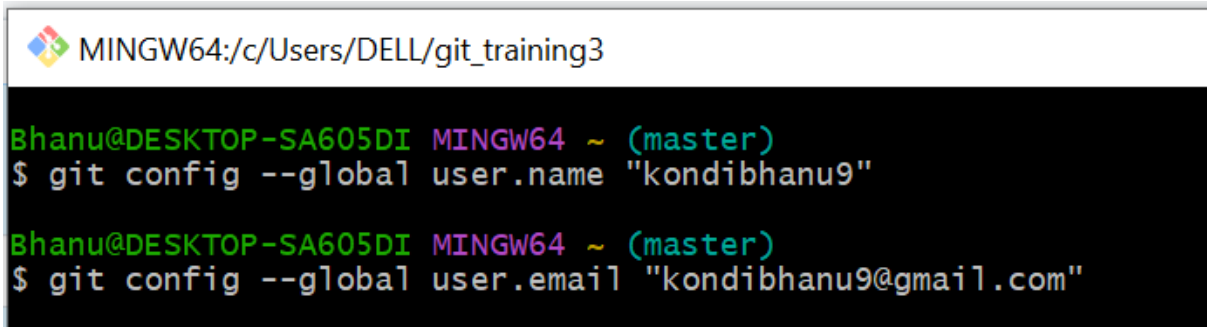
DevOps Assignment

Problem Statement :

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.

Git allows the multiple users to work on the same project.Each one will work on the different branches.if in case you need to pull the other developer's branch,and your code is partially complete,in that case you cannot commit the partial code,you have to remove your current changes and store them somewhere else,this can be done using a command called 'git stash'.

Git stash command saves the previously written code and then goes back to the last commit for a fresh start.Now you can add the new feature without disturbing the old one as it is saved locally.After committinh the new feature you can go on working with the old one which was incompleere and not committed.



*create a repository named git_testt and clone it.

*change to the git_testt repository using cd command

*create a file called dev1 using vi command

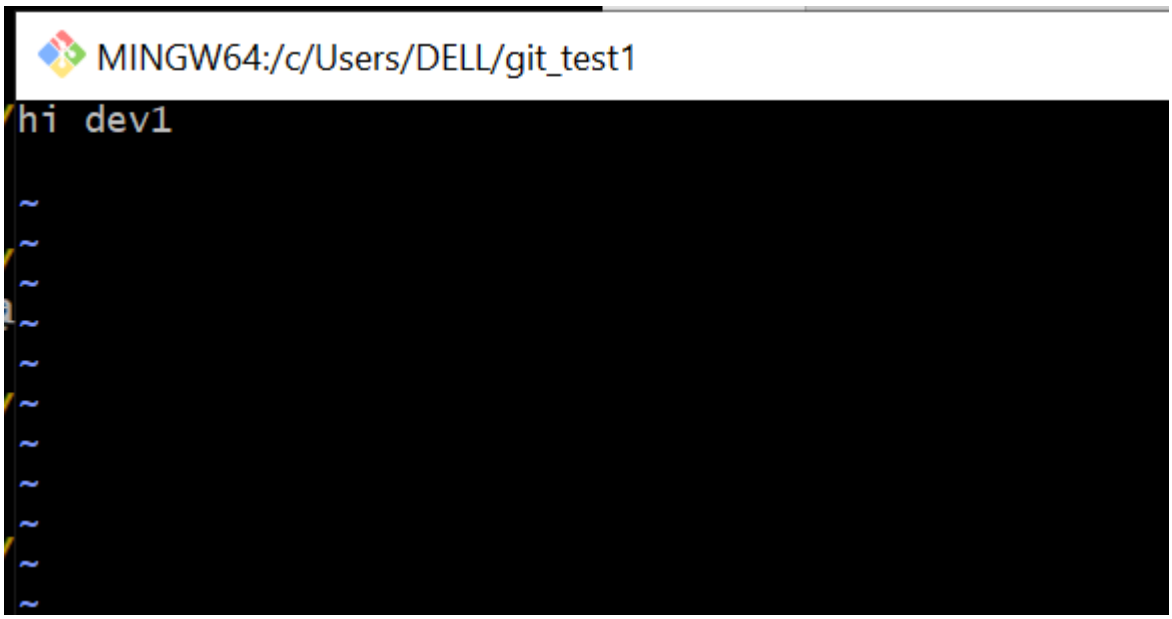
```
MINGW64:/c/Users/DELL/git_testt

Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ git clone https://github.com/kondibhanu9/git_testt.git
Cloning into 'git_testt'...
warning: You appear to have cloned an empty repository.

Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ cd git_testt

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ vi dev1

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        dev1

nothing added to commit but untracked files present (use "git add" to track)
```

*Now the file dev1 is under untracked state,so add it to staging area.

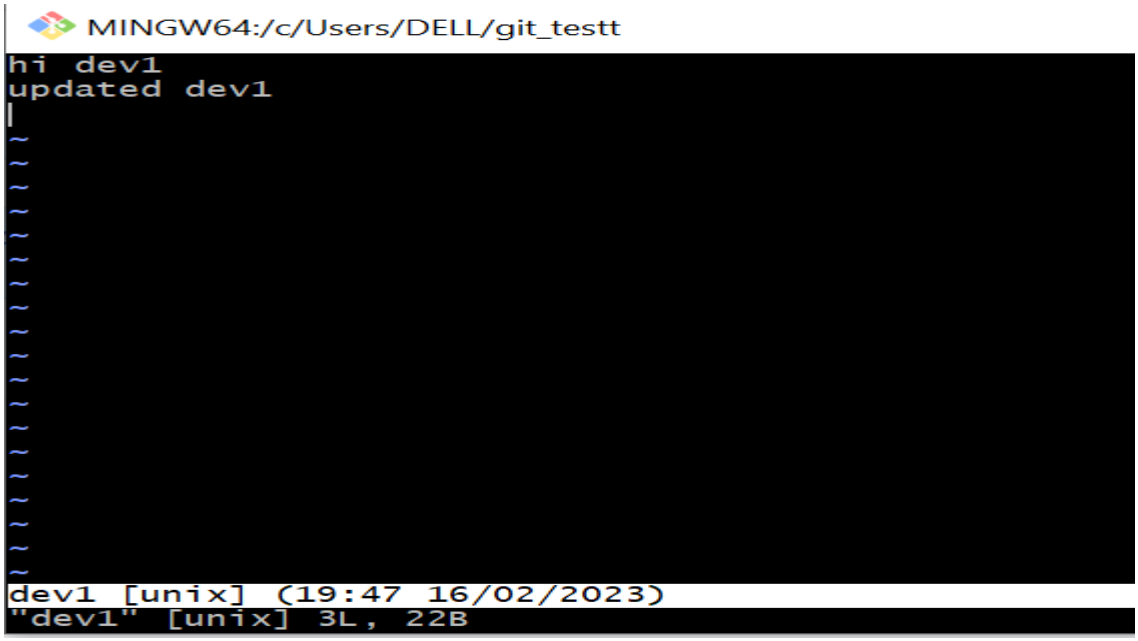
*use git add . to add the untracked file to the staging area.

*then commit the file.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git add .
warning: in the working copy of 'dev1', LF will be replaced by CRLF the next ti
e Git touches it

