DevOps Assignment -1

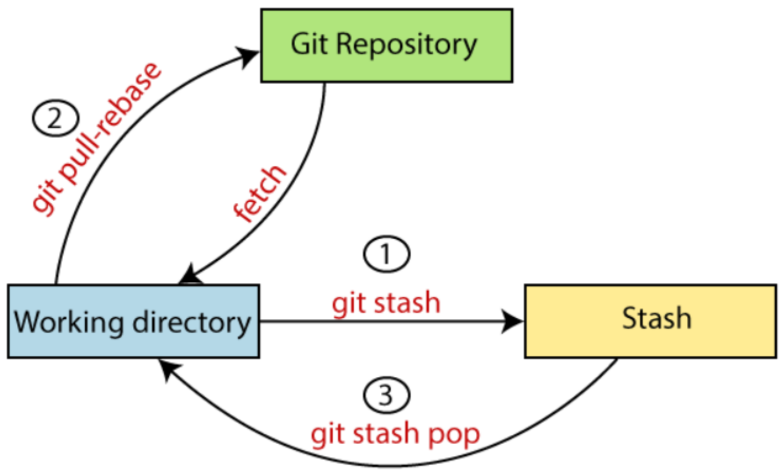
1.Describe the usage of stash command by using an example and also state the process by giving the screenshot of all the commands written in git bash.

Ans:

Git Stash:
Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branches without committing the current branch.

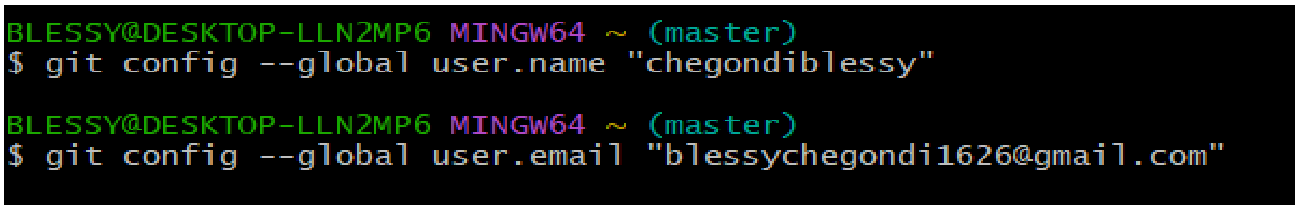
- The git stash command saves a copy of your uncommitted changes in a queue, off to the side of your project.
- By uncommitted changes, I mean items in either the staging area or the working directory that have been modified but not committed to the local repository.
- Each time the stash command is invoked and there is uncommitted content (since the last stash command), git creates a new element on the queue to save that content. That content can be in the staging area, in the working directory, or both.
- After creating the stash and saving the uncommitted content, Git is basically doing a git reset --hard HEAD operation. However, because you have the stash, you haven't lost your uncommitted changes.

The below figure demonstrates the properties and role of stashing concerning repository and working directory:

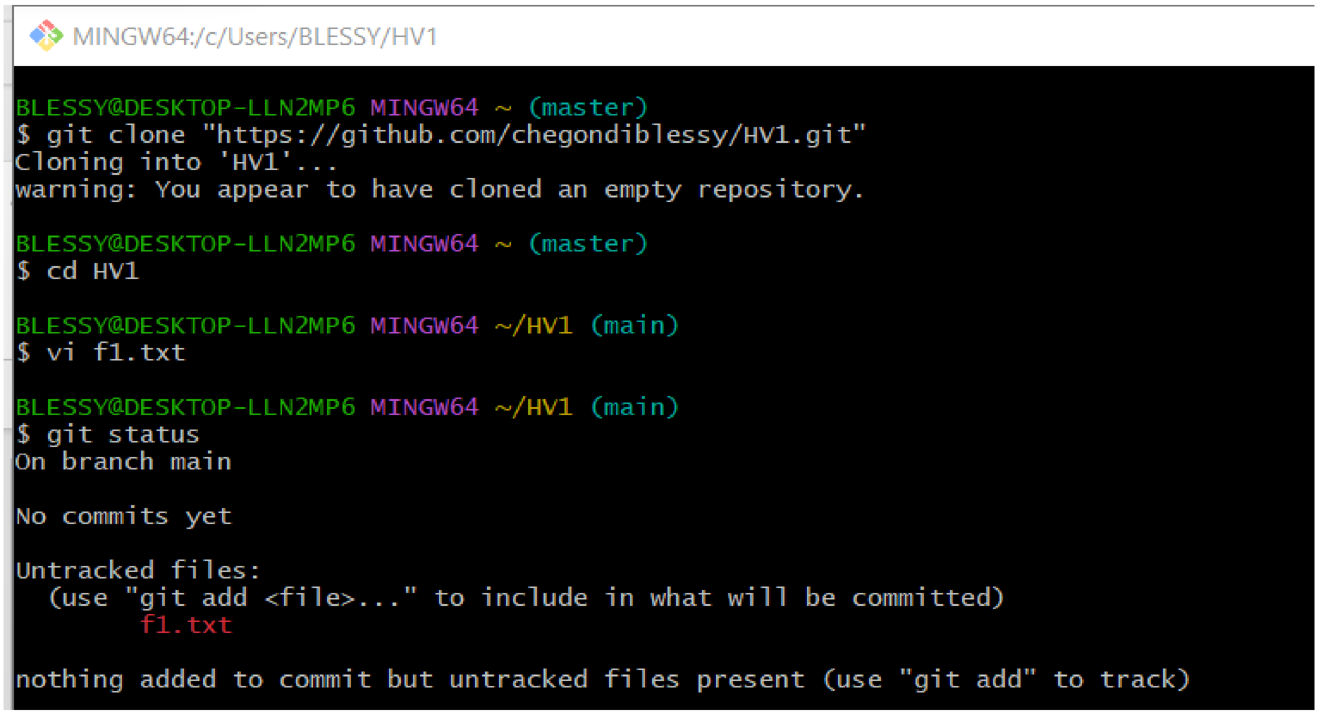


Many options are available with git stash. Some useful options are given below:

- **Git stash**
- **Git stash save**
- **Git stash list**
- **Git stash apply**
- **Git stash changes**
- **Git stash pop**
- **Git stash drop**
- **Git stash clear**
- **Git stash branch**

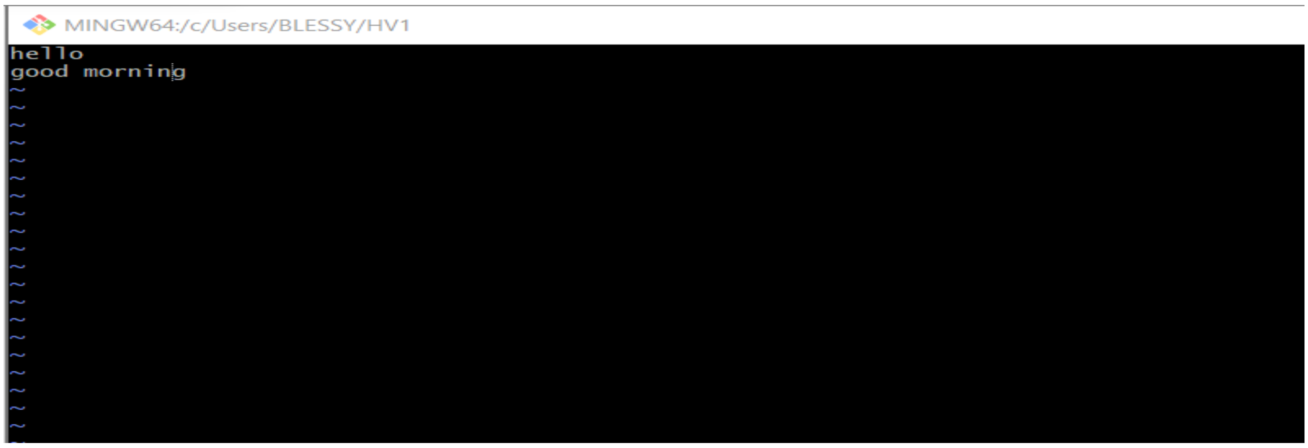


```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ git config --global user.name "chegondiblessy"

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ git config --global user.email "blessychegondi1626@gmail.com"
```



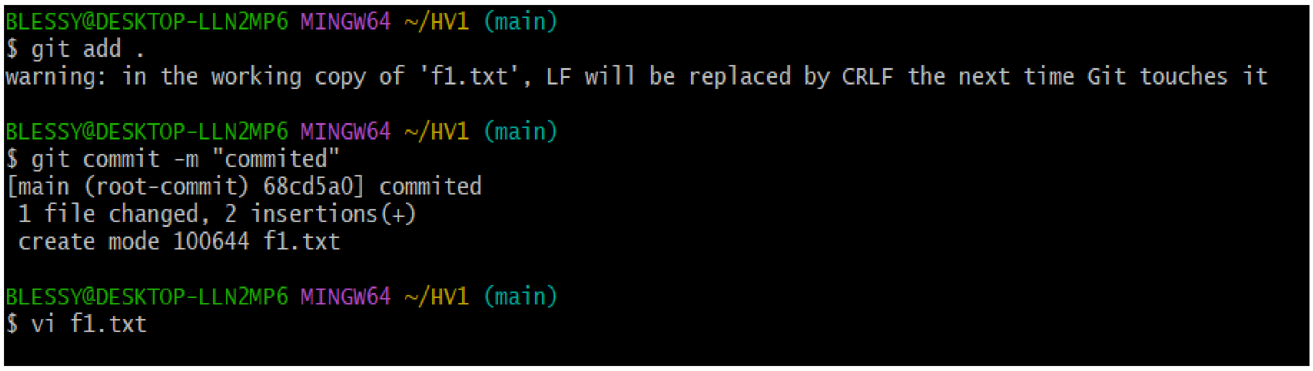
MINGW64:/c/Users/BLESSY/HV1

```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ git clone "https://github.com/chegondiblessy/HV1.git"
Cloning into 'HV1'...
warning: You appear to have cloned an empty repository.

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ cd HV1

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ vi f1.txt

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f1.txt

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

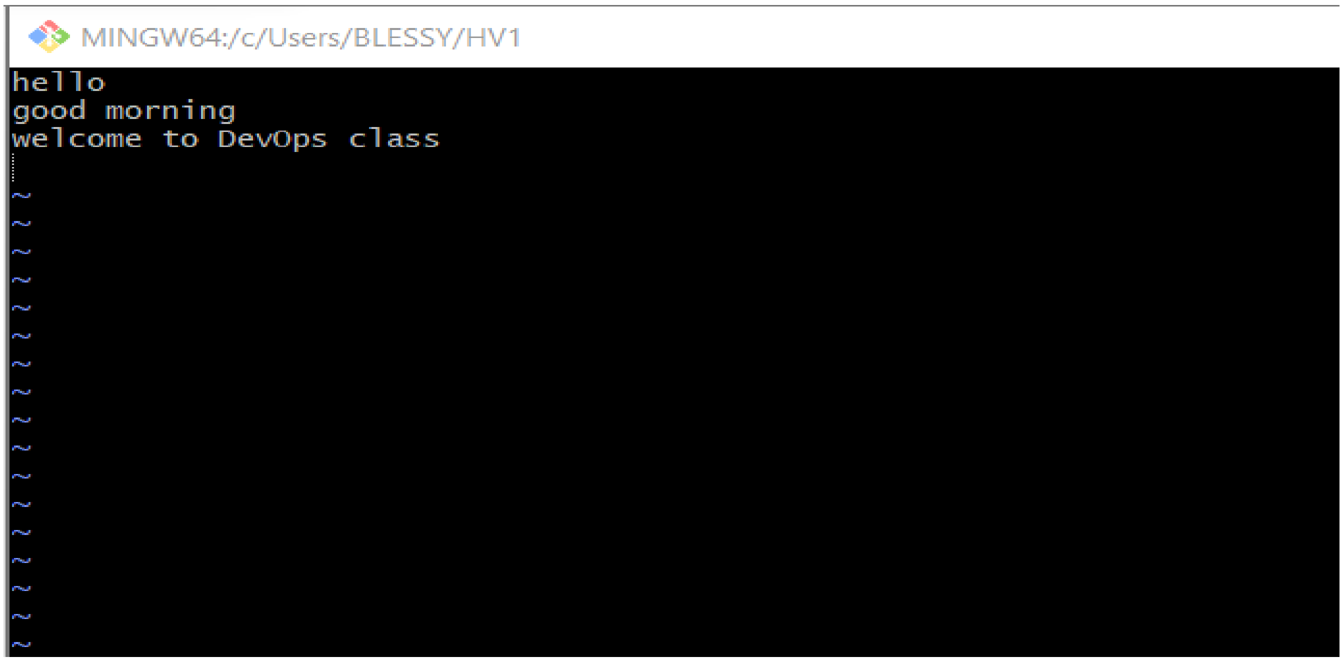


MINGW64:/c/Users/BLESSY/HV1

```
hello
good morning
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

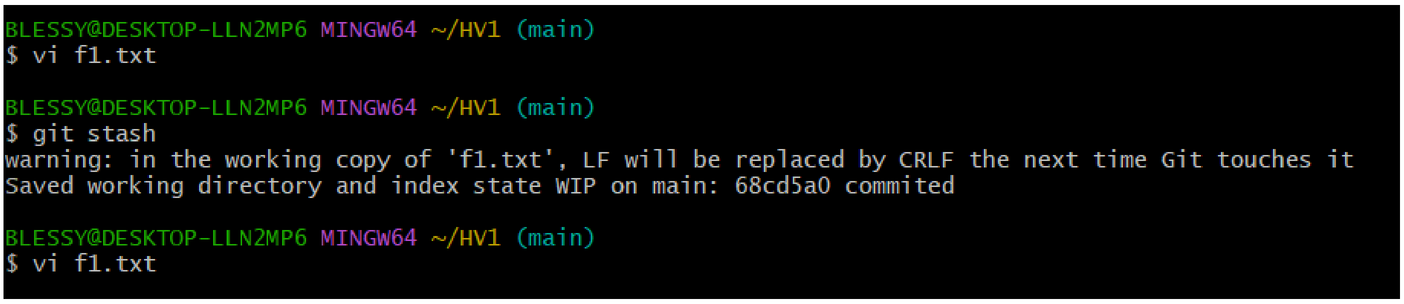


```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git add .
warning: in the working copy of 'f1.txt', LF will be replaced by CRLF the next time Git touches it

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git commit -m "commited"
[main (root-commit) 68cd5a0] commited
 1 file changed, 2 insertions(+)
 create mode 100644 f1.txt

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ vi f1.txt
```



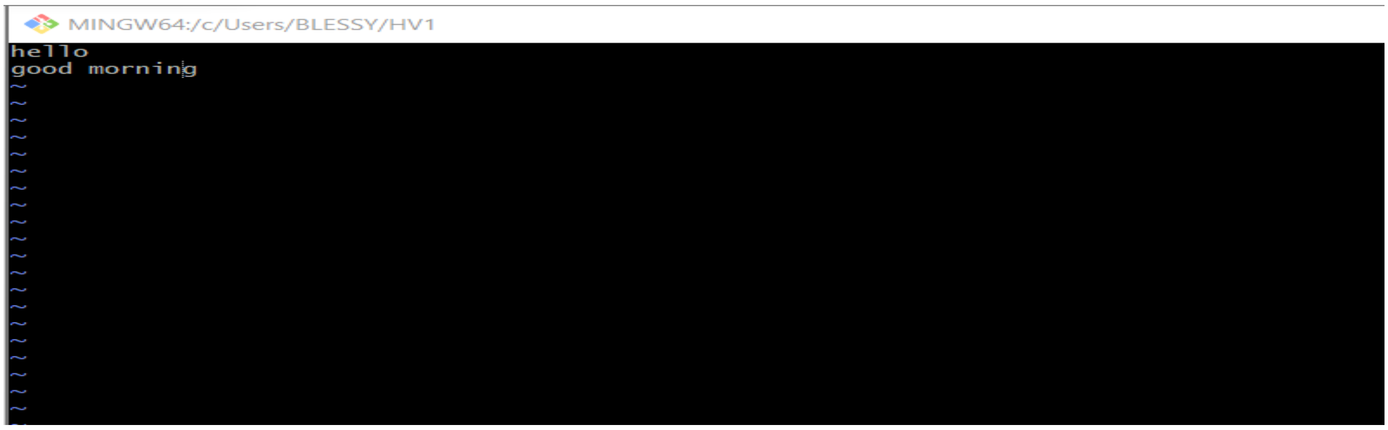
```
MINGW64:/c/Users/BLESSY/HV1
hello
good morning
welcome to DevOps class

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

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.

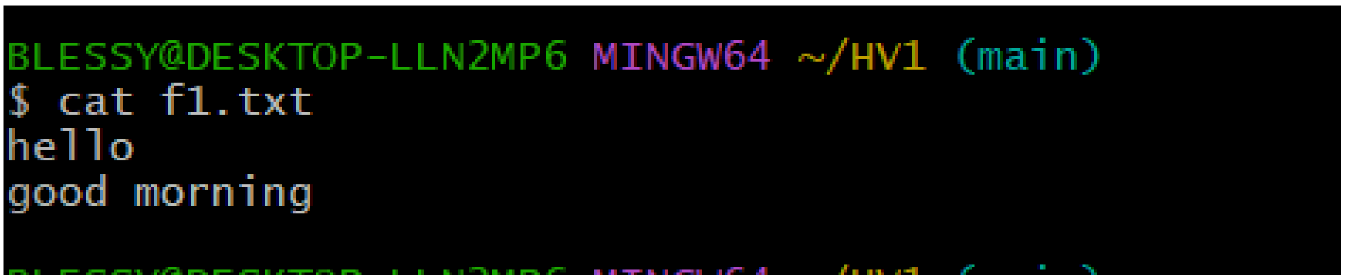


