### Viewing differences between files or commits

1. The `git diff` command is used to return the differences between the repository and the working directory. The repository is where Git saves the changes made and the working directory is the folder where you can physically see the files Git is managing.

2. Make some changes to any of the files in your working directory, for example Welcome.txt and then save the changes (you could add a couple of lines to the file or change one or more lines).

3. Now run the command `git status`. You should see a message similar to this one below.

```
On branch master

Changes not staged for commit:

  (use "git add <file>..." to update what will
be committed)

  (use "git checkout -- <file>..." to discard
changes in working directory)

        modified:   Welcome.txt

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

4. Now run the command `git diff`

5. What do you see? You should see some information that essentially show what changes have been made to the files in your working directory, taking reference from the repository version of the file (or files) that was changed. Take some time to study this information.

6. Recall that `git diff` shows only the differences between the repository version and the physical (or working) directory version. If you have staged any file (or files), you can view the differences

between your repository and the files in the staging area. To do this, run the command `git diff --staged`.

7. If changes were made to more than one file, the diff command would show all changes one after the other. To view the changes for a specific file, you can provide the file name as an argument to the diff command, for example, `git diff Welcome.txt`. Now run this (or a similar) command to view the changes made to one of the files in your working directory.

8. So far, we have been using git diff on files. We can also view the difference between the files in the repository as they currently are and how they existed at some point in the past. If you have made multiple commits to your repository, use git log to identify one of the earlier commit IDs and use that commit ID as an argument to the git diff command just like this `git diff 5609ff3a`. The first 8 characters of the commit ID should be enough.

9. Diff is also used to view the difference between two commit IDs, that is, the difference between the state of our files at the time the commits were made. Select two commits in your log and run a command similar to this `git diff 8bddd808..5609ff3a`. What do you see? Repeat the command, this time providing a file name as argument `git diff 8bddd808..5609ff3a Welcome.txt`

### Deleting, renaming and moving files

10. The rm command is used to remove files from Git. Identify one of your files, run the command (or a similar command) `git rm Welcome.txt` and then view the status of your repository using the `git status` command. You should see a message similar to the one below

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        deleted:    Welcome.txt
```

11. Now commit the change using `git commit -m "Deleted the Welcome.txt file"`.

12. The easiest way to rename (or move) a file is to use the mv command. Identify one of your files, rename it using a command similar to this `git mv Hello.txt HelloGit.txt`
and view the status of your repository using `git status`. You should see a message similar to the one below

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        renamed:    Hello.txt -> HelloGit.txt
```

13. Now commit the change using `git commit -m "Renamed Hello.txt to HelloGit.txt"`.

14. Note that a file name can also contain the path to its directory, like this: - git mv **oldfilelocation/OldFileName.txt newfilelocation/NewFileName.txt**

### Undoing changes made to our working directory

15. It is possible for us to accidentally change the files in our working directory. Fortunately, we can get back the original copy from the repository.

16. Make a change to one of the files in your working directory and save the change to the operating system.

17. Now take a look at the status of your files using the command `git status`
You should see the following message on screen

```
On branch master

Changes not staged for commit:

  (use "git add <file>..." to update what will be
committed)

  (use "git checkout -- <file>..." to discard
changes in working directory)

        modified:   Welcome.txt

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

Notice that it tells you how you can discard the changes.

18. Run the command `git checkout -- <file>...` to discard the changes and then view the status of your repository.

19. If you have already added one or more files to the staging area, you can use the command `git reset HEAD <filename>` to remove them from the staging area and then checkout the repository version (step 18) to discard the changes made to the file.