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git commit -m "dev1 added"
[main (root-commit) 691920b] dev1 added
 1 file changed, 2 insertions(+)
 create mode 100644 dev1
```



We have created a dev1 file,and now modify the file without commiting.



The dev1 file is now updated.

The file is now modified,and it is not committed,now if you want to pull the code on the other branch,then you have to remove these uncommitted changes,so use git stash command.

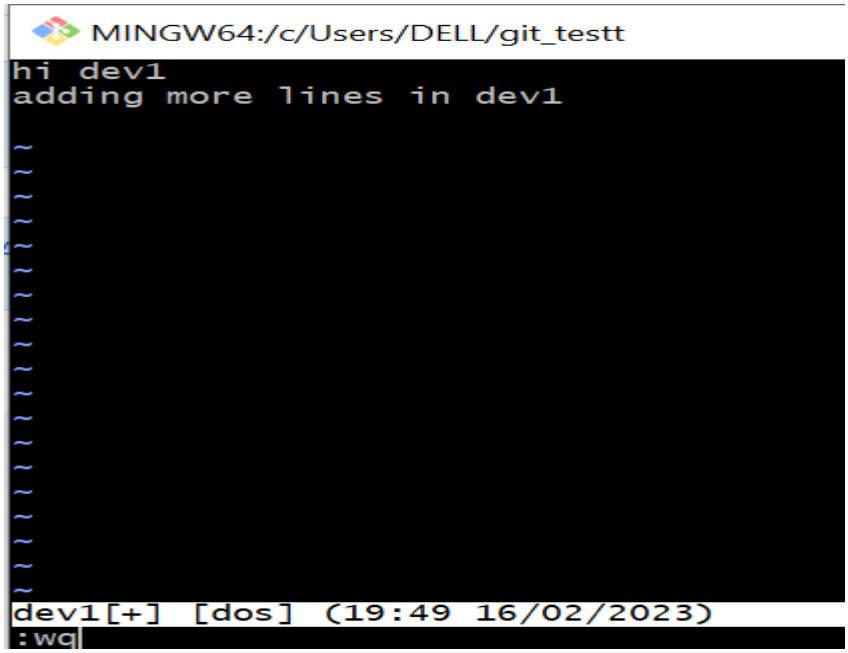
```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash
warning: in the working copy of 'dev1', LF will be replaced by CRLF the next tim
e Git touches it
Saved working directory and index state WIP on main: 691920b dev1 added

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash list
stash@{0}: WIP on main: 691920b dev1 added
```

.Once you use the git stash command,the modifications that you have made will not be there no longer.That means the modifications are not removed completely,but they are kept aside.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ vi dev1
```

View the file now

*modify the file and add the new line to create another stash



*the file is now modified state

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ vi dev1

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

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:   dev1

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

By default,running git stash will stash the changes that have been added to your index(staged changes)and unstages changes.To stash your untracked files,use git stash -u.

*as you have modified the file,stash them

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash
Saved working directory and index state WIP on main: 691920b dev1 added
```

Listing stashes:

You can create multiple slashes and view them using git stash list command

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash list
stash@{0}: WIP on main: 691920b dev1 added
stash@{1}: WIP on main: 691920b dev1 added
```

Git stash show:

Git stash show shows the changes recorded in the latest stash (stash@{0}) in the `--stat` format.

Git stash show index:

*git stash show can also be used done along with the index number to show the changes at that particular index.

Syntax: git stash show –index

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash show
 dev1 | 1 +
 1 file changed, 1 insertion(+)

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash show 0
 dev1 | 1 +
 1 file changed, 1 insertion(+)
```

Providing additional message:

To provide more context to the stash we create the stash using the following command.

git stash save “message”

Getting back stashed changes:

You can reapply the previously stashed changes with the ‘git stash pop’ or ‘git stash apply’ command.

‘git stash pop’ removes the changes from stash and reapplies the changes in working copy,

‘git stash apply’ do not remove changes .but reapplies the changes in working copy.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash apply
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

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:   dev1

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

Git stash clear:

To drop the stashes use the clear.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash clear
```

The stash is dropped.

Deleting stashes:

To delete a particular stash:

git stash drop stash@{1}

To delete all stashes at once,use the below comman

git stash clear

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash clear

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_testt (main)
$ git stash list
```

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.

Git fetch:

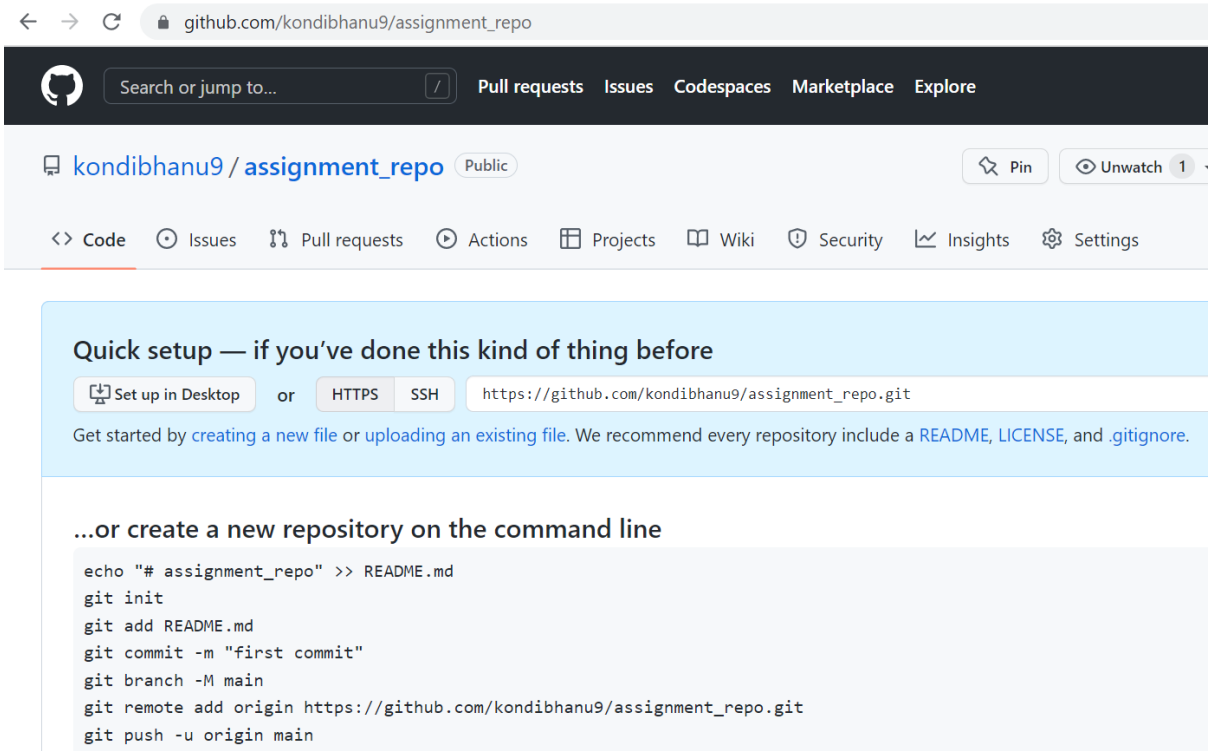
Fetch just downloads the objects and references from a remote repository and normally updated the remote tracking branches.

For example:

Suppose we have made some changes in the remote repository,and we need to bring them to the local repository.In that case we can use the git fetch command which downloads changes and\

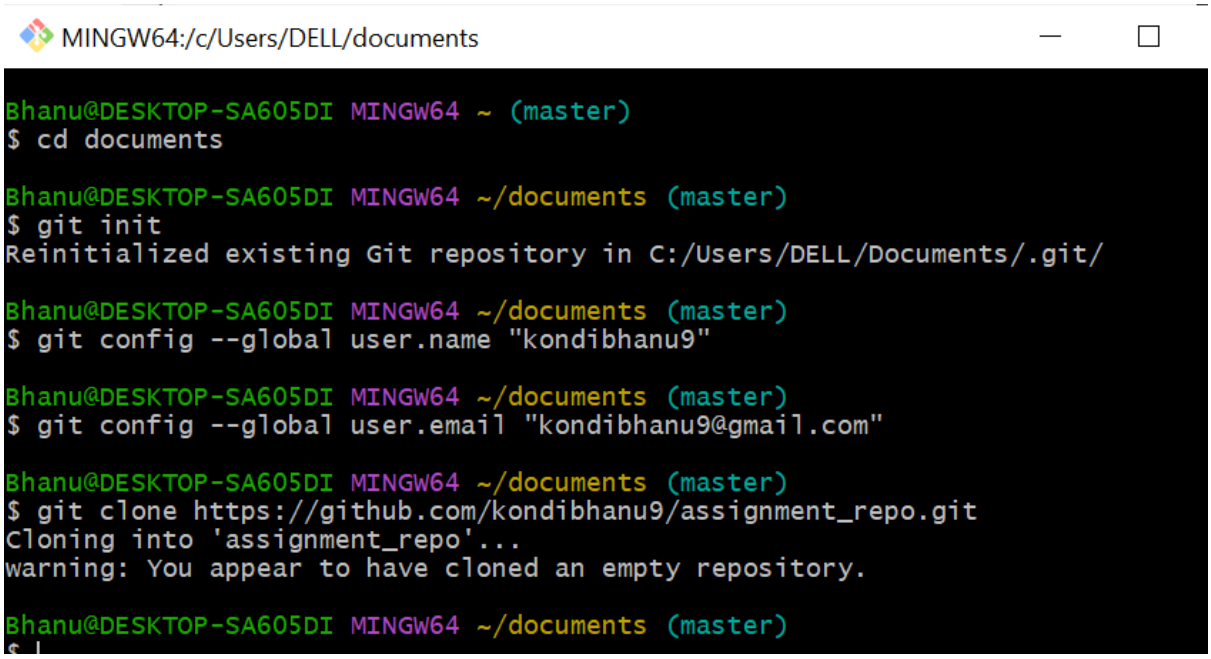
Merge command which merges the changes.

*create a new repository in github.



*clone the repository that you have created above.using git bash

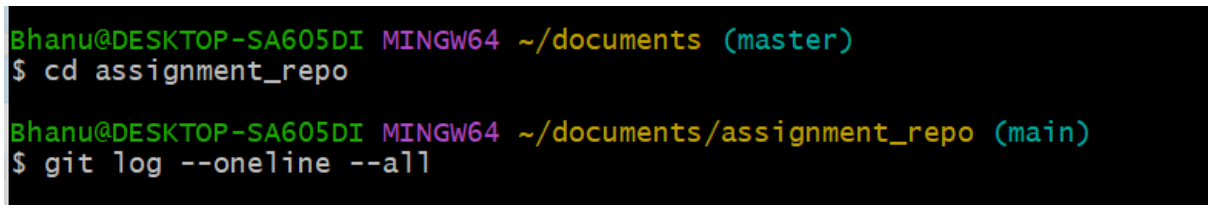
*using the git config give the username and the github mail id globally.



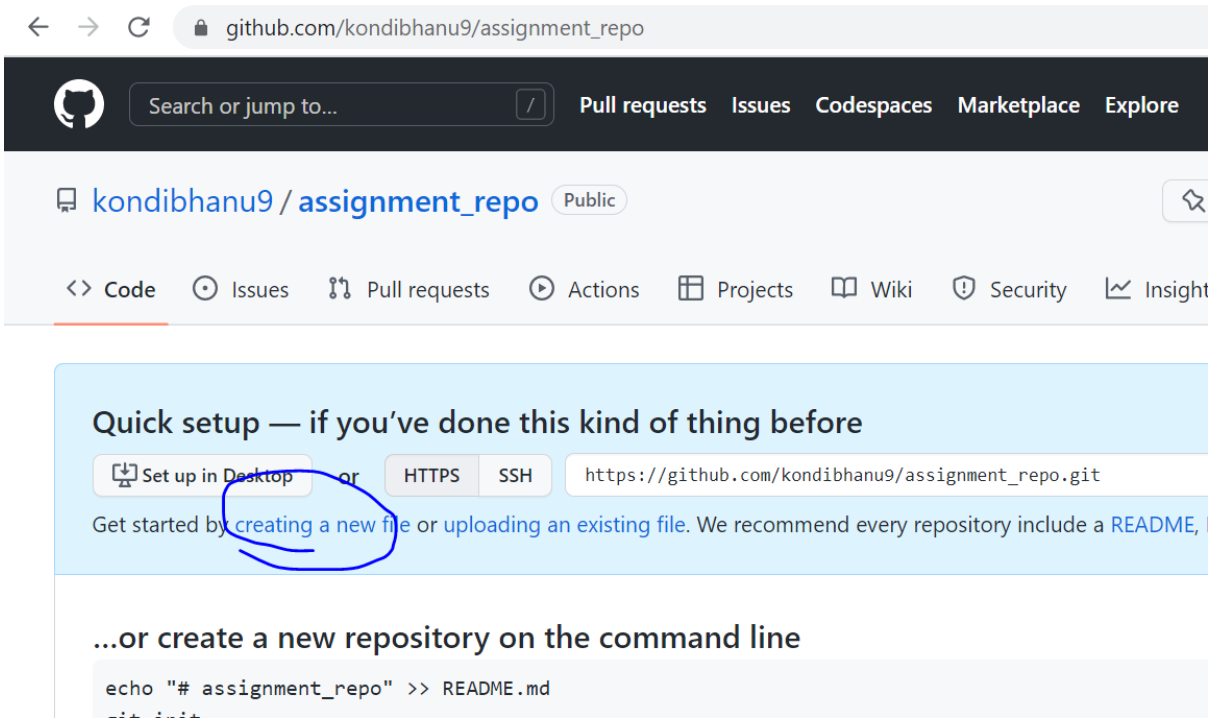
*check if there are any commits present or not in repository.

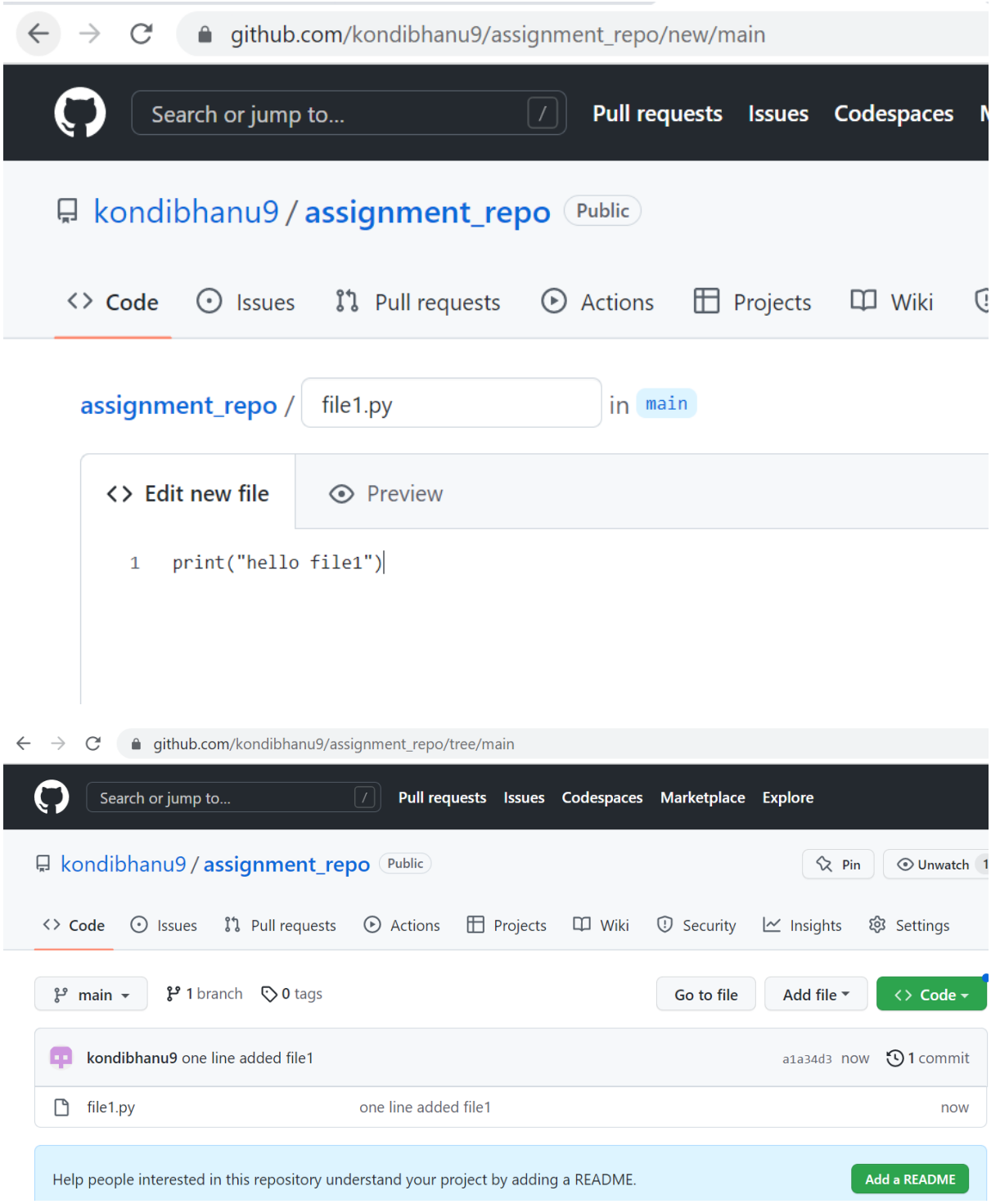
Git log –oneline –all

It is the command used to check what are all the commits that have been made.



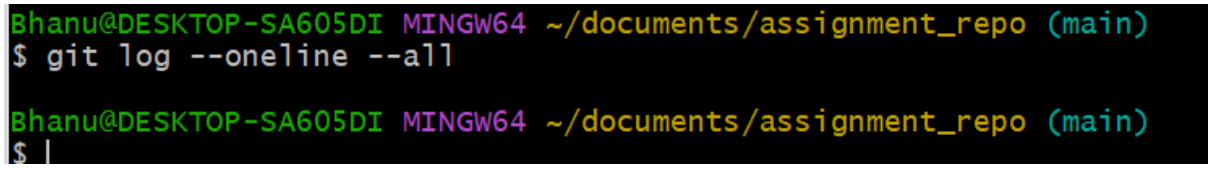
*create a file called file.py in github.





*we have one commit in file.py.the file is committed in the remote repository.

*In git bash,check what are the commits that have been made using git log –oneline –all.



*we cannot find any commits in local repository even though we made one commit in remote repo.

*git fetch is used to download changes from the git remote repository.

*check if we have any commits present after we have fetched the changes.

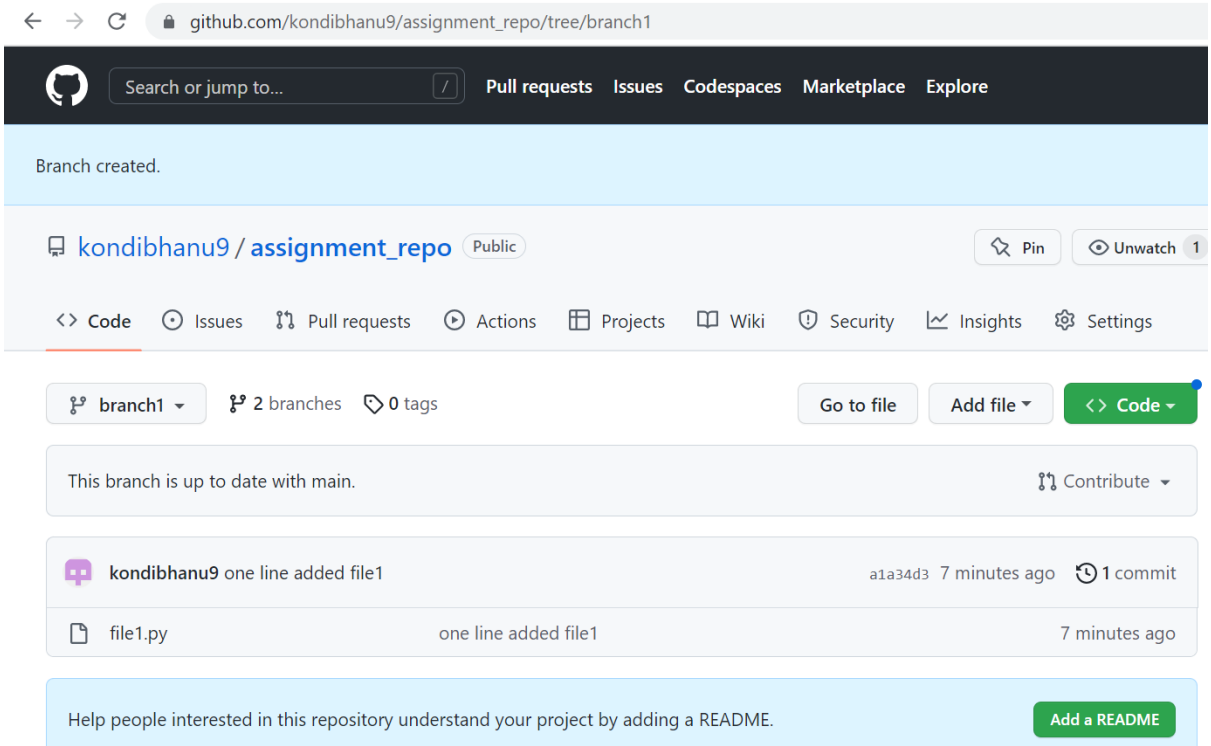
```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 612 bytes | 1024 bytes/s, done.
From https://github.com/kondibhanu9/assignment_repo
 * [new branch]      main       -> origin/main