```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ vi f1.txt

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash
warning: in the working copy of 'f1.txt', LF will be replaced by CRLF the next time Git touches it
Saved working directory and index state WIP on main: 68cd5a0 commited

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ vi f1.txt
```

Now the changes are removed

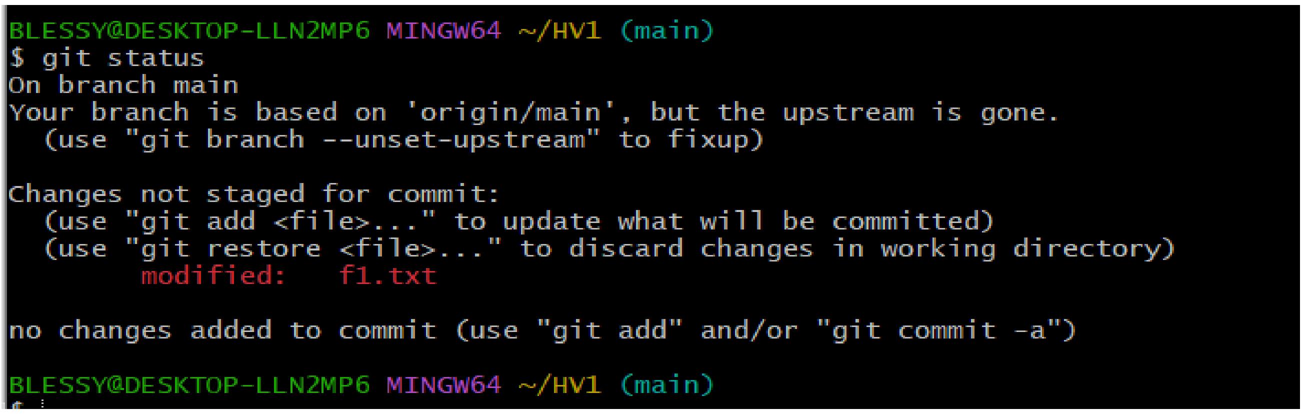


```
MINGW64:/c/Users/BLESSY/HV1
hello
good morning
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```



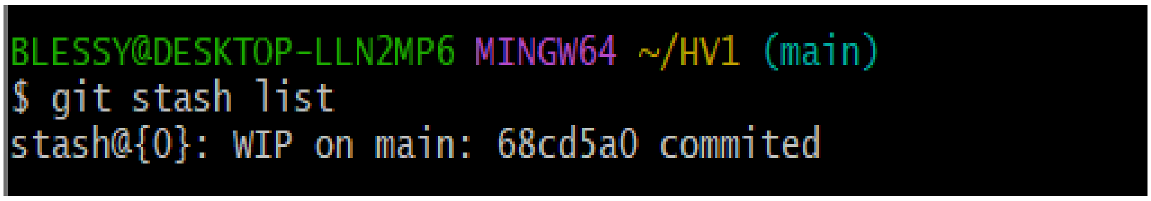
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ cat f1.txt
hello
good morning
```

The file is now stashed and it is under untracked state.
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.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (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:   f1.txt

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

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
```

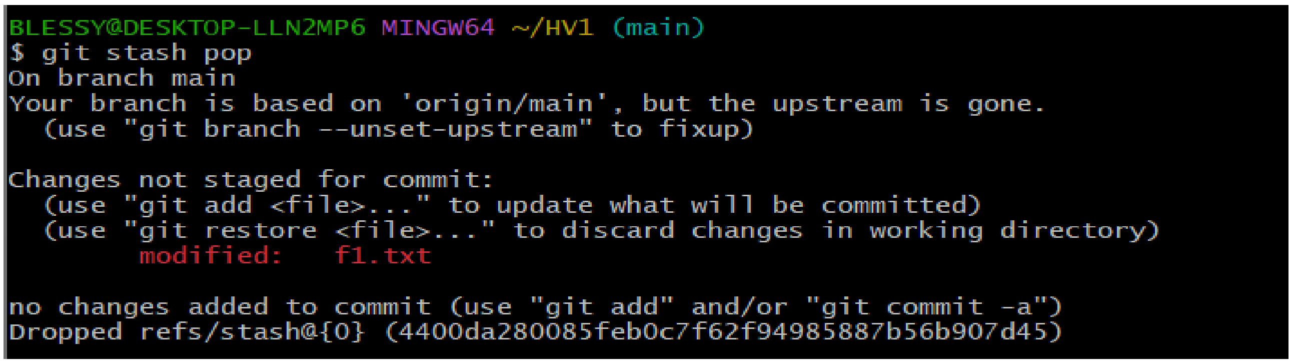
Listing stashes:
You can create multiple slashes and view them using git stash list command.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash list
stash@{0}: WIP on main: 68cd5a0 commited
```

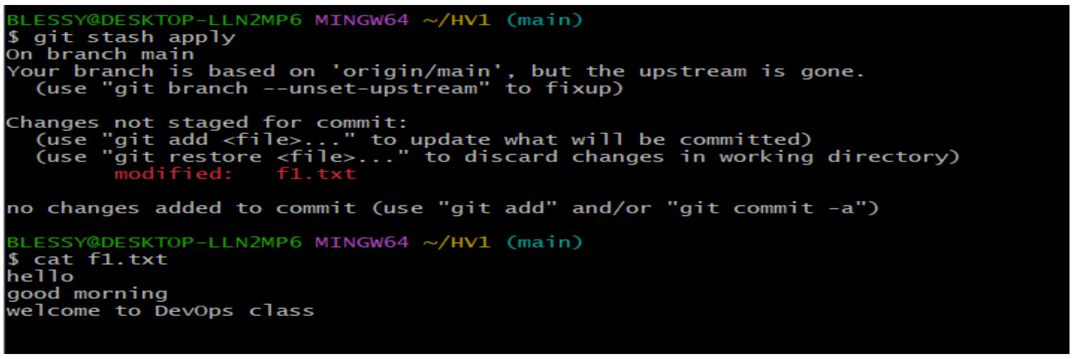
Providing additional message:
To provide more context to the stash we create the stash using the following command.
git stash save “message”

Getting back stashed changes:
You can reapply the previously stashed changes with the ‘git stash pop’ or ‘git stash apply’ command.
1.‘git stash pop’ removes the changes from stash and reapplies the changes in working copy,
2.‘git stash apply’ do not remove changes .but reapplies the changes in working copy.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash pop
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:   f1.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (4400da280085feb0c7f62f94985887b56b907d45)
```

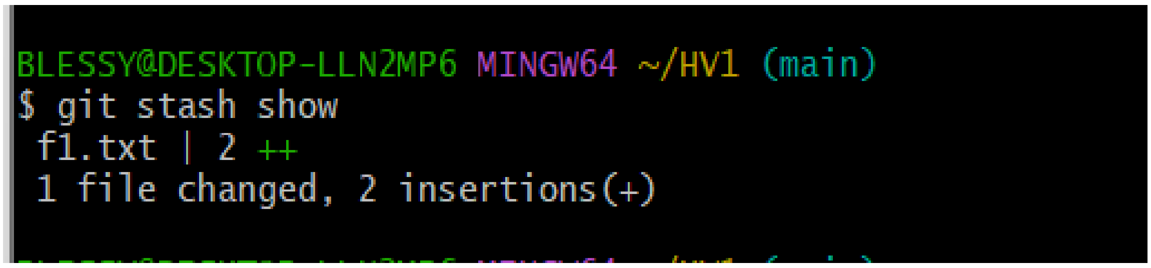
By using “git stash apply” We got the previous uncommitted changes.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (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:   f1.txt

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

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ cat f1.txt
hello
good morning
welcome to DevOps class
```

To view the stash summary:

Git stash show is used to view the summary



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash show
 f1.txt | 2 ++
 1 file changed, 2 insertions(+)
```

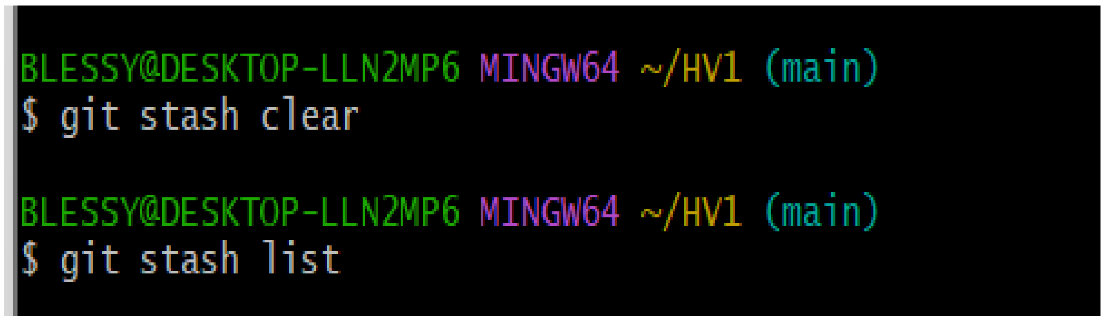
Deleting stashes:

To delete a particular stash:

    git stash drop stash@{1}

To delete all stashes at once,use the below comman

git stash clear



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash clear

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV1 (main)
$ git stash list
```

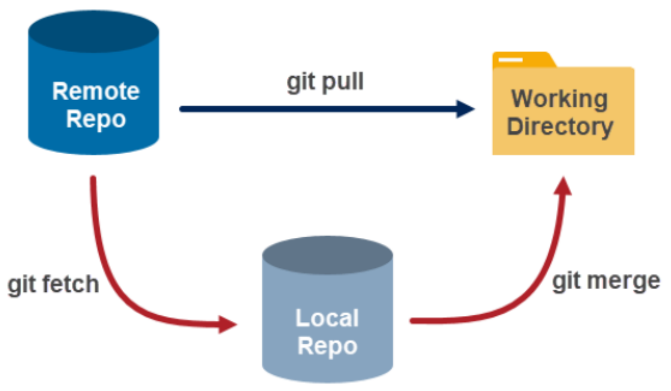
2.By using a sample example of your choice ,use the git fetch command and also use the git merge command and also describe the whole process through a screenshot with all the commands and their output in git bash.

Ans:

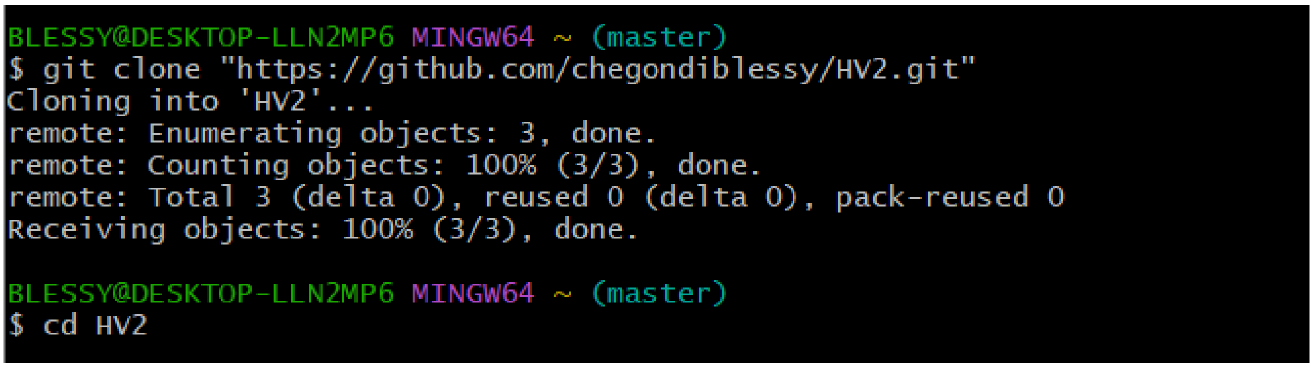
Git Fetch:
The **git fetch** command downloads objects to the local machine without overwriting existing local code in the current branch. The command pulls a record of remote repository changes, allowing insight into progress history before adjustments.
The **git fetch** command retrieves commits, files, branches, and tags from a remote repository. The general syntax for command is:

git fetch <options> <remote name> <branch name>

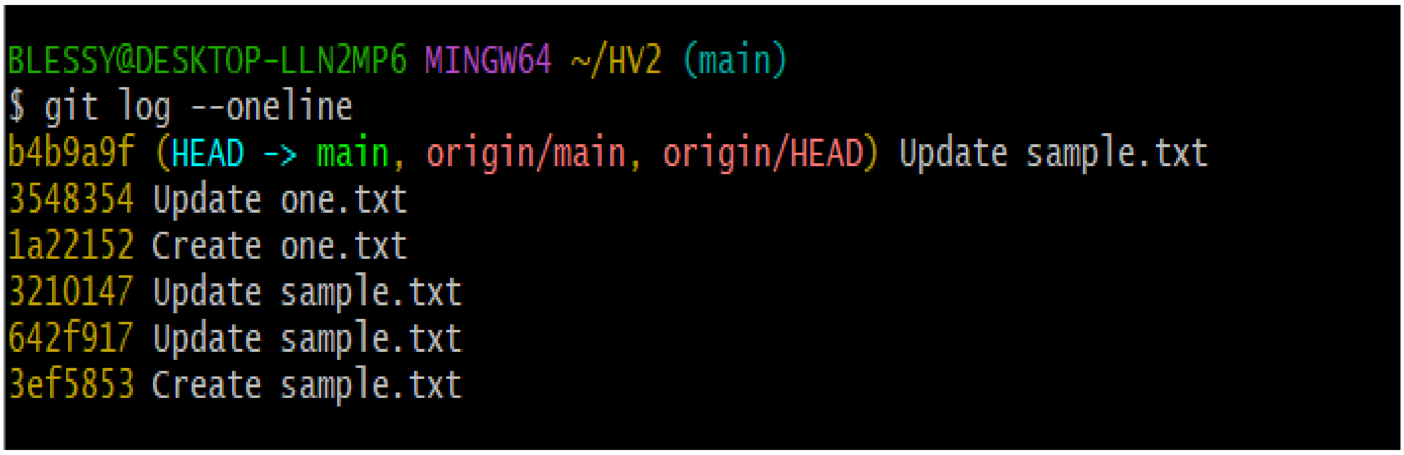
- The **git fetch** command gets all the changes from a remote repository. The fetched metadata resides in the *.git* directory, while the working directory stays unaltered.
- Effectively, **git fetch** retrieves the metadata without applying changes locally. The git pull command combines **git fetch** and git merge functions into one.



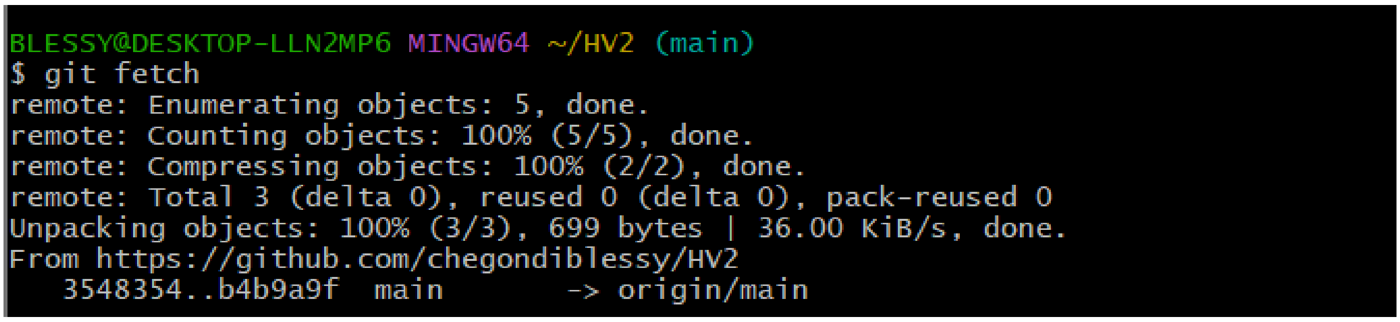
- Since the working directory state remains unaffected, the fetched contents must be checked out with the git checkout command or merged with git merge.
- However, since joining contents is a manual process, **git fetch** allows reviewing code before changing anything. The review process helps avoid merge conflicts.



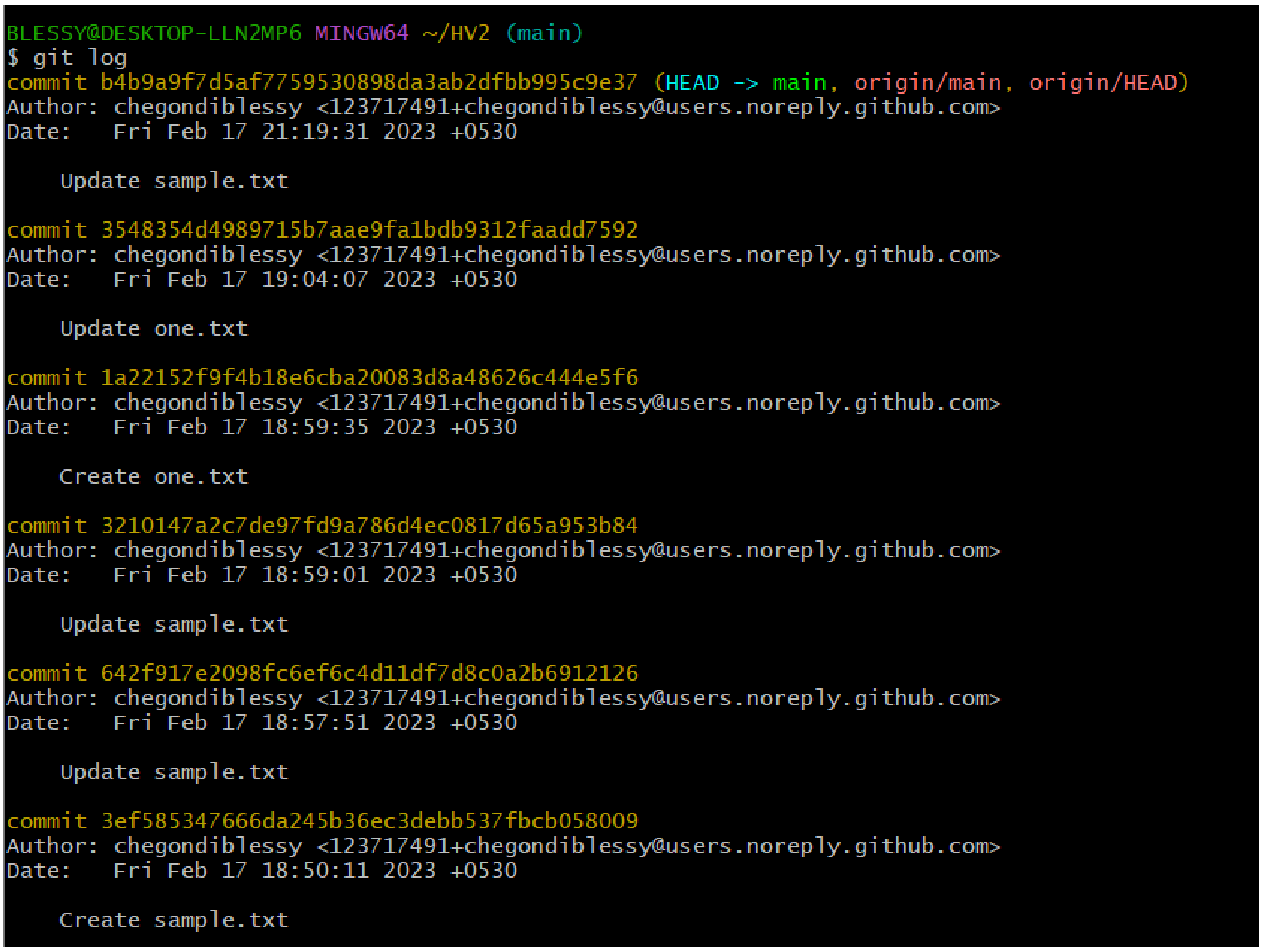
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ git clone "https://github.com/chegondiblessy/HV2.git"
Cloning into 'HV2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ cd HV2
```



