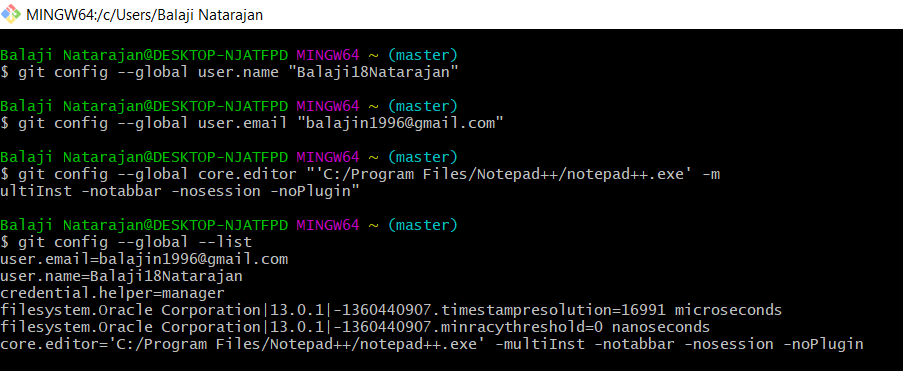
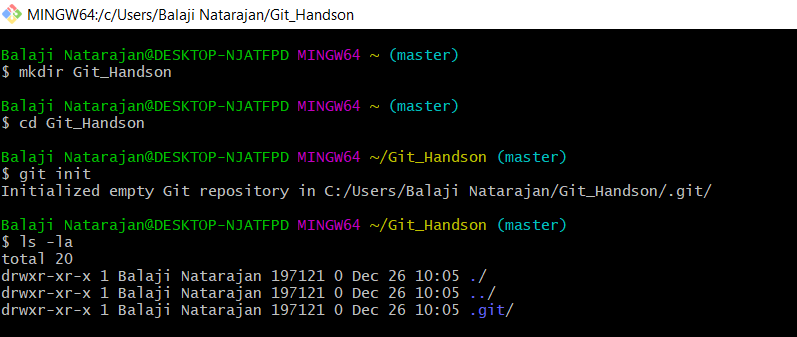
1. Set the global configuration file with your user name, email and editor as Notepad++. List all the properties which you just set.

* git config –global user.name “username”
* git config –global user.email “emailid”
* git config –global core.editor “notepad++”