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git log --oneline --all
a1a34d3 (origin/main) one line added file1

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$
```

Git merging is basically to merge multiple sequences of commits, stored in multiple branches in a unified history or to be simple you can say in a single branch.

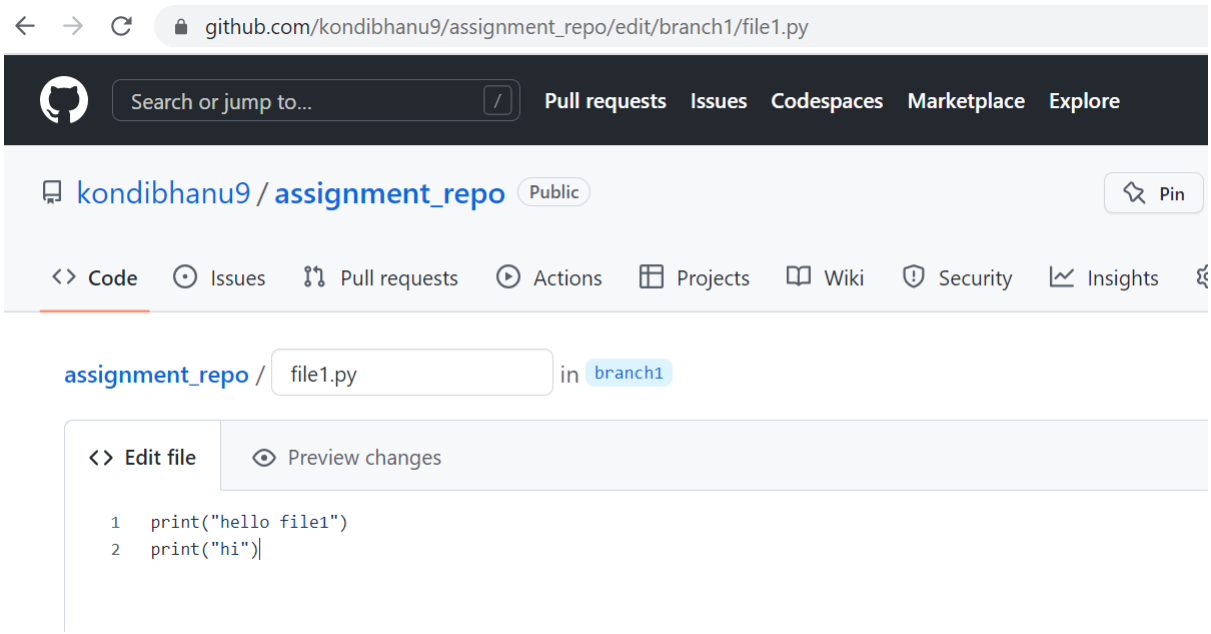
*It can be used to merge the changes that are made in the remote repository to the local repository after fetching the changes.

*in github create a new branch branch1.

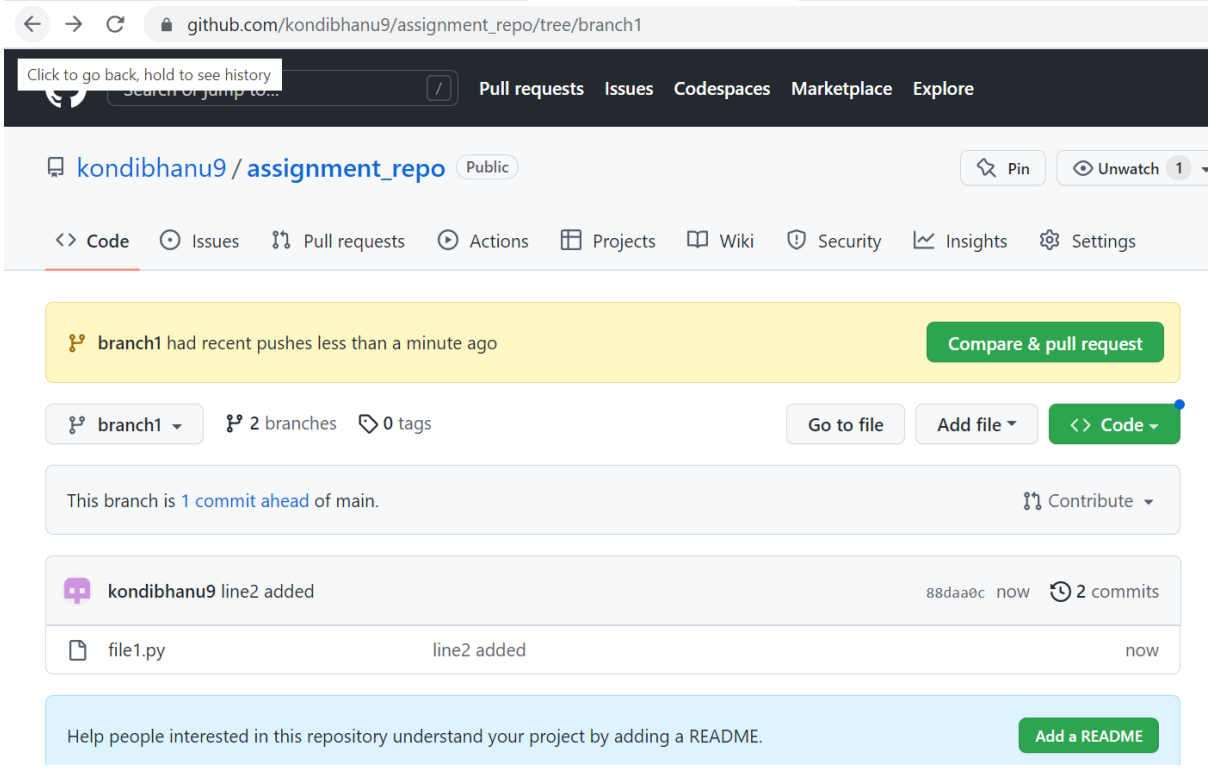
*by default,in branch 1 we can see that there is a file.py



*open the file1.py and add some content.



*commit the changes in branch1.



*use the git fetch command to download the changes in the remote repository.

*it clearly shows that a new branch called branch1 is created and it is modified.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 644 bytes | 4.00 KiB/s, done.
From https://github.com/kondibhanu9/assignment_repo
 * [new branch]      branch1    -> origin/branch1
```