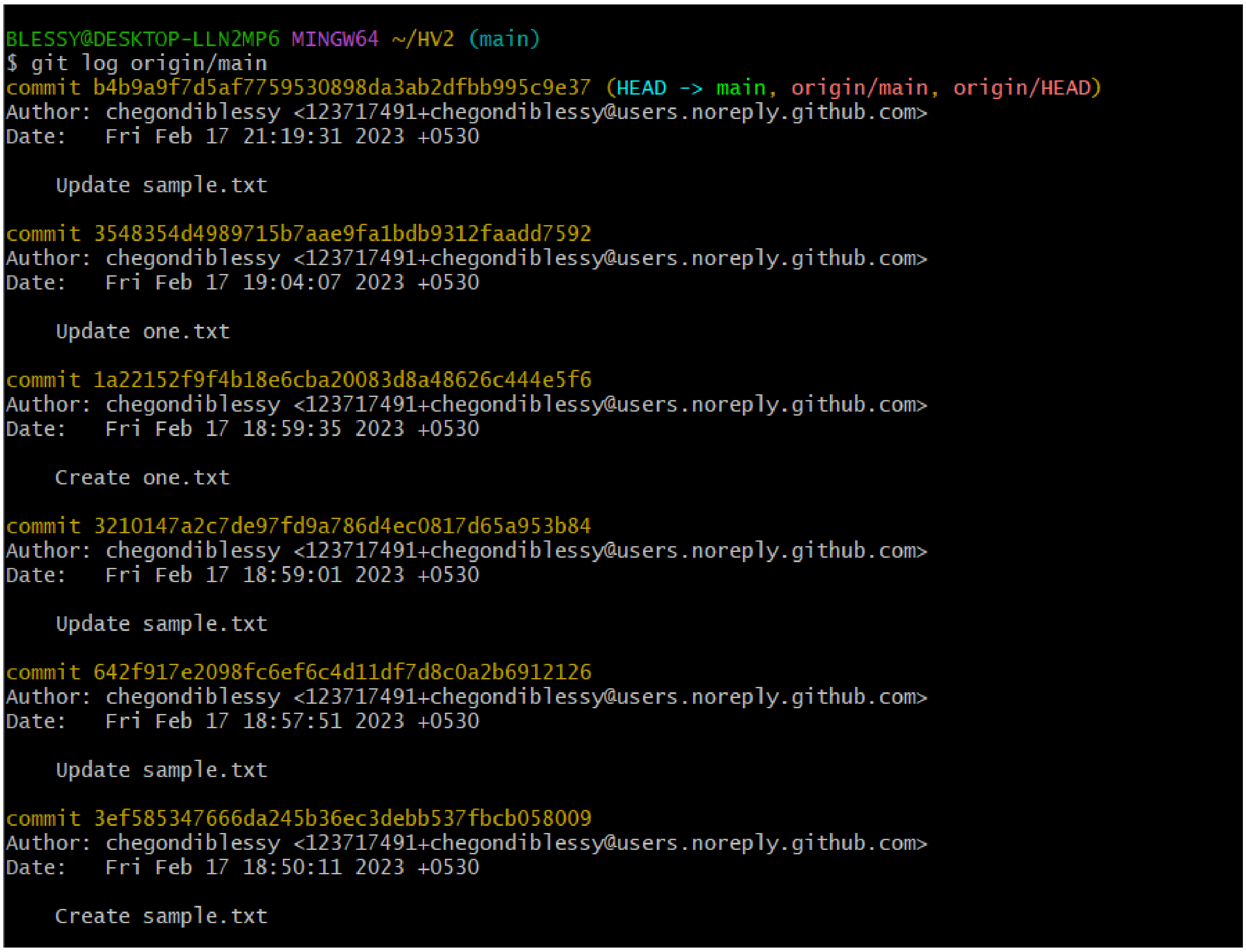
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log --oneline
b4b9a9f (HEAD -> main, origin/main, origin/HEAD) Update sample.txt
3548354 Update one.txt
1a22152 Create one.txt
3210147 Update sample.txt
642f917 Update sample.txt
3ef5853 Create sample.txt
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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), 699 bytes | 36.00 KiB/s, done.
From https://github.com/chegondiblessy/HV2
   3548354..b4b9a9f  main       -> origin/main
```



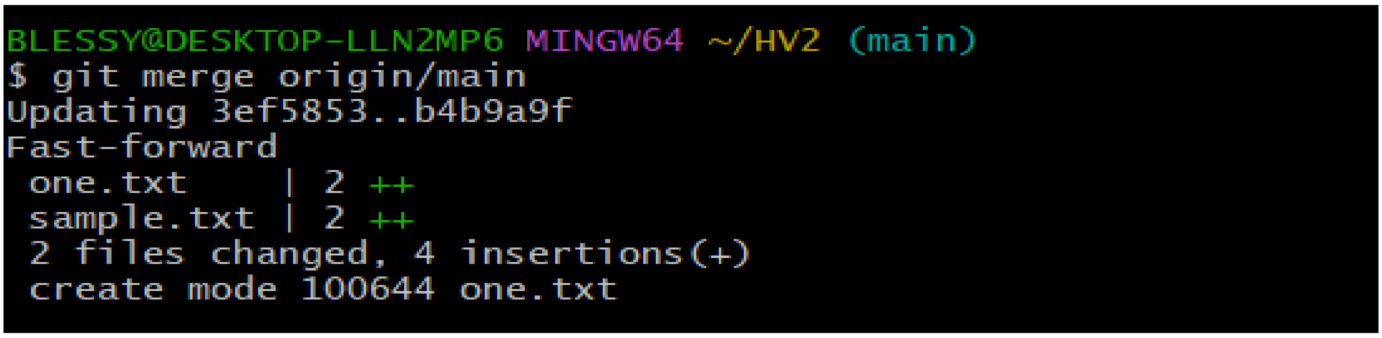
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log
commit b4b9a9f7d5af7759530898da3ab2dfbb995c9e37 (HEAD -> main, origin/main, origin/HEAD)
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 21:19:31 2023 +0530

    Update sample.txt

commit 3548354d4989715b7aae9fa1bdb9312faadd7592
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 19:04:07 2023 +0530

    Update one.txt

commit 1a22152f9f4b18e6cba20083d8a48626c444e5f6
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:35 2023 +0530

    Create one.txt

commit 3210147a2c7de97fd9a786d4ec0817d65a953b84
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:01 2023 +0530

    Update sample.txt

commit 642f917e2098fc6ef6c4d11df7d8c0a2b6912126
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:57:51 2023 +0530

    Update sample.txt

commit 3ef585347666da245b36ec3debb537fbcb058009
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:50:11 2023 +0530

    Create sample.txt
```



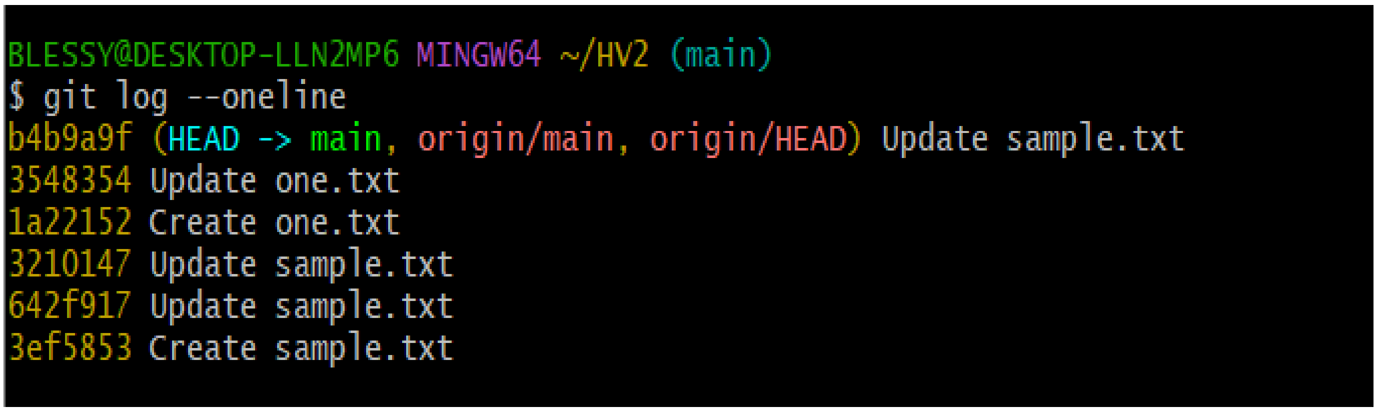
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log origin/main
commit b4b9a9f7d5af7759530898da3ab2dfbb995c9e37 (HEAD -> main, origin/main, origin/HEAD)
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 21:19:31 2023 +0530

    Update sample.txt

commit 3548354d4989715b7aae9fa1bdb9312faadd7592
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 19:04:07 2023 +0530

    Update one.txt

commit 1a22152f9f4b18e6cba20083d8a48626c444e5f6
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:35 2023 +0530

    Create one.txt

commit 3210147a2c7de97fd9a786d4ec0817d65a953b84
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:01 2023 +0530

    Update sample.txt

commit 642f917e2098fc6ef6c4d11df7d8c0a2b6912126
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:57:51 2023 +0530

    Update sample.txt

commit 3ef585347666da245b36ec3debb537fbcb058009
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:50:11 2023 +0530

    Create sample.txt
```

Git Merge:

Merging is Git's way of putting a forked history back together again. The **git merge** command lets you take the independent lines of development created by git branch and integrate them into a single branch. Git merge will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git merge origin/main
Updating 3ef5853..b4b9a9f
Fast-forward
 one.txt    | 2 ++
 sample.txt | 2 ++
 2 files changed, 4 insertions(+)
 create mode 100644 one.txt
```



