker run -it ubuntu
root@c3cac8461967:/#
```

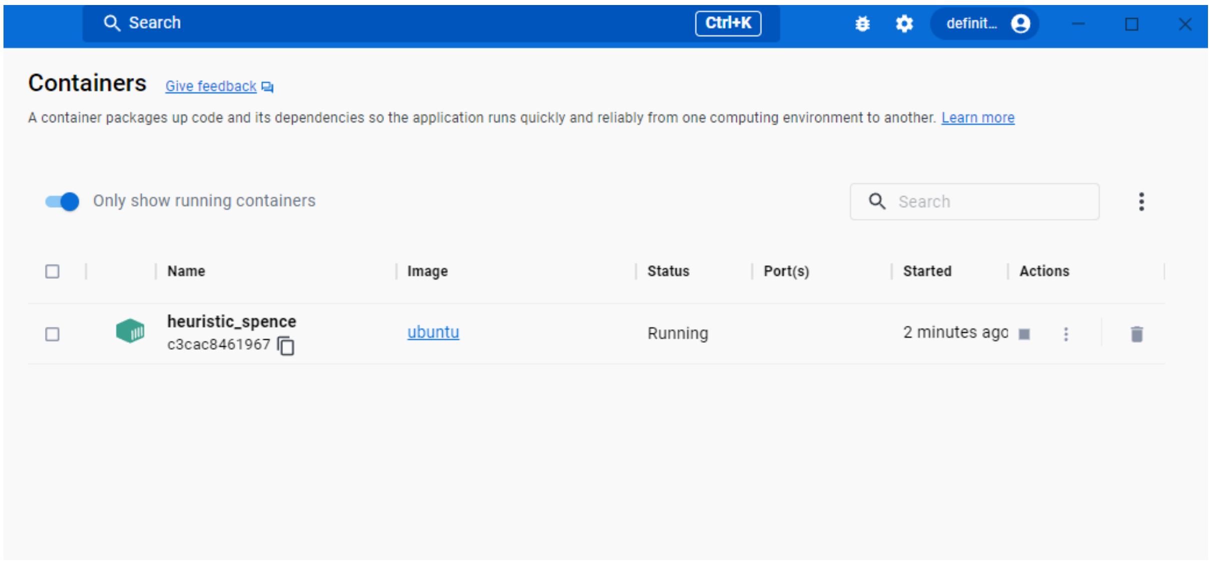
We have entered a Ubuntu environment and the terminal is accessible to us. The highlighted text after “root@” is the container ID.

When we execute the command “cat etc/os-release” in the terminal, we can see the OS is Ubuntu and its version.



```
\pratik@DESKTOP-V4HRCK7 MINGW64 ~/Desktop/DevOps/Assignment (master)
$ docker run -it ubuntu
root@c3cac8461967:/# cat etc/os-release
PRETTY_NAME="Ubuntu 22.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.1 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@c3cac8461967:/#
```

In our docker desktop application, we can see our container is successfully up and running.



A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another.

| Name | Image | Status | Port(s) | Started | Actions |
|---|---|---|---|---|---|
| heuristic_spence c3cac8461967 | ubuntu | Running |  | 2 minutes ago |  |