*git log –oneline –all

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git log --oneline --all
88daa0c (origin/branch1) line2 added
a1a34d3 (origin/main) one line added file1
```

*git merge is used to combine the multiple commits into a single commit.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git merge origin/branch1
Updating d99e1ed..d547923
Fast-forward
 one.py | 1 +
 1 file changed, 1 insertion(+)
```

*check whether the single commit has been added or not.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git log --oneline --all
88daa0c (HEAD -> main, origin/branch1) line2 added
a1a34d3 (origin/main) one line added file1

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$
```

*here HEAD->main,origin/branch1 indicated that we have merged the commit from main branch and the branch1 and it appears to be a single commit.

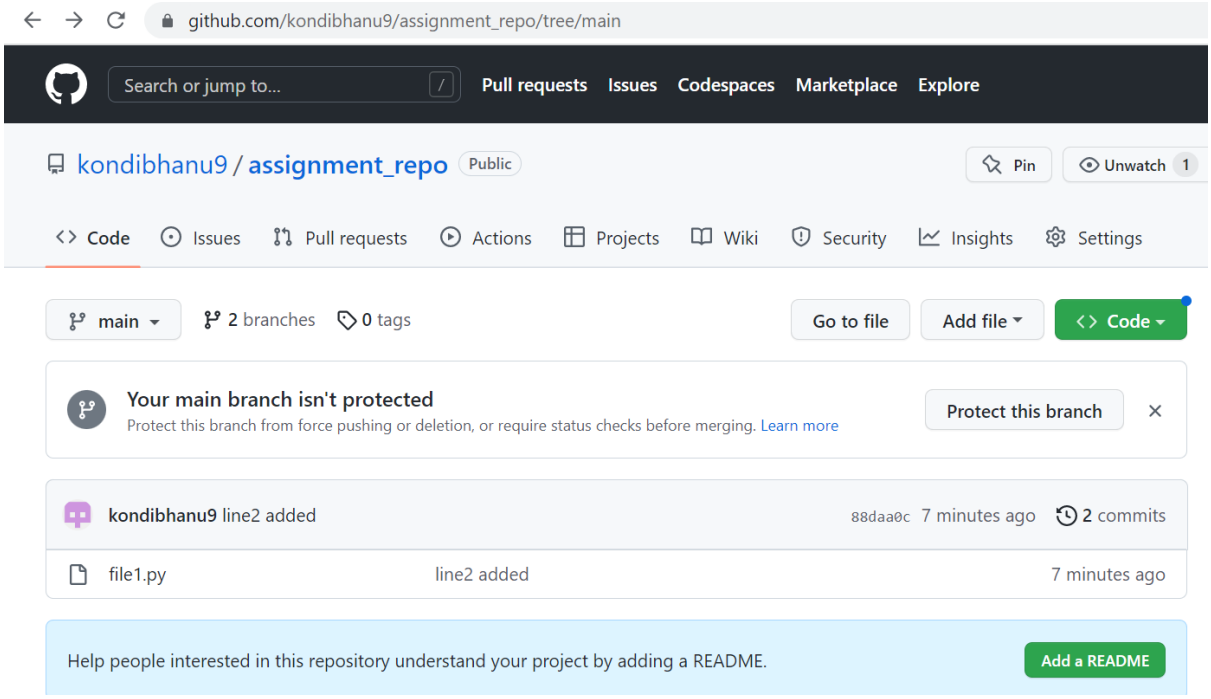
```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git log --oneline --all
88daa0c (HEAD -> main, origin/branch1) line2 added
a1a34d3 (origin/main) one line added file1

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (main)
$ git checkout branch1
Switched to a new branch 'branch1'
branch 'branch1' set up to track 'origin/branch1'.

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (branch1)
$ git log --oneline --all
88daa0c (HEAD -> branch1, origin/branch1, main) line2 added
a1a34d3 (origin/main) one line added file1

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (branch1)
$
```

*now push the changes from the local repository to the remote repository.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (branch1)
$ git remote add org https://github.com/kondibhanu9/assignment_repo.git
```

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (branch1)
$ git push org main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/kondibhanu9/assignment_repo.git
   a1a34d3..88daa0c  main -> main