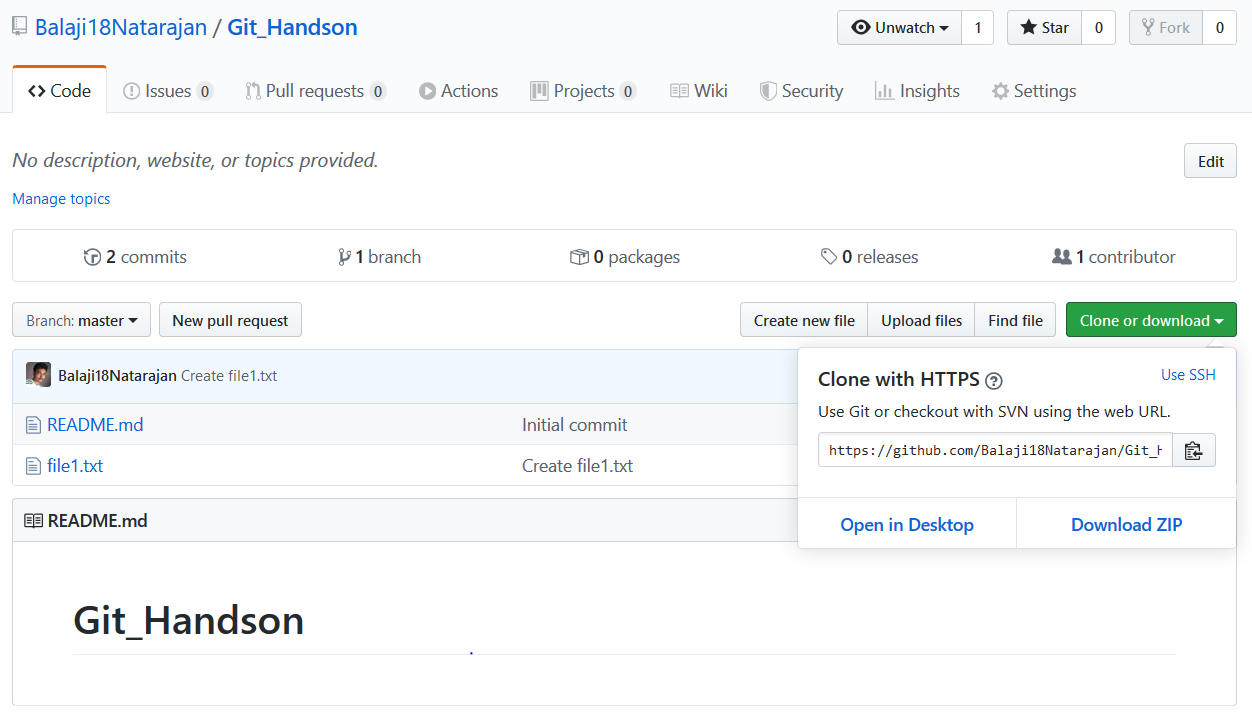
1. Make a fresh Git project

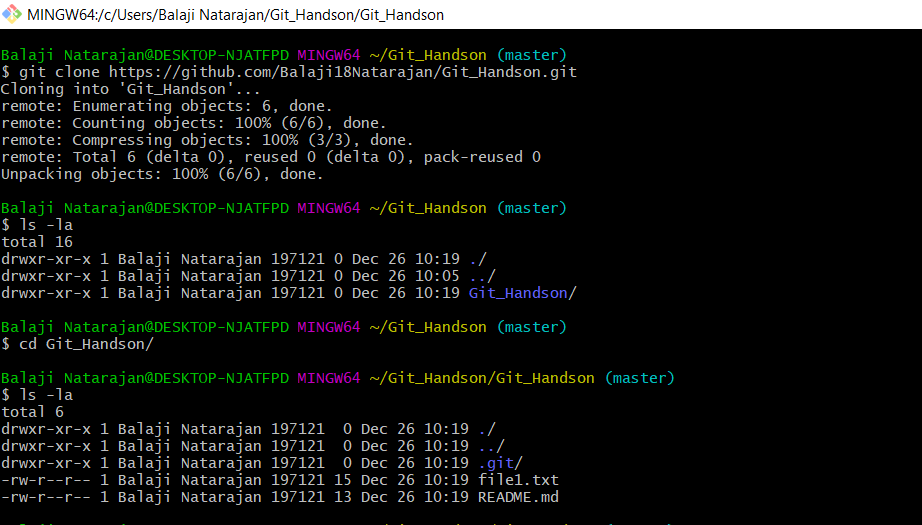
* git init



1. Create a Github account (Or use the account if already registered). Clone a project from the remote repository to your local repository