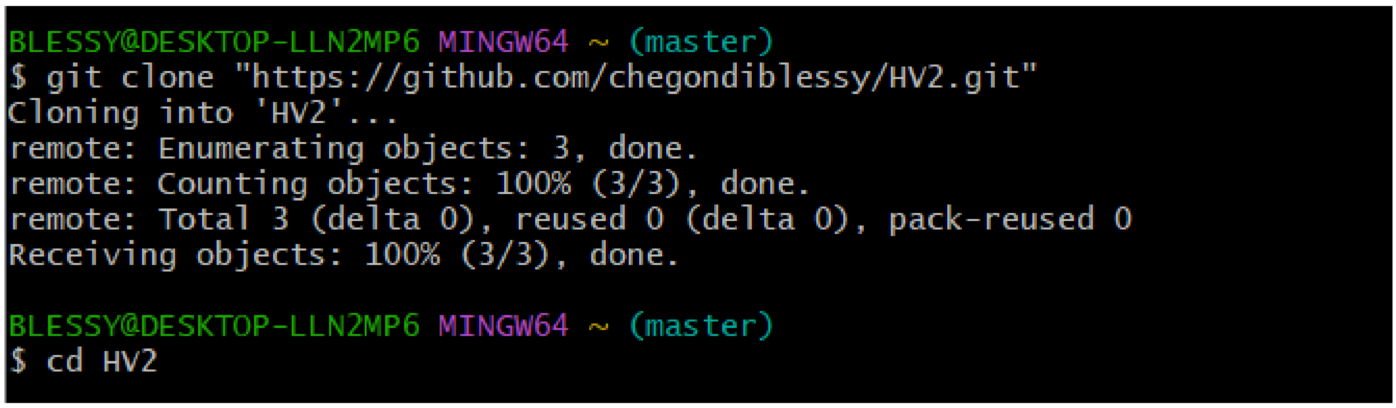
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log --oneline
b4b9a9f (HEAD -> main, origin/main, origin/HEAD) Update sample.txt
3548354 Update one.txt
1a22152 Create one.txt
3210147 Update sample.txt
642f917 Update sample.txt
3ef5853 Create sample.txt
```

3.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.

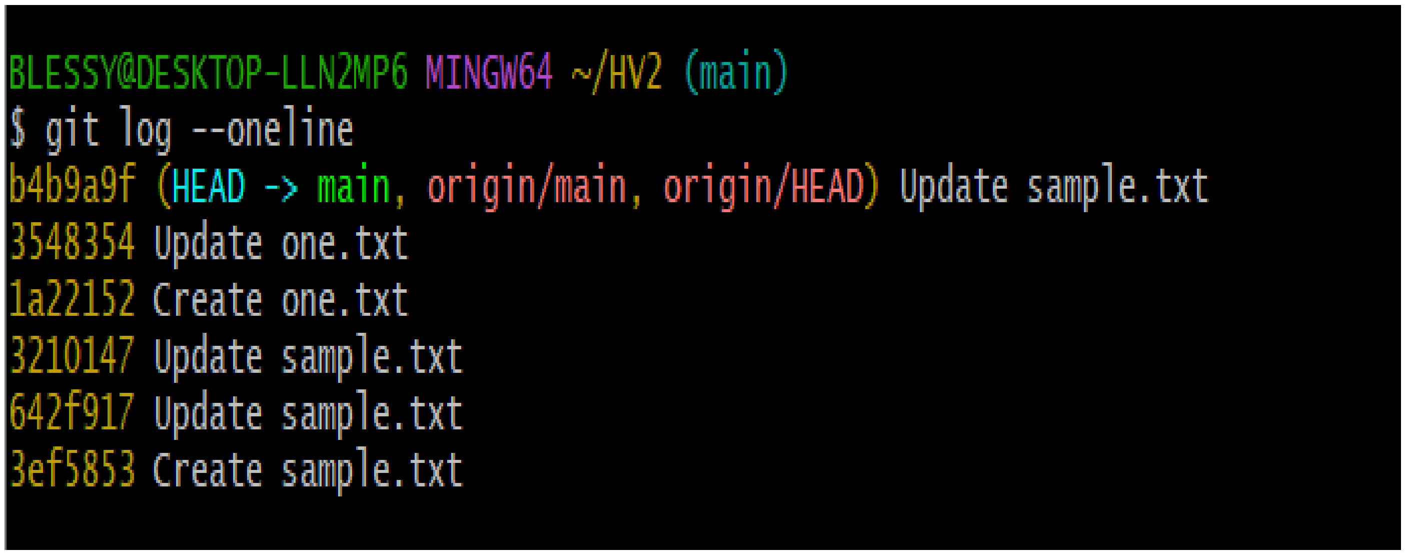
Ans:

Fetch:

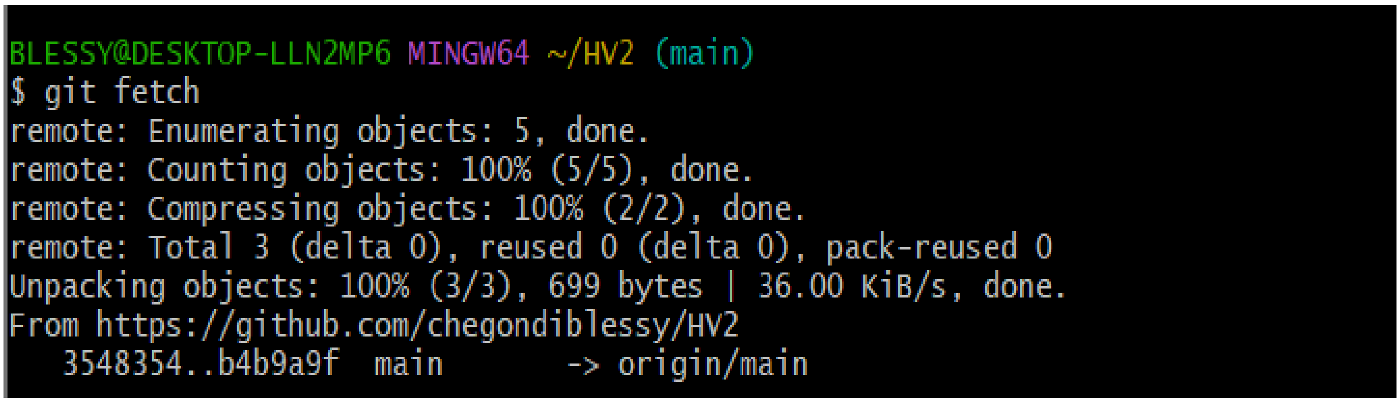
git fetch really only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files. Fetch is great for getting a fresh view on all the things that happened in a remote repository.
Due to it's "harmless" nature, you can rest assured: fetch will never manipulate, destroy, or screw up anything. This means you can never fetch often enough.



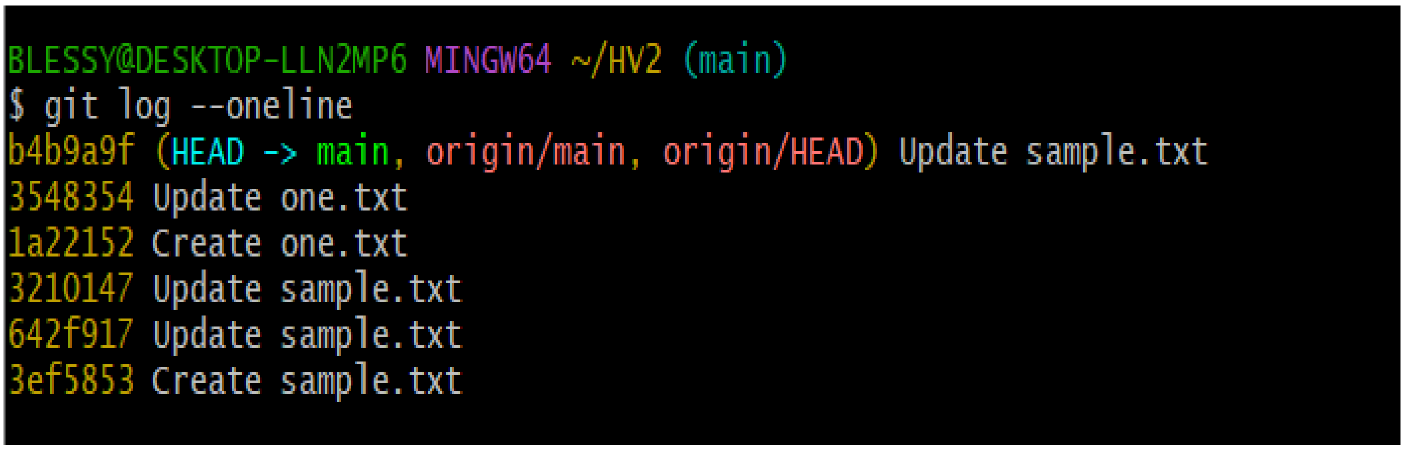
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ git clone "https://github.com/chegondiblessy/HV2.git"
Cloning into 'HV2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ cd HV2
```



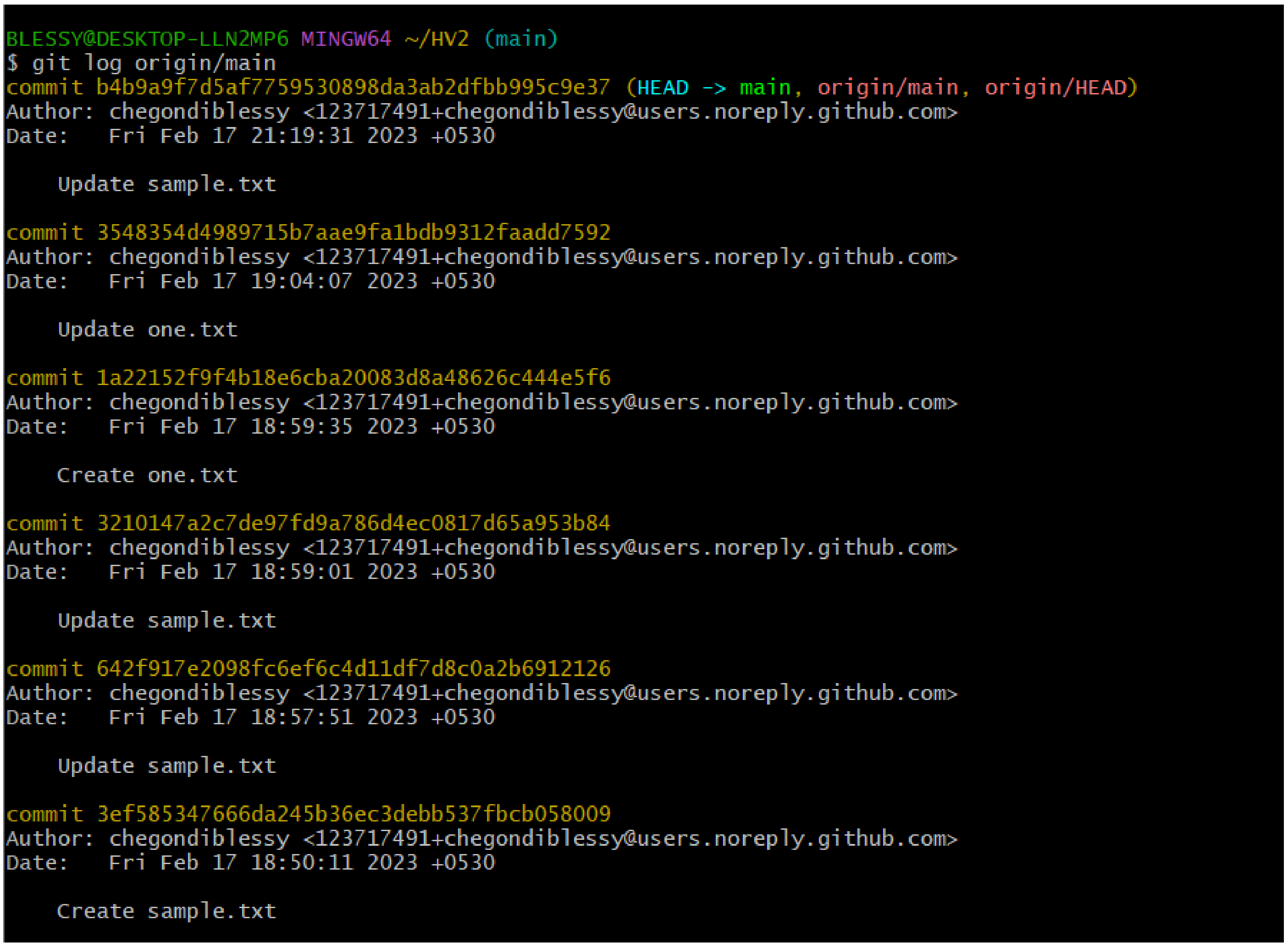
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log --oneline
b4b9a9f (HEAD -> main, origin/main, origin/HEAD) Update sample.txt
3548354 Update one.txt
1a22152 Create one.txt
3210147 Update sample.txt
642f917 Update sample.txt
3ef5853 Create sample.txt
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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), 699 bytes | 36.00 KiB/s, done.
From https://github.com/chegondiblessy/HV2
   3548354..b4b9a9f  main       -> origin/main
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log --oneline
b4b9a9f (HEAD -> main, origin/main, origin/HEAD) Update sample.txt
3548354 Update one.txt
1a22152 Create one.txt
3210147 Update sample.txt
642f917 Update sample.txt
3ef5853 Create sample.txt
```

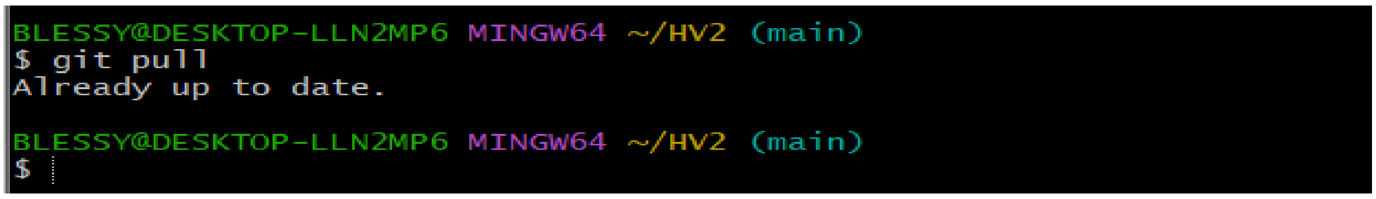


```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log origin/main
commit b4b9a9f7d5af7759530898da3ab2dfbb995c9e37 (HEAD -> main, origin/main, origin/HEAD)
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 21:19:31 2023 +0530

    Update sample.txt

commit 3548354d4989715b7aae9fa1bdb9312faadd7592
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 19:04:07 2023 +0530

    Update one.txt

commit 1a22152f9f4b18e6cba20083d8a48626c444e5f6
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:35 2023 +0530

    Create one.txt

commit 3210147a2c7de97fd9a786d4ec0817d65a953b84
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:59:01 2023 +0530

    Update sample.txt

commit 642f917e2098fc6ef6c4d11df7d8c0a2b6912126
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:57:51 2023 +0530

    Update sample.txt

commit 3ef585347666da245b36ec3debb537fbcb058009
Author: chegondiblessy <123717491+chegondiblessy@users.noreply.github.com>
Date:   Fri Feb 17 18:50:11 2023 +0530