Bhanu@DESKTOP-SA605DI MINGW64 ~/documents/assignment_repo (branch1)
$
```

*Here when github is open,it shows that,the new commit has been added to the main branch.



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.

Fetch just downloads the objects and references from a remote repository and normally updated the remote tracking branches.

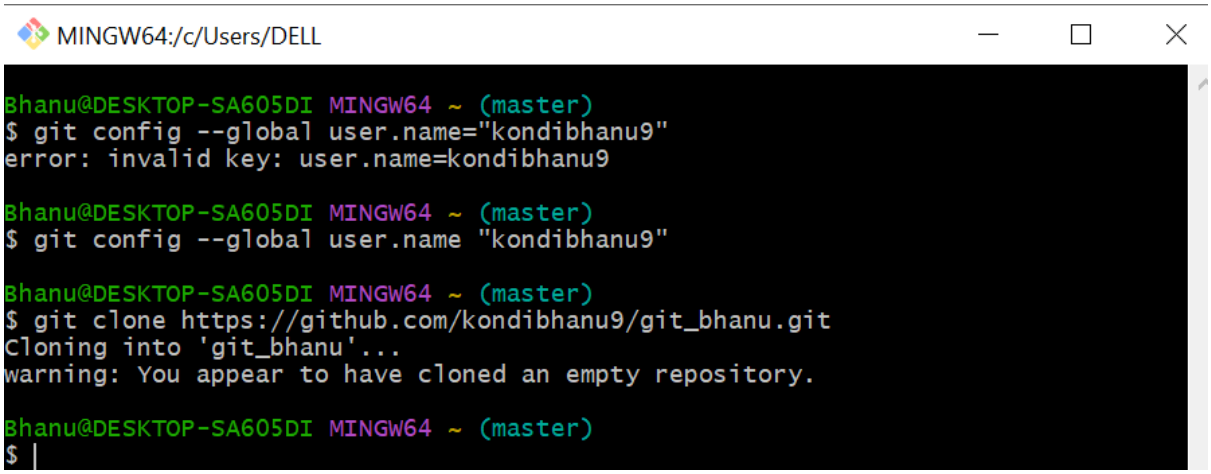
Pull,however will not only download the changes ,but also merges them-it is the combination of fetch and merge.The configured remote tracking branch is selected automatically.

Git pull=git fetch+git merge

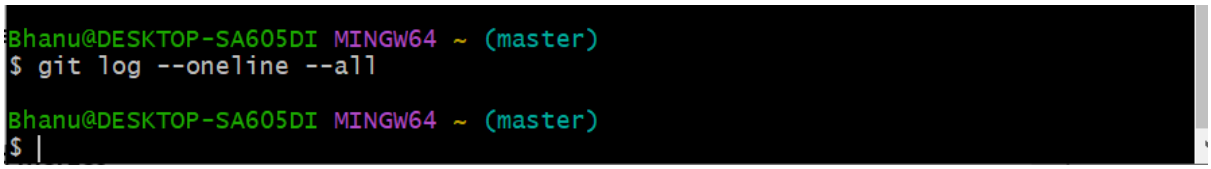
Step 1)Git clone an empty repository

First give the username and email using git config

Then create a new repository named git_bhanu and clone using git clone command

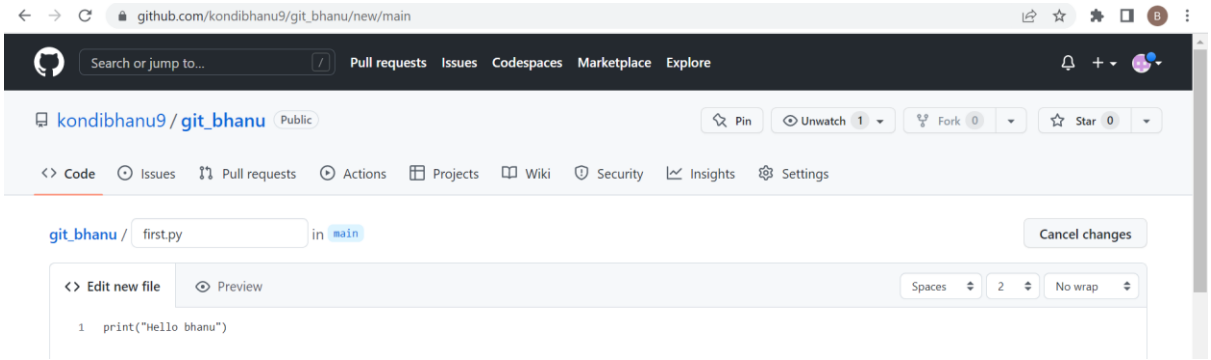


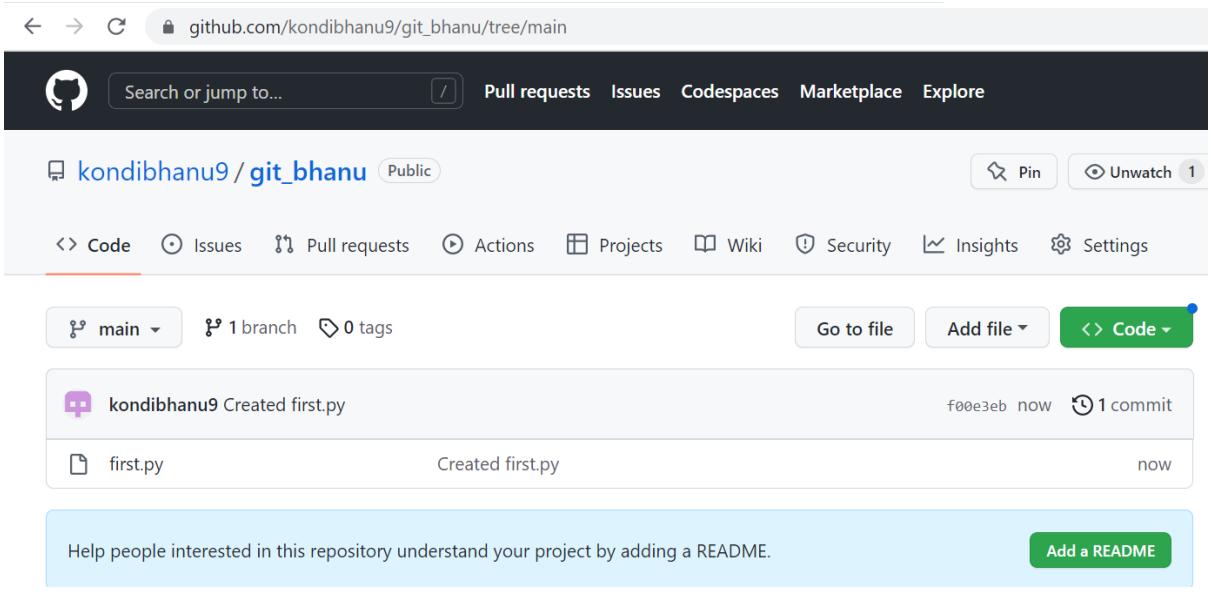
Step 2)Git log –oneline --all



This command gives no ouput as we have not commited anything yet.

Go to github and go to the repository git_bhanu and create a file and then update it.





Step 4)In the git bash ,go to the git_bhanu repository .

```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ cd git_bhanu

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_bhanu (main)
$ git log --oneline --all

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_bhanu (main)
$
```

We have moved to the git_bhanu repository and use the git log command to view the commit history,but here we cannot see any commits ,even though we made one commit in the remote repository.

Step 5)Git fetch

Git fetch will download the changes in remote repository,fetch commands just downloads the changes made in the remote repository.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~/git_bhanu (main)
$ git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 605 bytes | 1024 bytes/s, done.
From https://github.com/kondibhanu9/git_bhanu
 * [new branch]      main       -> origin/main

Bhanu@DESKTOP-SA605DI MINGW64 ~/git_bhanu (main)
$
```

Once the git fetch command is used,it downloads the changes made in the remote repository.

Git log –oneline –all

```
Bhanu@DESKTOP-SA6O5DI MINGW64 ~/git_bhanu (main)
$ git log --oneline --all
f00e3eb (origin/main) Created first.py

Bhanu@DESKTOP-SA6O5DI MINGW64 ~/git_bhanu (main)
$
```

Here when git log command is used then,it shows one commit made in the remote repository.

Here one commit that is made in the remote repository is being shown,but it is not applied.To apply and download we use the git pull command.

Git pull------downloads and merges

```
Bhanu@DESKTOP-SA6O5DI MINGW64 ~/git_bhanu (main)
$ git log --oneline --all
f00e3eb (origin/main) Created first.py

Bhanu@DESKTOP-SA6O5DI MINGW64 ~/git_bhanu (main)
$ git pull

Bhanu@DESKTOP-SA6O5DI MINGW64 ~/git_bhanu (main)
$ git log --oneline
f00e3eb (HEAD -> main, origin/main) Created first.py
```

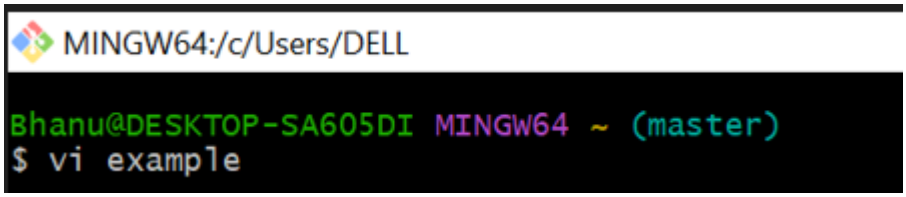
Now the commits are applied to the main branch by using the git pull command.

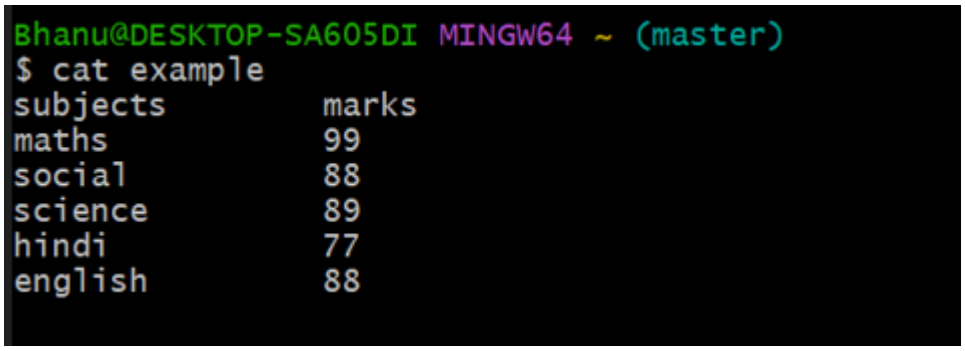
Q4. Try to find out about the awk command and use it while reading a file created by yourself. Also, make a bash script file and try to find out the prime number from the range 1 to 20.

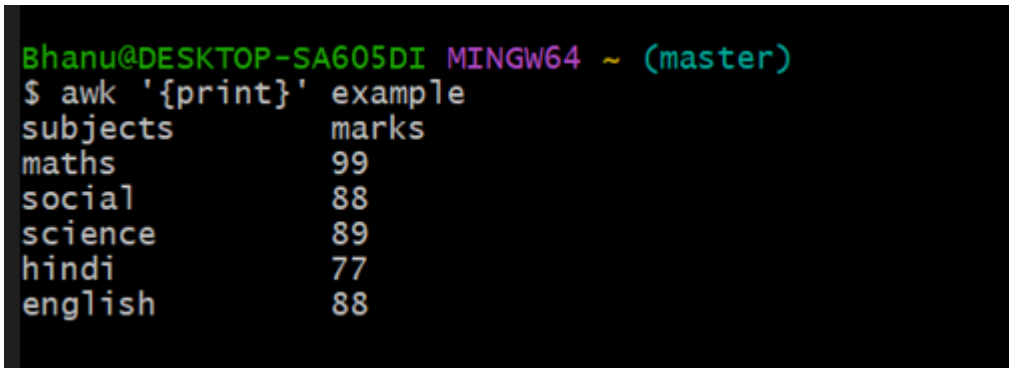
Awk command is a scripting language used for manipulating data and generating reports.

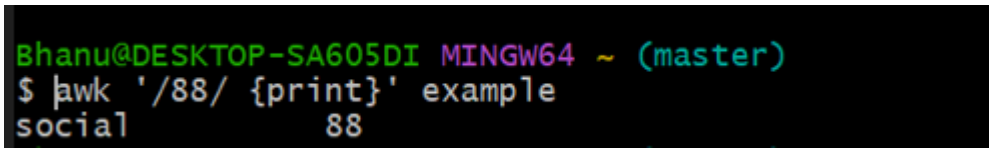
The awk command programming language requires no compiling and allows the user to use variables,numeric functions,string functions,and logical operators.

*create a file called example

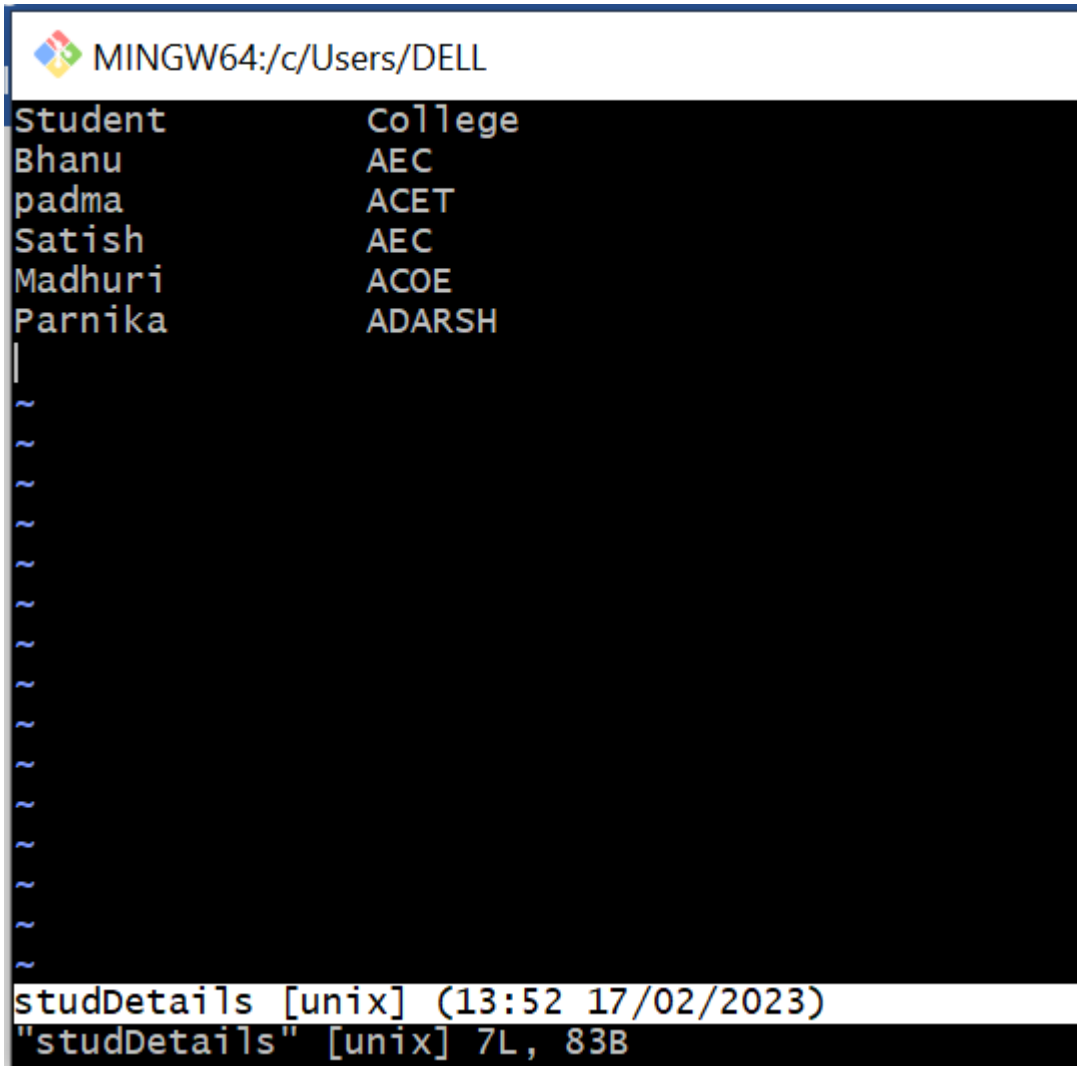








```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ awk '{ print "this is an awk command"}'

this is an awk command

this is an awk command

this is an awk command

[1]+  Stopped                 awk '{ print "this is an awk command"}'

Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ vi studDetails
```



*print the studDetails using awk command.

*we can also print the details by filtering .

```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ awk '{print}' studDetails
Student         College
Bhanu           AEC
padma           ACET
Satish          AEC
Madhuri         ACOE
Parnika         ADARSH


Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ awk '/AEC/ {print}' studDetails
Bhanu           AEC
Satish          AEC

Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$
```

*$1 indicates to print the first column elements.

The awk action is enclosed in braces ({}) and consists of statements.Each statement specifies the Operation to be performed.An action can have more than one statement separated by newline or Semi colons .If the rule has no action,it defaults to printing the whole record.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ awk '{print $1}' studDetails
Student
Bhanu
padma
Satish
Madhuri
Parnika
```

```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ awk '{print $NF}' studDetails
College
AEC
ACET
AEC
ACOE
ADARSH
```

Steps to follow bash scripting:

Step1) create the file with extension .sh.

Step 2)open the shell and write the script.

Step 3)we can give the permissions of read, write and execute.

Step 4)save the code and run the code .

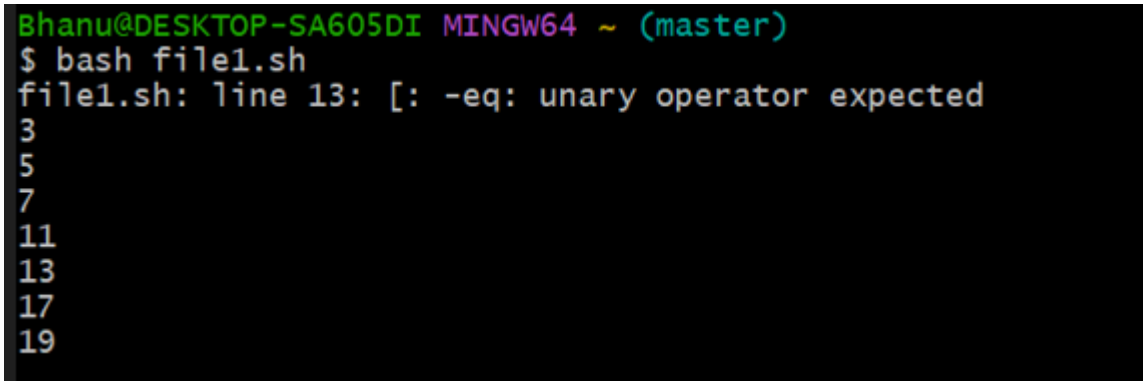
To run the run a code

Syntax: ./filename.