    Create sample.txt
```

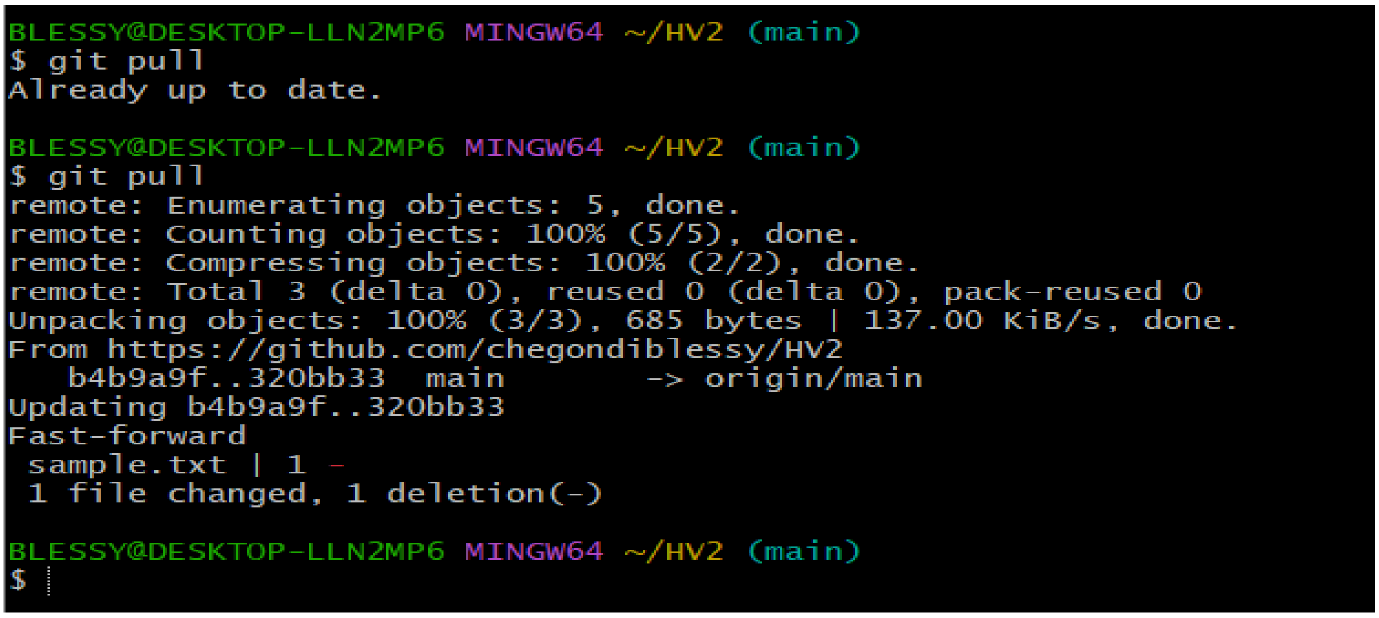
Pull:

git pull, in contrast, is used with a different goal in mind: to update your current HEAD branch with the latest changes from the remote server. This means that pull not only downloads new data; it also directly integrates it into your current working copy files. This has a couple of consequences:

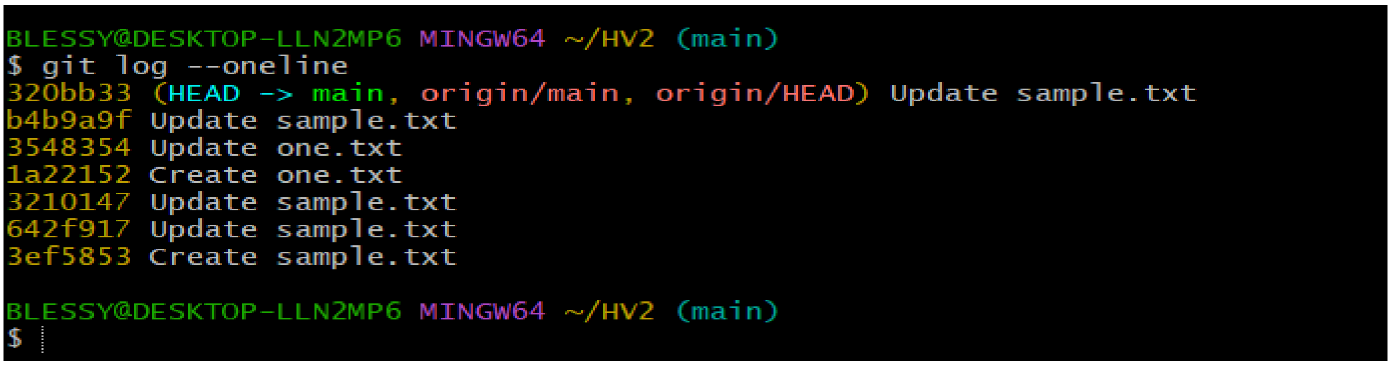
- Since "git pull" tries to merge remote changes with your local ones, a so-called "merge conflict" can occur.
- Like for many other actions, it's highly recommended to start a "git pull" only with a clean working copy. This means that you should *not* have any uncommitted local changes before you pull.



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git pull
Already up to date.

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git pull
Already up to date.

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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), 685 bytes | 137.00 KiB/s, done.
From https://github.com/chegondiblessy/HV2
   b4b9a9f..320bb33  main       -> origin/main
Updating b4b9a9f..320bb33
Fast-forward
 sample.txt | 1 -
 1 file changed, 1 deletion(-)

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ git log --oneline
320bb33 (HEAD -> main, origin/main, origin/HEAD) Update sample.txt
b4b9a9f Update sample.txt
3548354 Update one.txt
1a22152 Create one.txt
3210147 Update sample.txt
642f917 Update sample.txt
3ef5853 Create sample.txt

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$
```

Difference between git fetch and git pull:

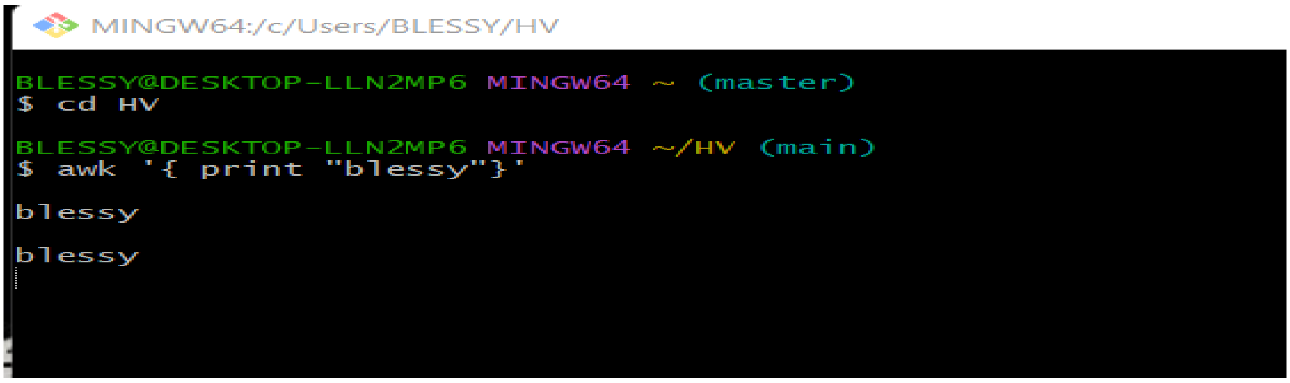
- When comparing Git pull vs fetch, Git fetch is a safer alternative because it pulls in all the commits from your remote but doesn't make any changes to your local files.
- Git pull is faster as you're performing multiple actions in one – a better bang for your buck. Using the Git pull command can be seen in one light as a feature of convenience; you're probably less worried about introducing conflicts into your local repo and you just want the most up-to-date changes from the remote branch you're pulling from.
- Git pull is a more advanced action and it's important to understand that you will be introducing changes and immediately applying them to your currently checked out branch.

4.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.

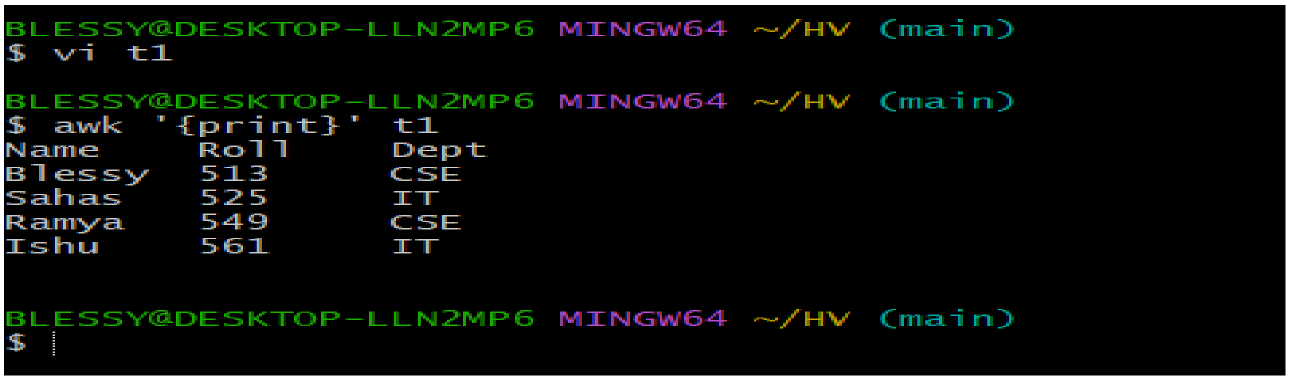
Ans:

Awk:

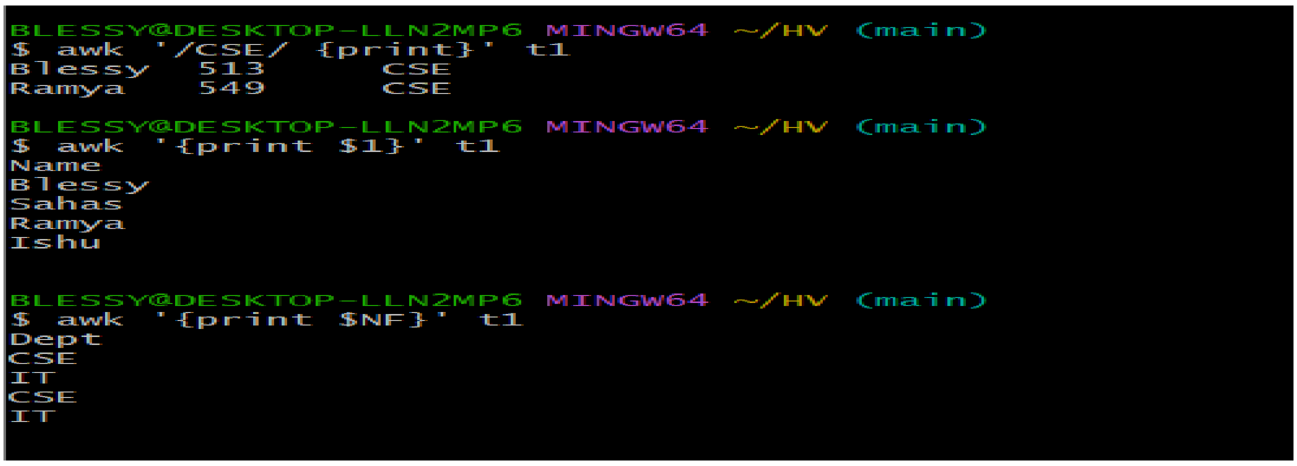
- The Awk is a powerful scripting language used for **text scripting**. It searches and replaces the texts and sorts, validates, and indexes the database. It performs various actions on a file like searching a specified text and more.
- The awk command is a Linux tool and programming language that allows users to process and manipulate data and produce formatted reports. The tool supports various operations for advanced text processing and facilitates expressing complex data selections.



```
MINGW64:/c/Users/BLESSY/HV
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~ (master)
$ cd HV

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ awk '{ print "blessy"}'

blessy

blessy
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ vi t1

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ awk '{print}' t1
Name     Roll     Dept
Blessy   513      CSE
Sahas    525      IT
Ramya    549      CSE
Ishu     561      IT


BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$
```



```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ awk '/CSE/ {print}' t1
Blessy   513      CSE
Ramya    549      CSE

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ awk '{print $1}' t1
Name
Blessy
Sahas
Ramya
Ishu


BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ awk '{print $NF}' t1
Dept
CSE
IT
CSE
IT
```

Steps to follow bash scripting:

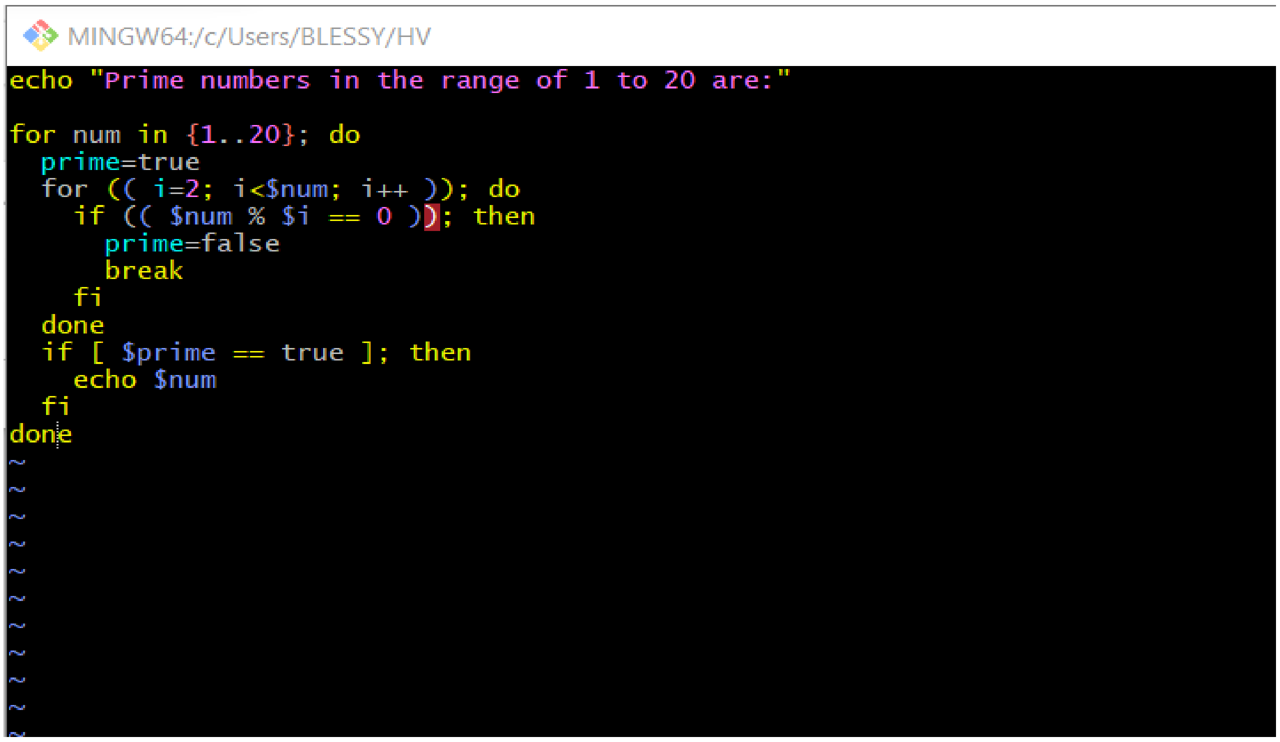
Step1 : Create the file with extension .sh.

Step 2 : Open the shell and write the script.

Step 3 : Save the code and run the code .

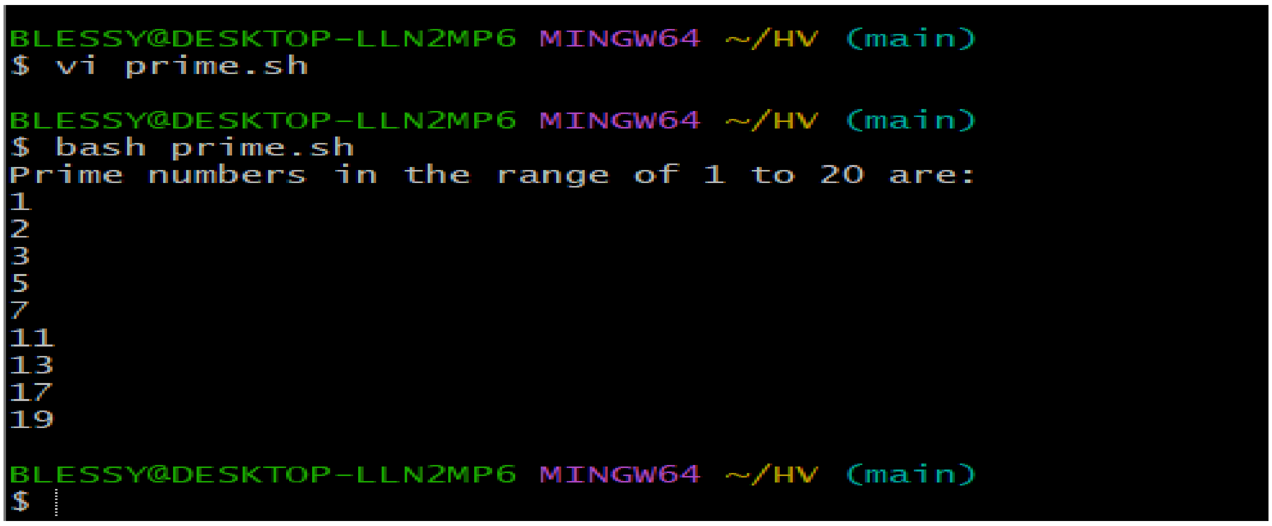
To run the run a code

Syntax: bash filename.sh



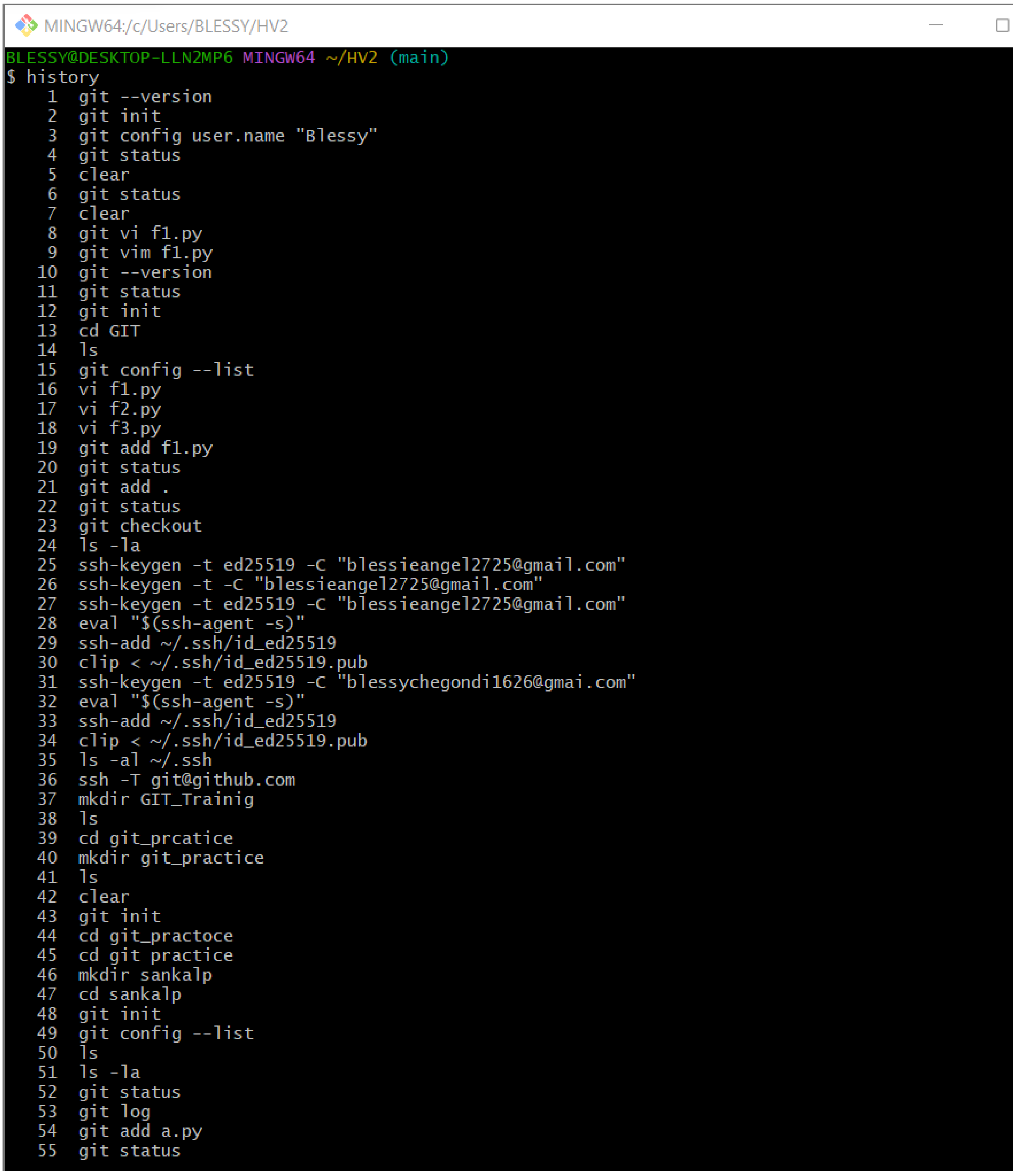
MINGW64:/c/Users/BLESSY/HV

```
echo "Prime numbers in the range of 1 to 20 are:"

for num in {1..20}; do
  prime=true
  for (( i=2; i<$num; i++ )); do
    if (( $num % $i == 0 )); then
      prime=false
      break
    fi
  done
  if [ $prime == true ]; then
    echo $num
  fi
done
~
~
~
~
~
~
~
~
~
~
~
```

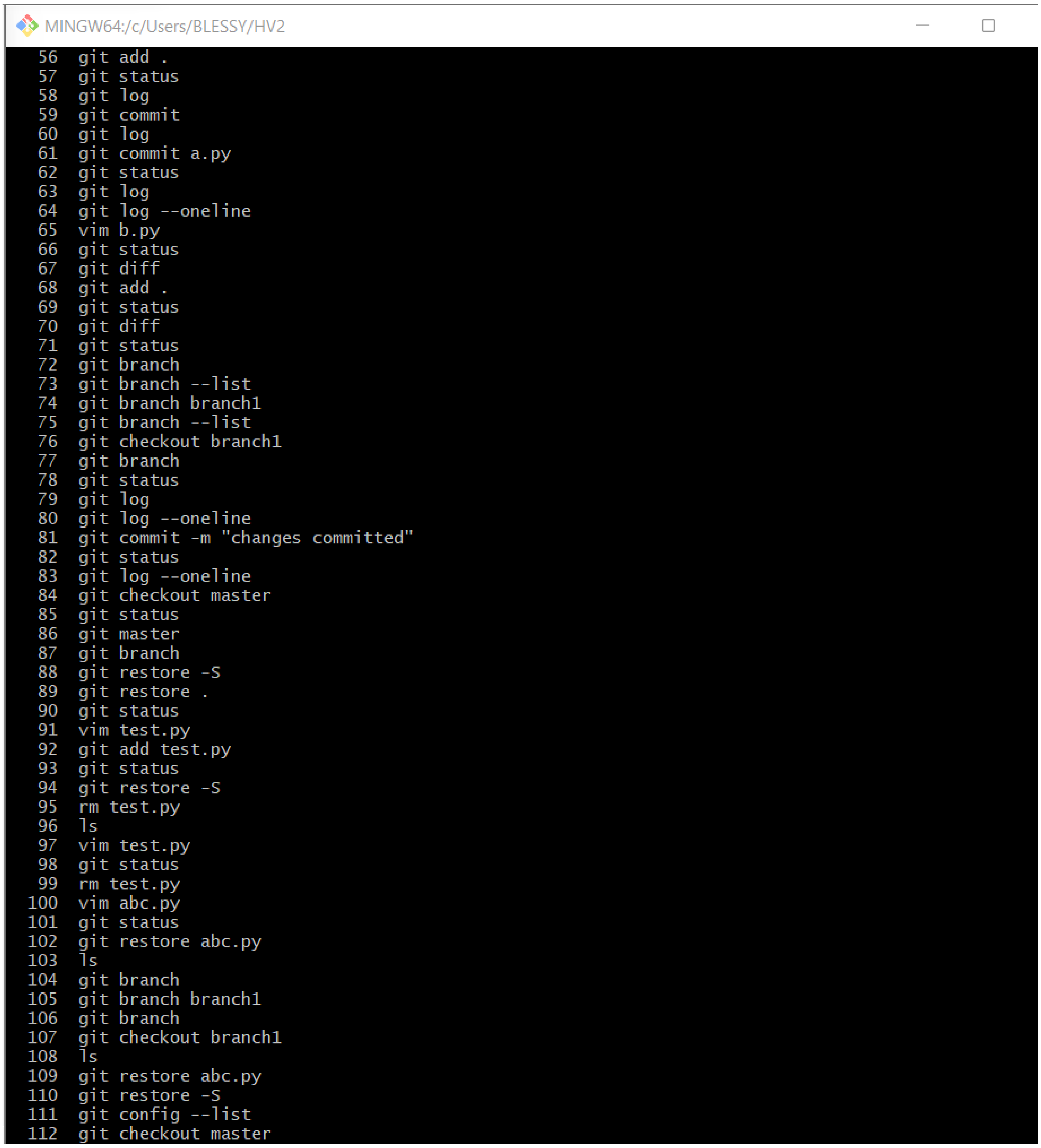


```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ vi prime.sh

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$ bash prime.sh
Prime numbers in the range of 1 to 20 are:
1
2
3
5
7
11
13
17
19

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV (main)
$
```

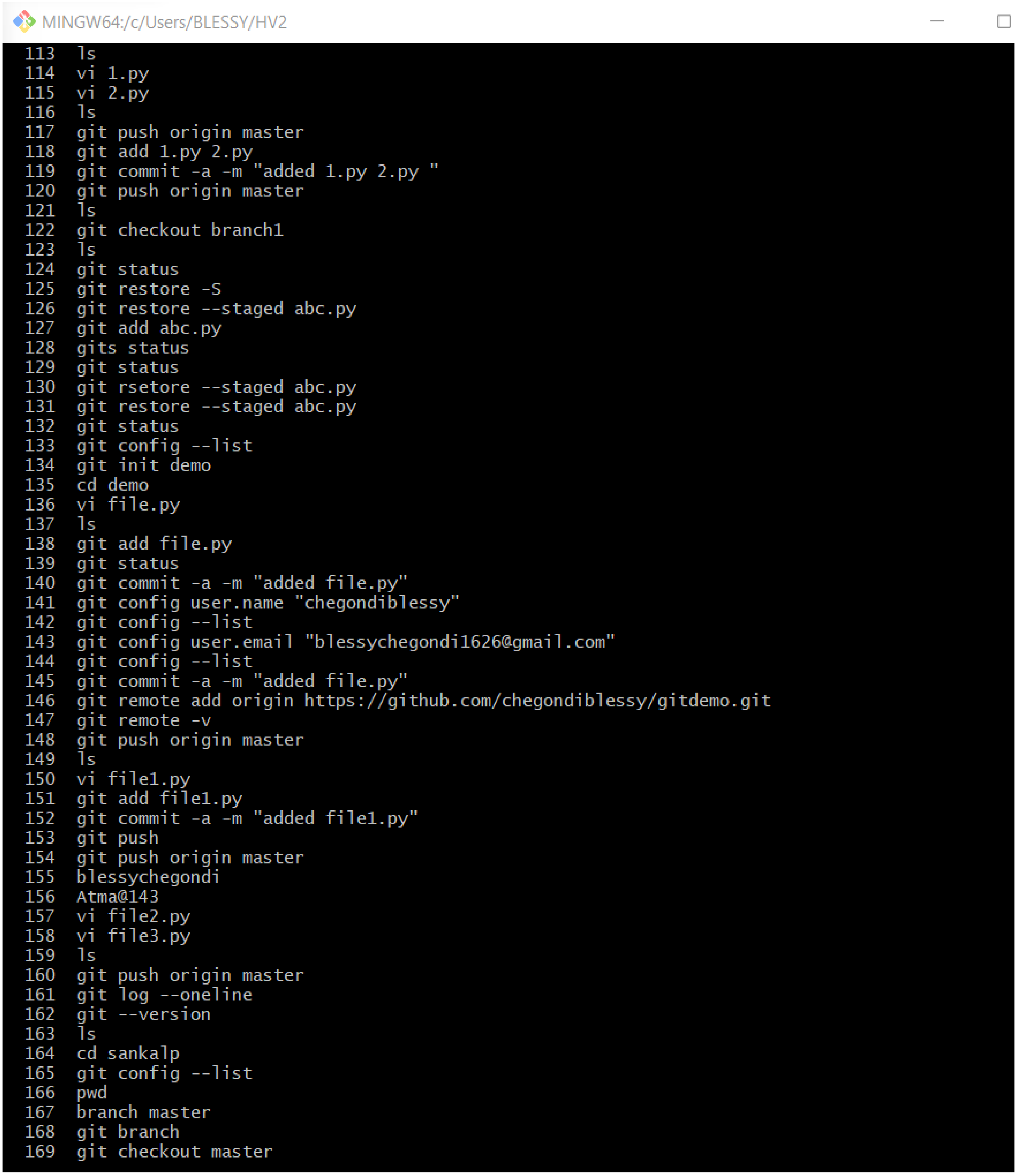
History:
History command used to show the history of the commands which we are executed until now.


MINGW64:/c/Users/BLESSY/HV2

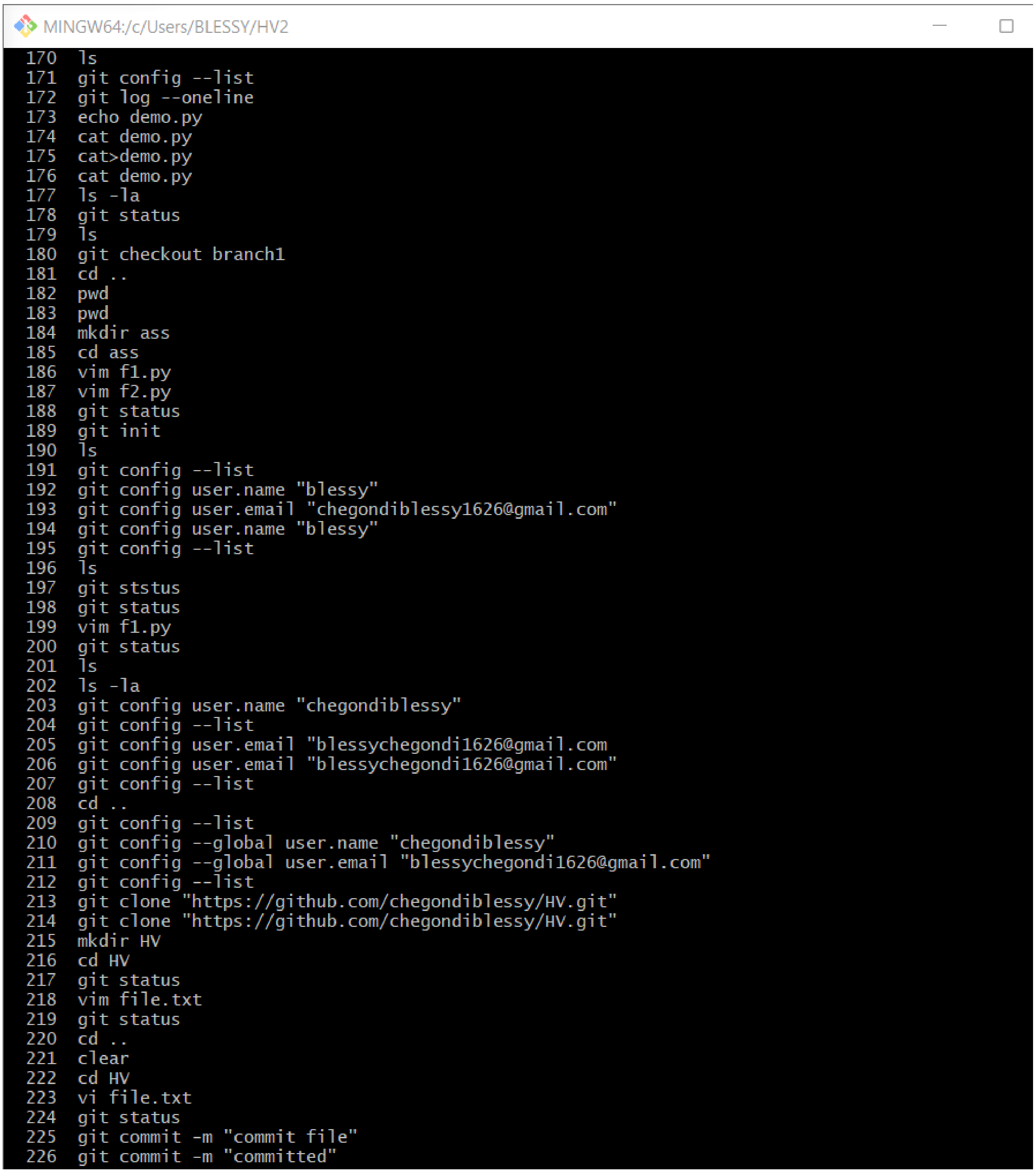
```
BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$ history
    1  git --version
    2  git init
    3  git config user.name "Blessy"
    4  git status
    5  clear
    6  git status
    7  clear
    8  git vi f1.py
    9  git vim f1.py
   10  git --version
   11  git status
   12  git init
   13  cd GIT
   14  ls
   15  git config --list
   16  vi f1.py
   17  vi f2.py
   18  vi f3.py
   19  git add f1.py
   20  git status
   21  git add .
   22  git status
   23  git checkout
   24  ls -la
   25  ssh-keygen -t ed25519 -C "blessieangel2725@gmail.com"
   26  ssh-keygen -t -C "blessieangel2725@gmail.com"
   27  ssh-keygen -t ed25519 -C "blessieangel2725@gmail.com"
   28  eval "$(ssh-agent -s)"
   29  ssh-add ~/.ssh/id_ed25519
   30  clip < ~/.ssh/id_ed25519.pub
   31  ssh-keygen -t ed25519 -C "blessychegondi1626@gmai.com"
   32  eval "$(ssh-agent -s)"
   33  ssh-add ~/.ssh/id_ed25519
   34  clip < ~/.ssh/id_ed25519.pub
   35  ls -al ~/.ssh
   36  ssh -T git@github.com
   37  mkdir GIT_Trainig
   38  ls
   39  cd git_prcatice
   40  mkdir git_practice
   41  ls
   42  clear
   43  git init
   44  cd git_practoce
   45  cd git practice
   46  mkdir sankalp
   47  cd sankalp
   48  git init
   49  git config --list
   50  ls
   51  ls -la
   52  git status
   53  git log
   54  git add a.py
   55  git status
```