```
Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ vi file1.sh
```

MINGW64:/c/Users/DELL

```
for((i=2;i<=20;))
do
        for((j=i-1;j>=2;))
        do
                if [ `expr $i % $j` -ne 0 ] ; then
                        prime=1
                else
                        prime=0
                        break
                fi
                j=`expr $j - 1`
        done
        if [ $prime -eq 1 ] ; then
                echo $i
        fi
        i=`expr $i + 1`
done

~
~
```


Bhanu@DESKTOP-SA605DI MINGW64 ~ (master)
$ bash file1.sh
file1.sh: line 13: [: -eq: unary operator expected
3
5
7
11
13
17
19

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.
All the processes pertaining to this should be provided in a screenshot for grading.
Steps for setting up a container and run a Ubuntu os
Docker client when runs the command ,then it means that it is communicating with docker daemon and pulls the image from docker hub.
*First step is to download the image of ubuntu ,the pull command is used to perform this action.

Syntax : docker pull ubuntu

Step 2)Once the ubuntu image is being downloaded,then we are ready to run the image.

Syntax: docker run -it ubuntu
Here -it option runs the container in an interactive mode and opens up a shell within the ubuntu os.

**to know about the updates or update the image we can use the apt update command.
Syntax:apt update.

Image: Images are used to create containers. It uses a private container registry to share container images within the enterprise and also use public container registry to share container images with whole world.
Container: Containers are used to hold the entire package that is needed to run the application. We can say that the image is a template and the container is a copy of the template.
*These are the containers present in the docker desktop.

*check the version of docker using the
Docker version command.

```
C:\Users\Bhanu>docker version
Client:
 Cloud integration: v1.0.29
 Version:           20.10.22
 API version:       1.41
 Go version:        go1.18.9
 Git commit:        3a2c30b
 Built:             Thu Dec 15 22:36:18 2022
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Desktop 4.16.3 (96739)
 Engine:
  Version:          20.10.22
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.9
  Git commit:       42c8b31
  Built:            Thu Dec 15 22:26:14 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.14
  GitCommit:        9ba4b250366a5ddde94bb7c9d1def331423aa323
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

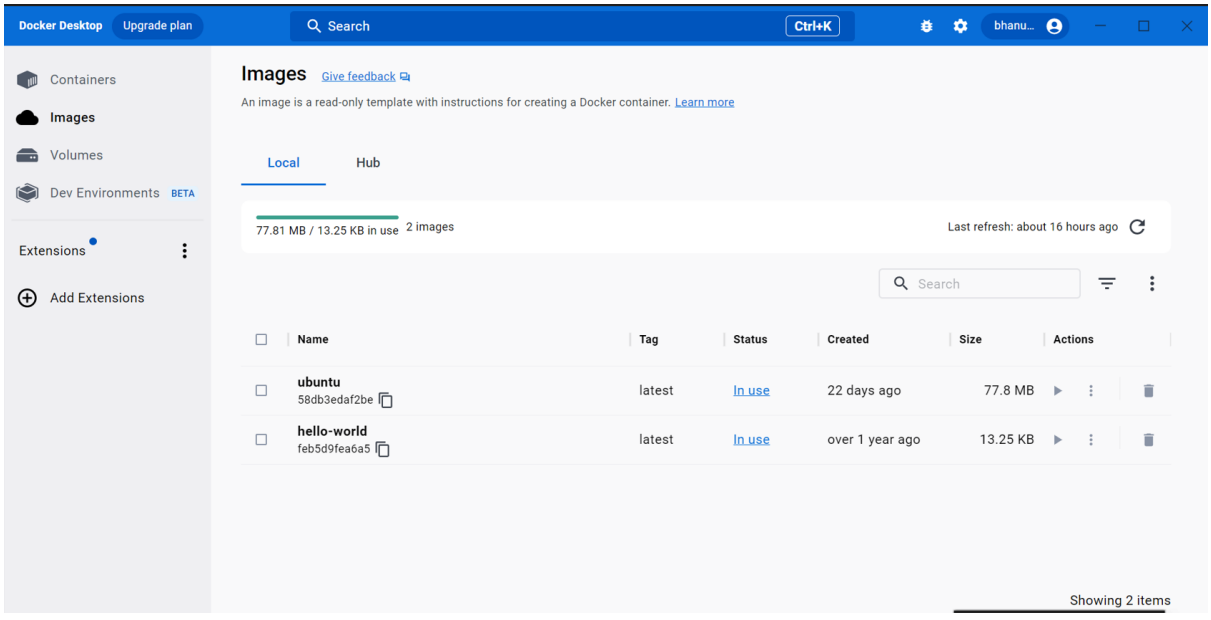
*use the docker pull ubuntu

It is used to pull the ubuntu image from the docker hub.

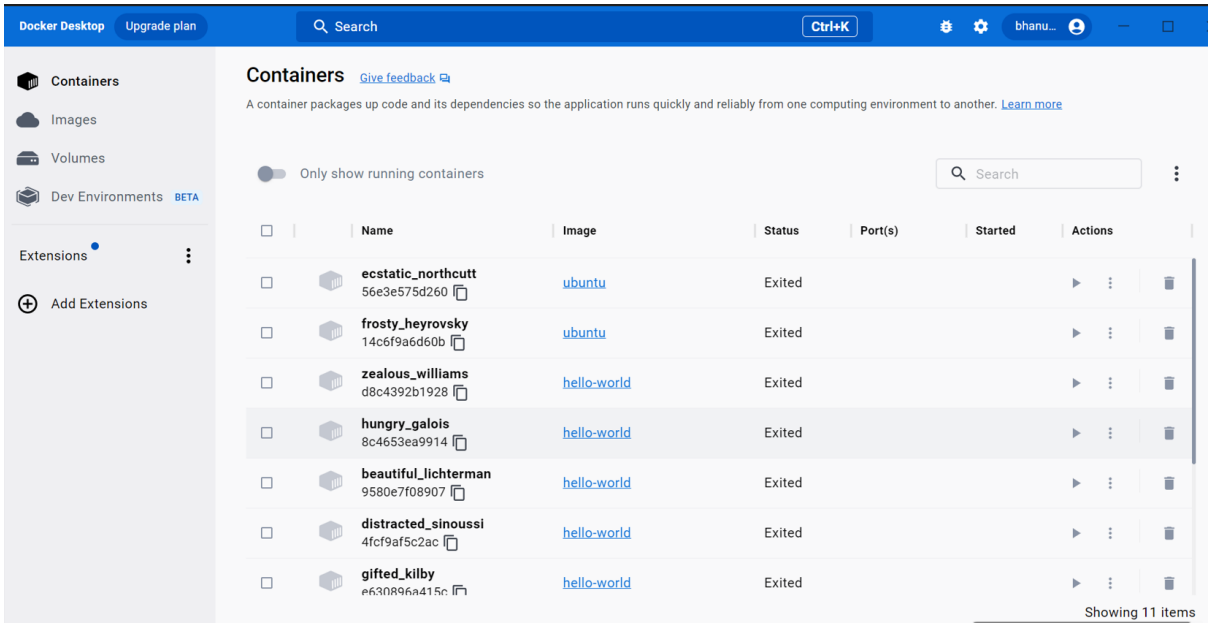
```
root@76829b3f38d3: /
  GitCommit:        de40ad0

C:\Users\Bhanu>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
```

```
root@76829b3f38d3: /
  GitCommit:        de40ad0

C:\Users\Bhanu>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

C:\Users\Bhanu>docker run -it ubuntu
root@76829b3f38d3:/# apt update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [5557 B]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [860 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [752 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [107 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [807 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [10.9 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [808 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1091 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1136 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [49.0 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [22.4 kB]
Fetched 25.8 MB in 1min 36s (270 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
5 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@76829b3f38d3:/#
```

*now open the docker desktop and check whether you got the ubuntu image or not.



*it shows that ubuntu image is being downloaded successfully.



*go to containers and check whether the ubuntu container is there or it,

*it shows that ubuntu container is downloaded.