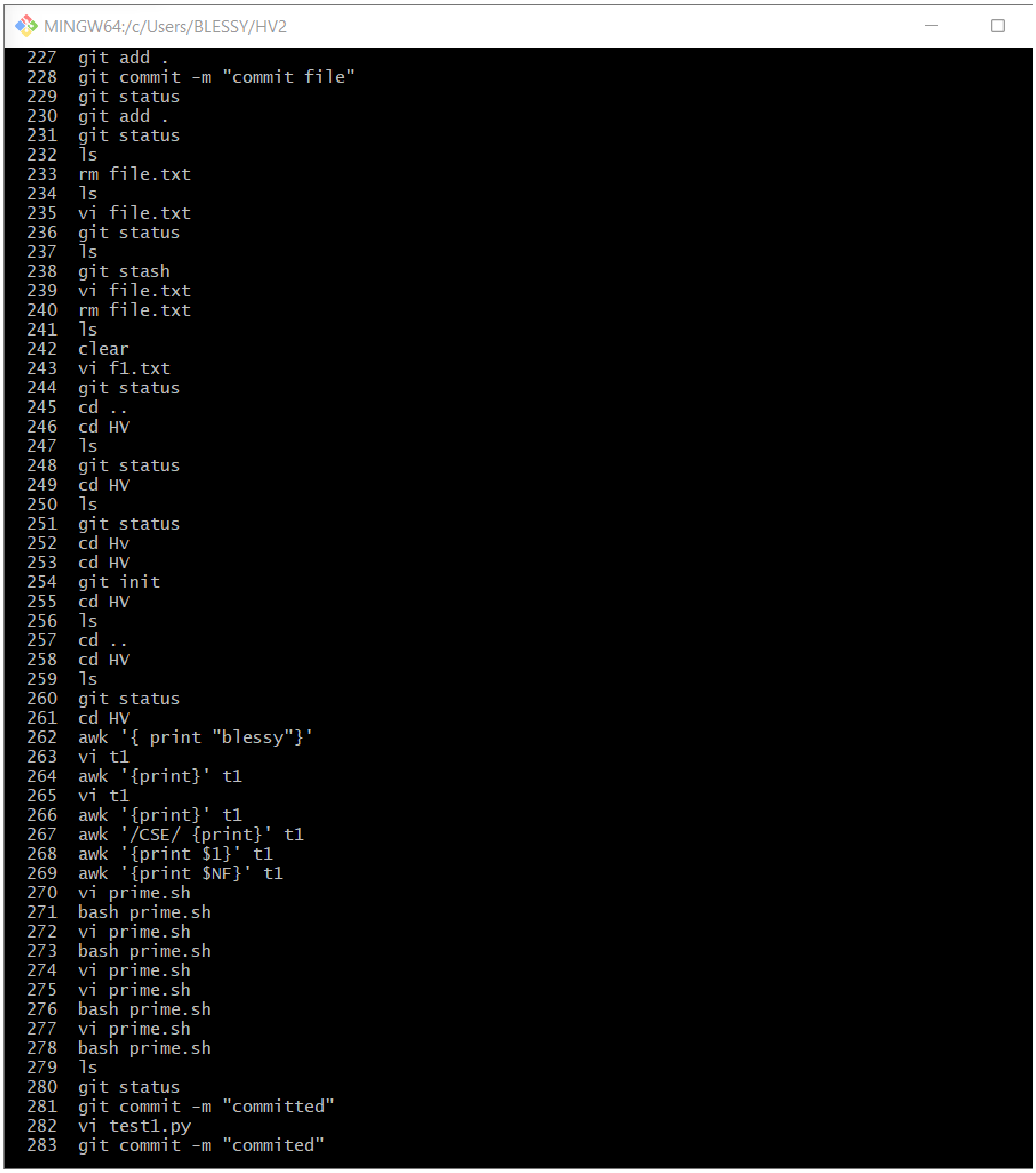
```
   56  git add .
   57  git status
   58  git log
   59  git commit
   60  git log
   61  git commit a.py
   62  git status
   63  git log
   64  git log --oneline
   65  vim b.py
   66  git status
   67  git diff
   68  git add .
   69  git status
   70  git diff
   71  git status
   72  git branch
   73  git branch --list
   74  git branch branch1
   75  git branch --list
   76  git checkout branch1
   77  git branch
   78  git status
   79  git log
   80  git log --oneline
   81  git commit -m "changes committed"
   82  git status
   83  git log --oneline
   84  git checkout master
   85  git status
   86  git master
   87  git branch
   88  git restore -S
   89  git restore .
   90  git status
   91  vim test.py
   92  git add test.py
   93  git status
   94  git restore -S
   95  rm test.py
   96  ls
   97  vim test.py
   98  git status
   99  rm test.py
  100  vim abc.py
  101  git status
  102  git restore abc.py
  103  ls
  104  git branch
  105  git branch branch1
  106  git branch
  107  git checkout branch1
  108  ls
  109  git restore abc.py
  110  git restore -S
  111  git config --list
  112  git checkout master
```



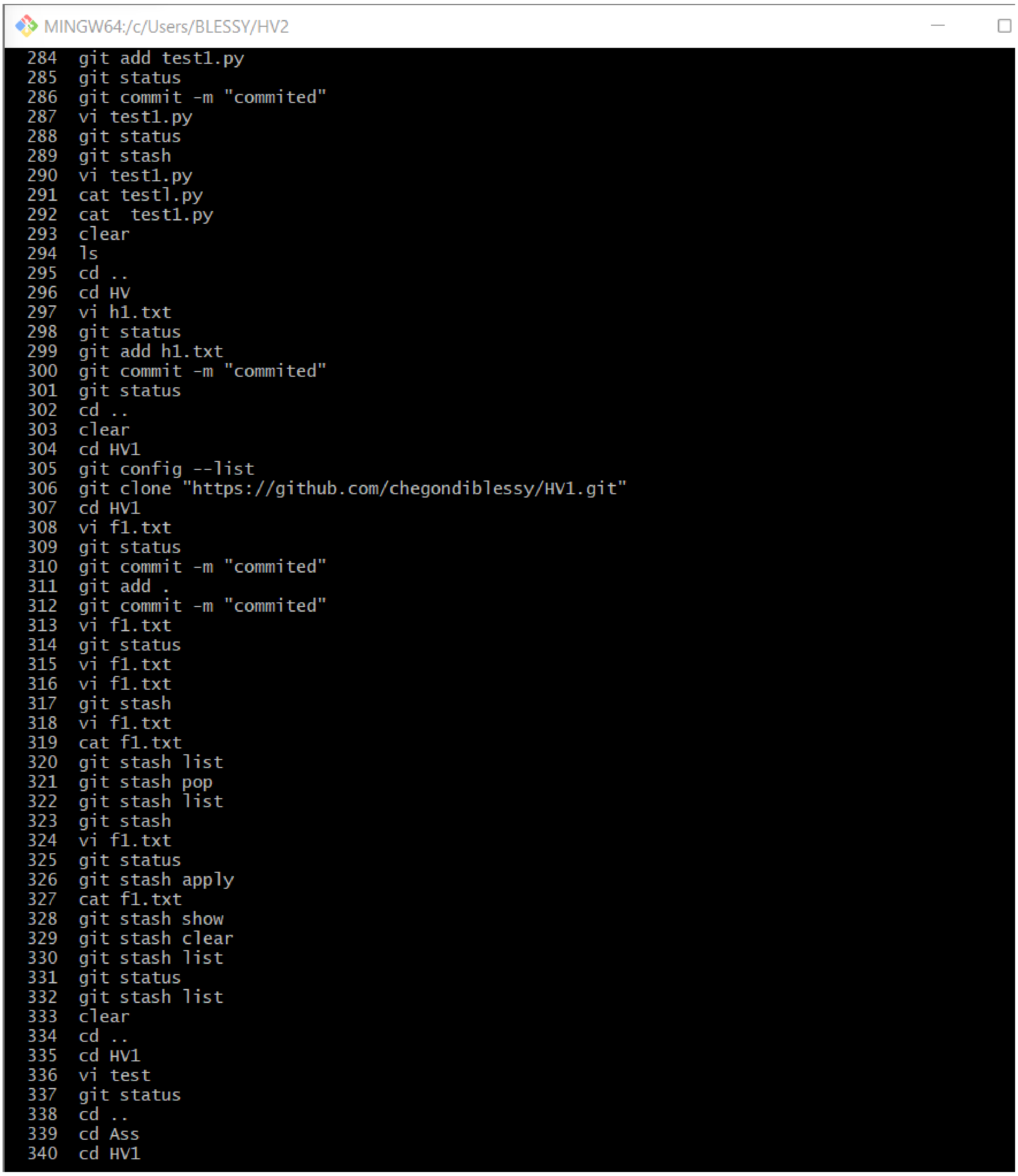
```
  113  ls
  114  vi 1.py
  115  vi 2.py
  116  ls
  117  git push origin master
  118  git add 1.py 2.py
  119  git commit -a -m "added 1.py 2.py "
  120  git push origin master
  121  ls
  122  git checkout branch1
  123  ls
  124  git status
  125  git restore -S
  126  git restore --staged abc.py
  127  git add abc.py
  128  gits status
  129  git status
  130  git rsetore --staged abc.py
  131  git restore --staged abc.py
  132  git status
  133  git config --list
  134  git init demo
  135  cd demo
  136  vi file.py
  137  ls
  138  git add file.py
  139  git status
  140  git commit -a -m "added file.py"
  141  git config user.name "chegondiblessy"
  142  git config --list
  143  git config user.email "blessychegondi1626@gmail.com"
  144  git config --list
  145  git commit -a -m "added file.py"
  146  git remote add origin https://github.com/chegondiblessy/gitdemo.git
  147  git remote -v
  148  git push origin master
  149  ls
  150  vi file1.py
  151  git add file1.py
  152  git commit -a -m "added file1.py"
  153  git push
  154  git push origin master
  155  blessychegondi
  156  Atma@143
  157  vi file2.py
  158  vi file3.py
  159  ls
  160  git push origin master
  161  git log --oneline
  162  git --version
  163  ls
  164  cd sankalp
  165  git config --list
  166  pwd
  167  branch master
  168  git branch
  169  git checkout master
```



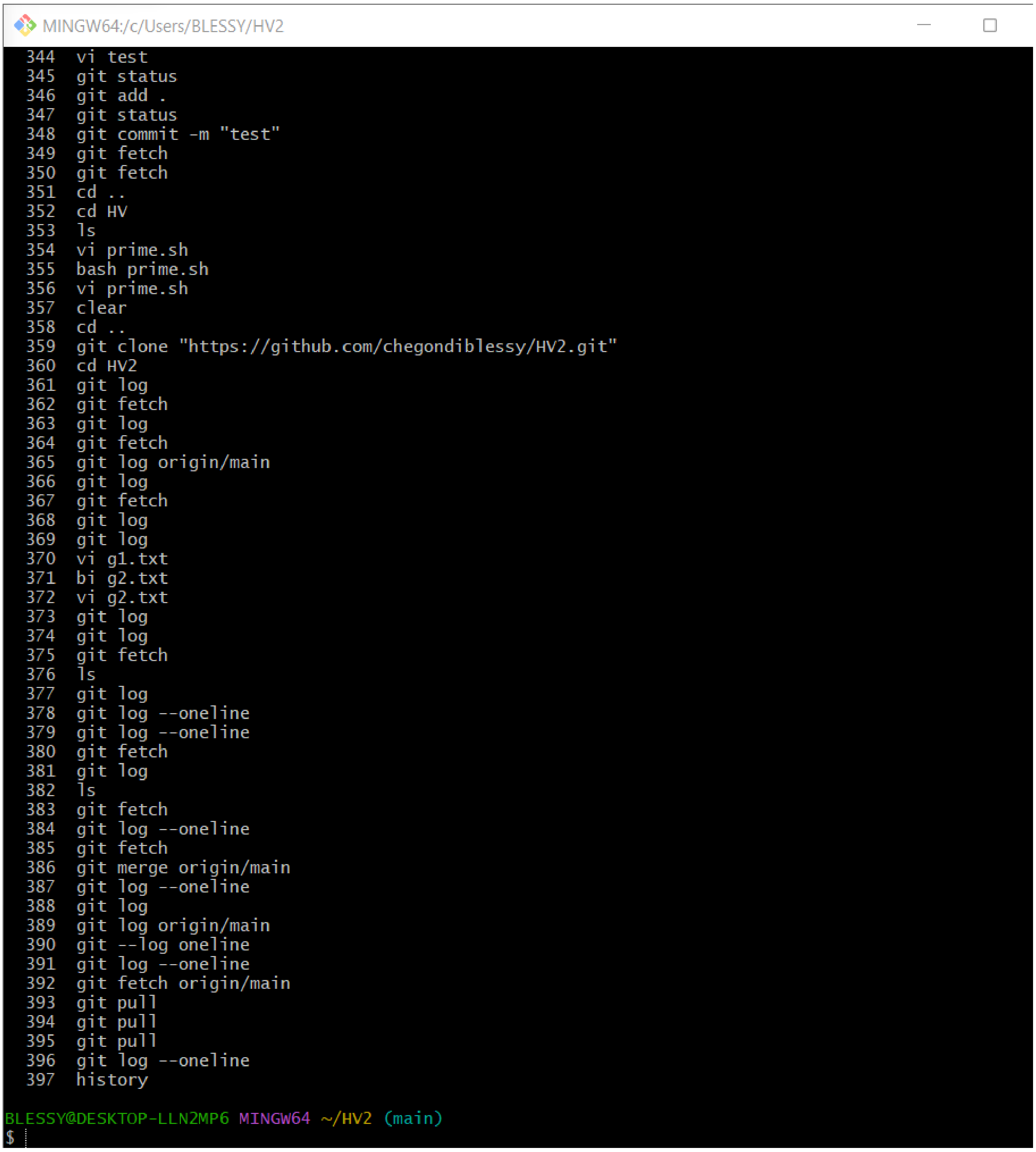
```
  170  ls
  171  git config --list
  172  git log --oneline
  173  echo demo.py
  174  cat demo.py
  175  cat>demo.py
  176  cat demo.py
  177  ls -la
  178  git status
  179  ls
  180  git checkout branch1
  181  cd ..
  182  pwd
  183  pwd
  184  mkdir ass
  185  cd ass
  186  vim f1.py
  187  vim f2.py
  188  git status
  189  git init
  190  ls
  191  git config --list
  192  git config user.name "blessy"
  193  git config user.email "chegondiblessy1626@gmail.com"
  194  git config user.name "blessy"
  195  git config --list
  196  ls
  197  git ststus
  198  git status
  199  vim f1.py
  200  git status
  201  ls
  202  ls -la
  203  git config user.name "chegondiblessy"
  204  git config --list
  205  git config user.email "blessychegondi1626@gmail.com
  206  git config user.email "blessychegondi1626@gmail.com"
  207  git config --list
  208  cd ..
  209  git config --list
  210  git config --global user.name "chegondiblessy"
  211  git config --global user.email "blessychegondi1626@gmail.com"
  212  git config --list
  213  git clone "https://github.com/chegondiblessy/HV.git"
  214  git clone "https://github.com/chegondiblessy/HV.git"
  215  mkdir HV
  216  cd HV
  217  git status
  218  vim file.txt
  219  git status
  220  cd ..
  221  clear
  222  cd HV
  223  vi file.txt
  224  git status
  225  git commit -m "commit file"
  226  git commit -m "committed"
```


MINGW64:/c/Users/BLESSY/HV2

```
  227  git add .
  228  git commit -m "commit file"
  229  git status
  230  git add .
  231  git status
  232  ls
  233  rm file.txt
  234  ls
  235  vi file.txt
  236  git status
  237  ls
  238  git stash
  239  vi file.txt
  240  rm file.txt
  241  ls
  242  clear
  243  vi f1.txt
  244  git status
  245  cd ..
  246  cd HV
  247  ls
  248  git status
  249  cd HV
  250  ls
  251  git status
  252  cd Hv
  253  cd HV
  254  git init
  255  cd HV
  256  ls
  257  cd ..
  258  cd HV
  259  ls
  260  git status
  261  cd HV
  262  awk '{ print "blessy"}'
  263  vi t1
  264  awk '{print}' t1
  265  vi t1
  266  awk '{print}' t1
  267  awk '/CSE/ {print}' t1
  268  awk '{print $1}' t1
  269  awk '{print $NF}' t1
  270  vi prime.sh
  271  bash prime.sh
  272  vi prime.sh
  273  bash prime.sh
  274  vi prime.sh
  275  vi prime.sh
  276  bash prime.sh
  277  vi prime.sh
  278  bash prime.sh
  279  ls
  280  git status
  281  git commit -m "committed"
  282  vi test1.py
  283  git commit -m "commited"
```



```
  284  git add test1.py
  285  git status
  286  git commit -m "commited"
  287  vi test1.py
  288  git status
  289  git stash
  290  vi test1.py
  291  cat testl.py
  292  cat  test1.py
  293  clear
  294  ls
  295  cd ..
  296  cd HV
  297  vi h1.txt
  298  git status
  299  git add h1.txt
  300  git commit -m "commited"
  301  git status
  302  cd ..
  303  clear
  304  cd HV1
  305  git config --list
  306  git clone "https://github.com/chegondiblessy/HV1.git"
  307  cd HV1
  308  vi f1.txt
  309  git status
  310  git commit -m "commited"
  311  git add .
  312  git commit -m "commited"
  313  vi f1.txt
  314  git status
  315  vi f1.txt
  316  vi f1.txt
  317  git stash
  318  vi f1.txt
  319  cat f1.txt
  320  git stash list
  321  git stash pop
  322  git stash list
  323  git stash
  324  vi f1.txt
  325  git status
  326  git stash apply
  327  cat f1.txt
  328  git stash show
  329  git stash clear
  330  git stash list
  331  git status
  332  git stash list
  333  clear
  334  cd ..
  335  cd HV1
  336  vi test
  337  git status
  338  cd ..
  339  cd Ass
  340  cd HV1
```



```
  344  vi test
  345  git status
  346  git add .
  347  git status
  348  git commit -m "test"
  349  git fetch
  350  git fetch
  351  cd ..
  352  cd HV
  353  ls
  354  vi prime.sh
  355  bash prime.sh
  356  vi prime.sh
  357  clear
  358  cd ..
  359  git clone "https://github.com/chegondiblessy/HV2.git"
  360  cd HV2
  361  git log
  362  git fetch
  363  git log
  364  git fetch
  365  git log origin/main
  366  git log
  367  git fetch
  368  git log
  369  git log
  370  vi g1.txt
  371  bi g2.txt
  372  vi g2.txt
  373  git log
  374  git log
  375  git fetch
  376  ls
  377  git log
  378  git log --oneline
  379  git log --oneline
  380  git fetch
  381  git log
  382  ls
  383  git fetch
  384  git log --oneline
  385  git fetch
  386  git merge origin/main
  387  git log --oneline
  388  git log
  389  git log origin/main
  390  git --log oneline
  391  git log --oneline
  392  git fetch origin/main
  393  git pull
  394  git pull
  395  git pull
  396  git log --oneline
  397  history

BLESSY@DESKTOP-LLN2MP6 MINGW64 ~/HV2 (main)
$
```

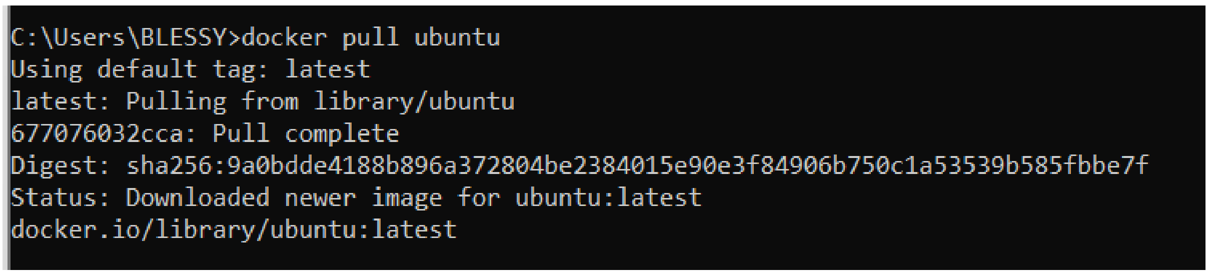
5.Set up a container and run a Ubuntu operating system.For this purpose,you can make use of the docker hub and run the container in interactive mode.

Ans:

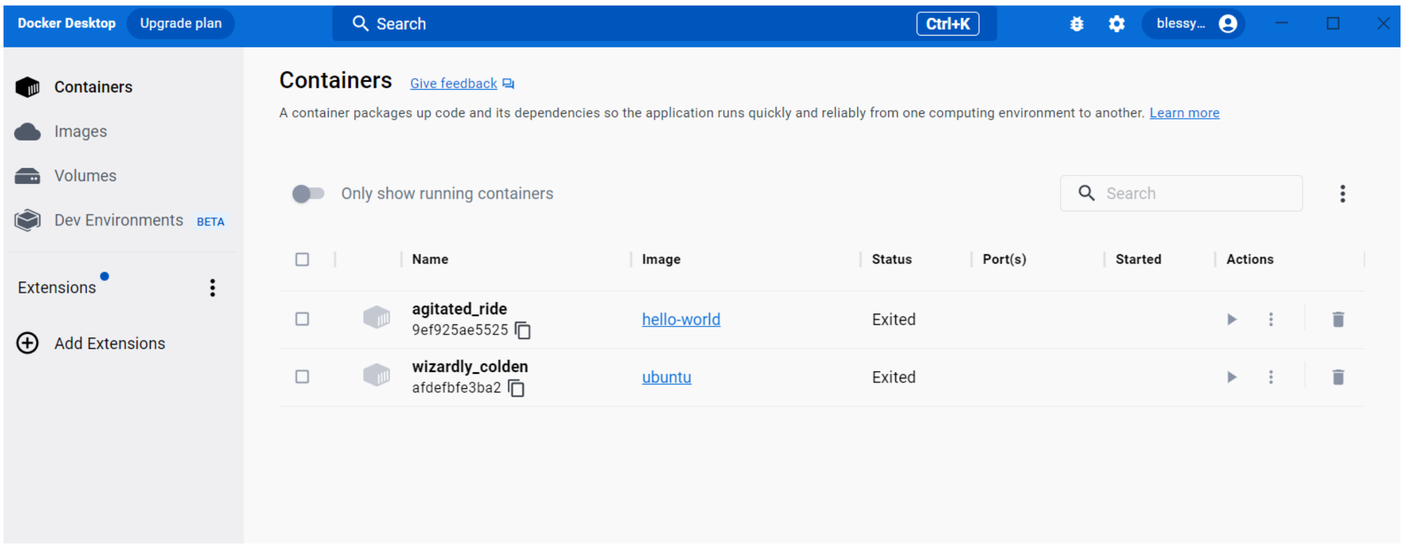
Image:
An image is a read-only template with instructions for creating a Docker container. A docker image is described in text file called a **Dockerfile**, which has a simple, well-defined syntax. An image does not have states and never changes. Docker Engine provides the core Docker technology that enables images and containers.
Container:
Docker container is a running instance of an image. You can use Command Line Interface (CLI) commands to run, start, stop, move, or delete a container. You can also provide configuration for the network and environment variables. Docker container is an isolated and secure application platform, but it can share and access to resources running in a different host or container.

Steps for running a Ubuntu OS:

- First we need to download the image of Ubuntu from docker hub using the command **docker pull ubuntu .**
- To create a container and execute the image use the command docker run -it ubuntu .
- To get an idea about the available update use apt update command.
- Download the ubuntu OS image from the docker hub.



```
C:\Users\BLESSY>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
677076032cca: Pull complete
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

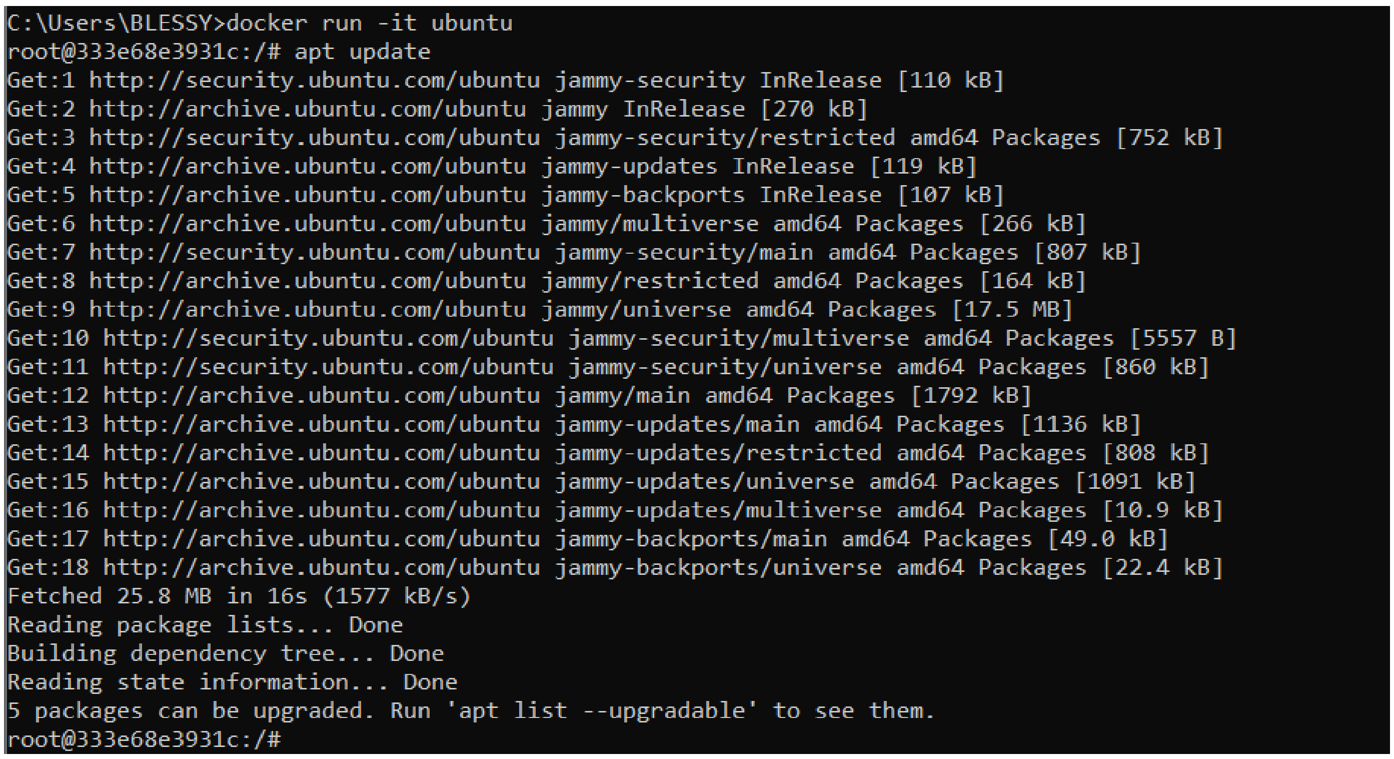


Containers Give feedback

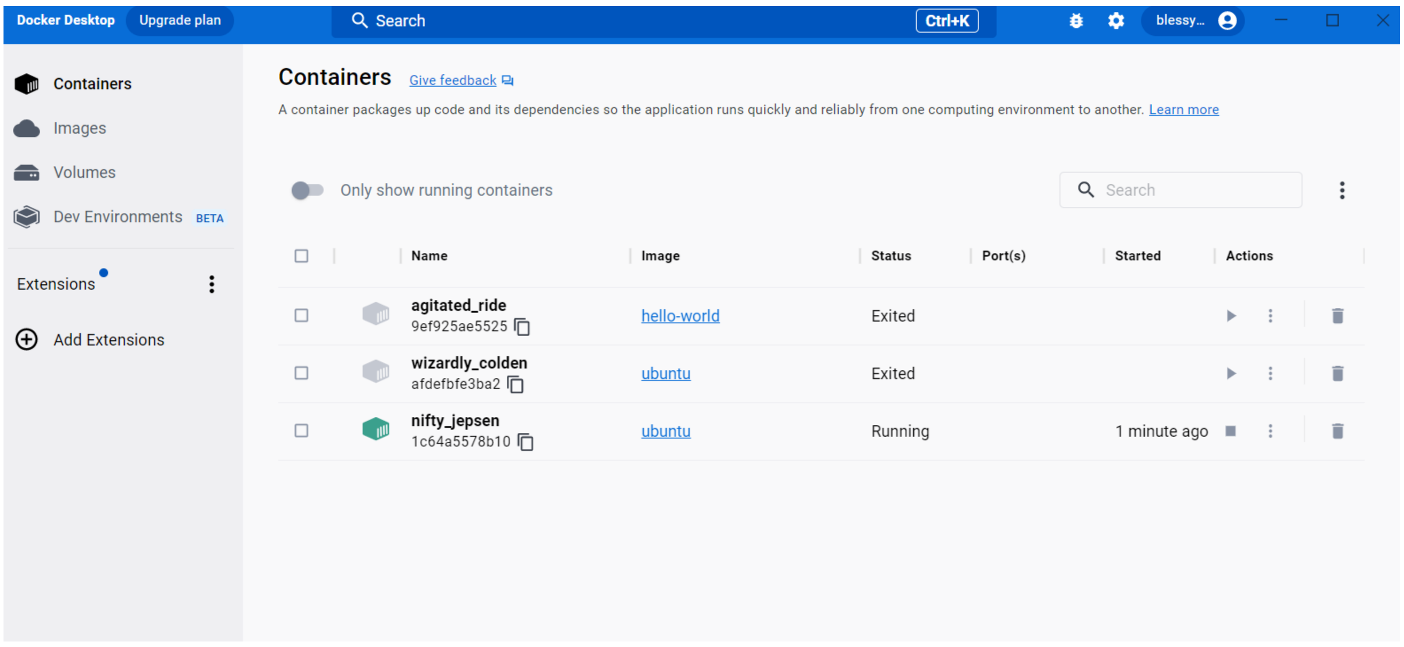
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Only show running containers

Name	Image	Status	Port(s)	Started	Actions
agitated_ride 9ef925ae5525	hello-world	Exited			
wizardly_colden afdefbfe3ba2	ubuntu	Exited			



```
C:\Users\BLESSY>docker run -it ubuntu
root@333e68e3931c:/# 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/restricted amd64 Packages [752 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [107 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [807 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [5557 B]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [860 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1136 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/multiverse amd64 Packages [10.9 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 16s (1577 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@333e68e3931c:/#
```

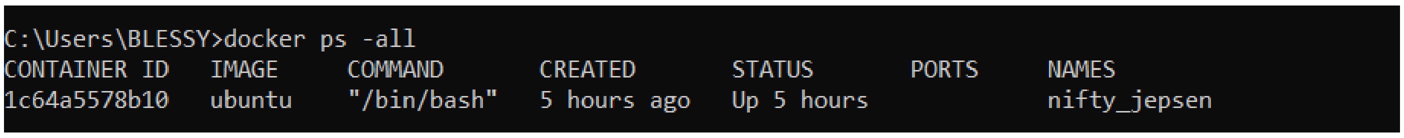


Containers

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

Only show running containers

Name	Image	Status	Port(s)	Started	Actions
agitated_ride 9ef925ae5525	hello-world	Exited			
wizardly_colden afdefbfe3ba2	ubuntu	Exited			
nifty_jepsen 1c64a5578b10	ubuntu	Running		1 minute ago	



```
C:\Users\BLESSY>docker ps -all
CONTAINER ID   IMAGE    COMMAND       CREATED       STATUS      PORTS     NAMES
1c64a5578b10   ubuntu   "/bin/bash"   5 hours ago   Up 5 hours            nifty_jepsen
```