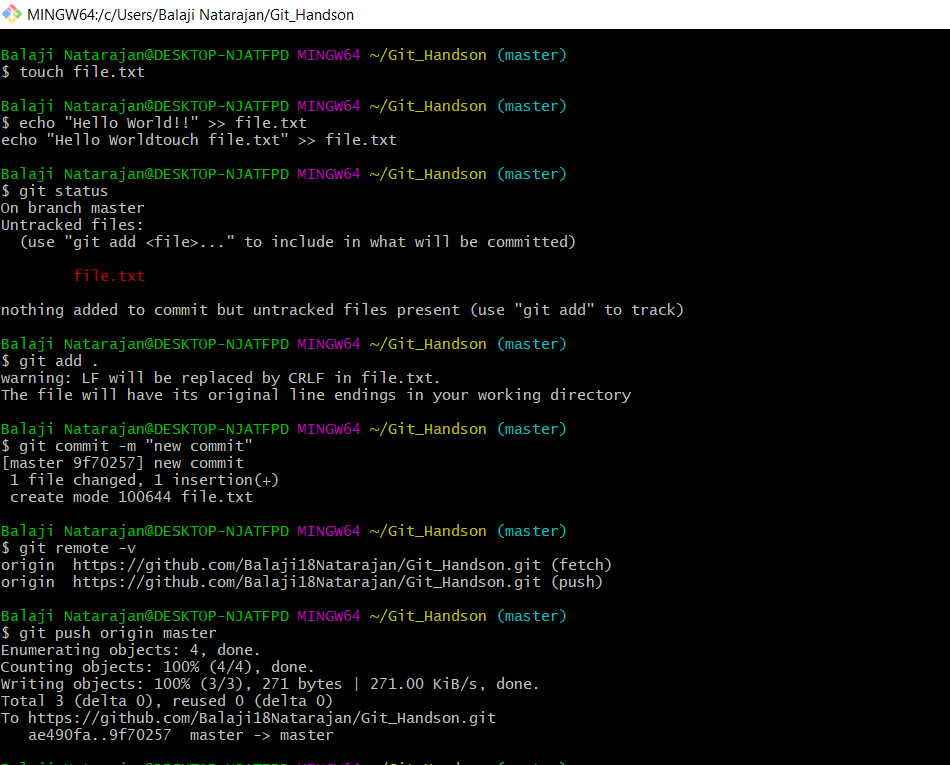
* git clone <git url>



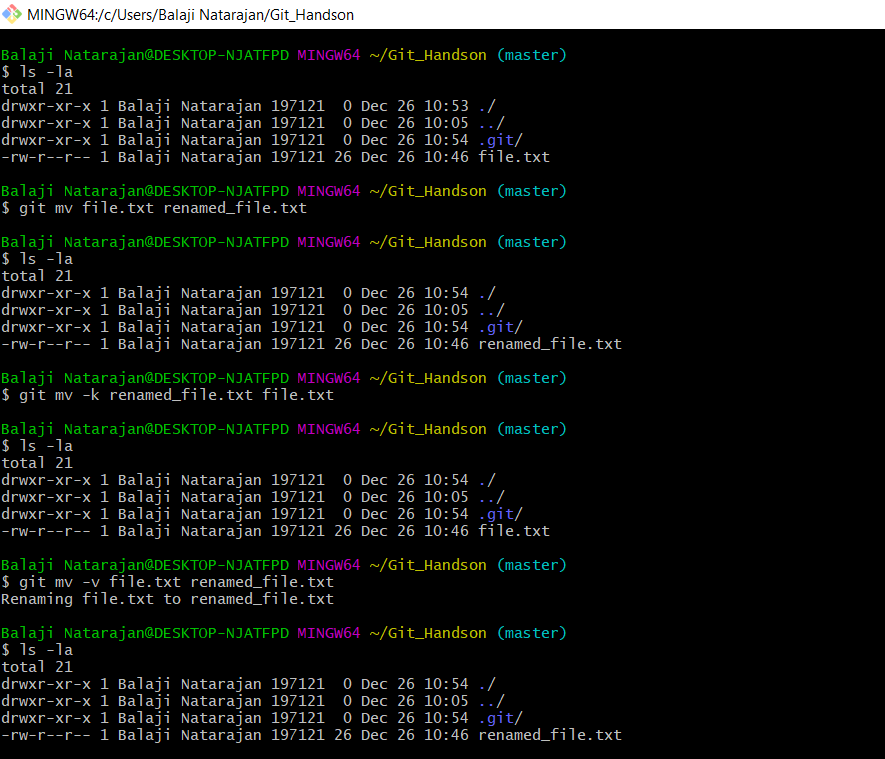


1. Push the project created in assignment 2 to the remote repository.

* git add .
* git commit -m “commit message”
* git push <remote> <branch>

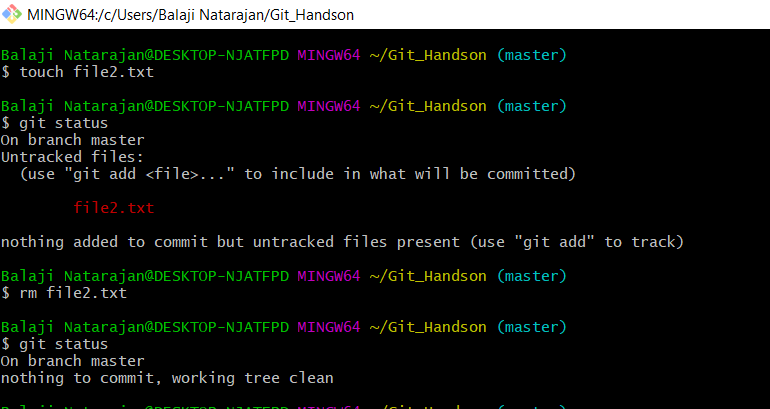


1. Use the different ways of renaming and moving files

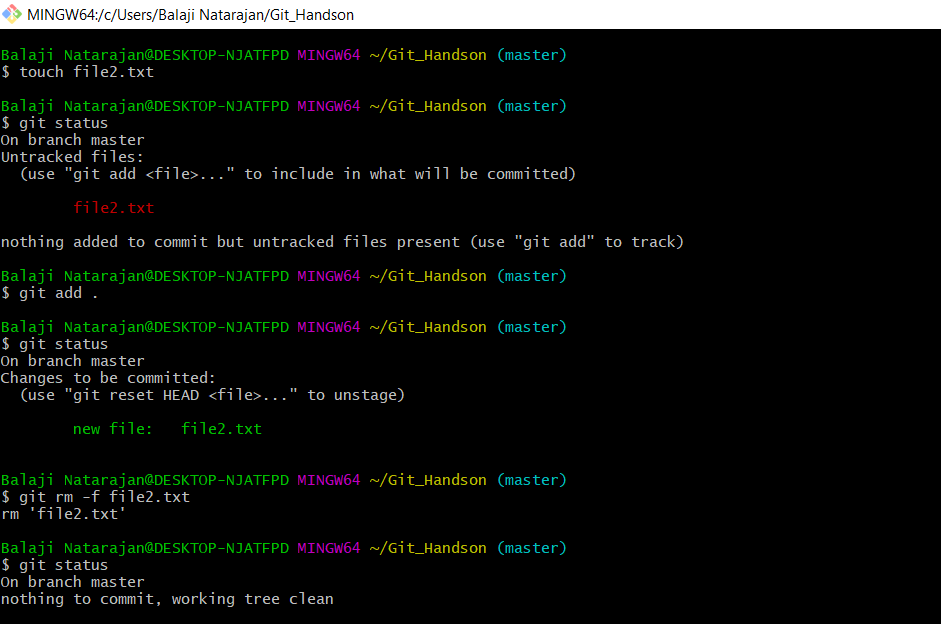
* git mv [-v] [-k] source destination  
    
  

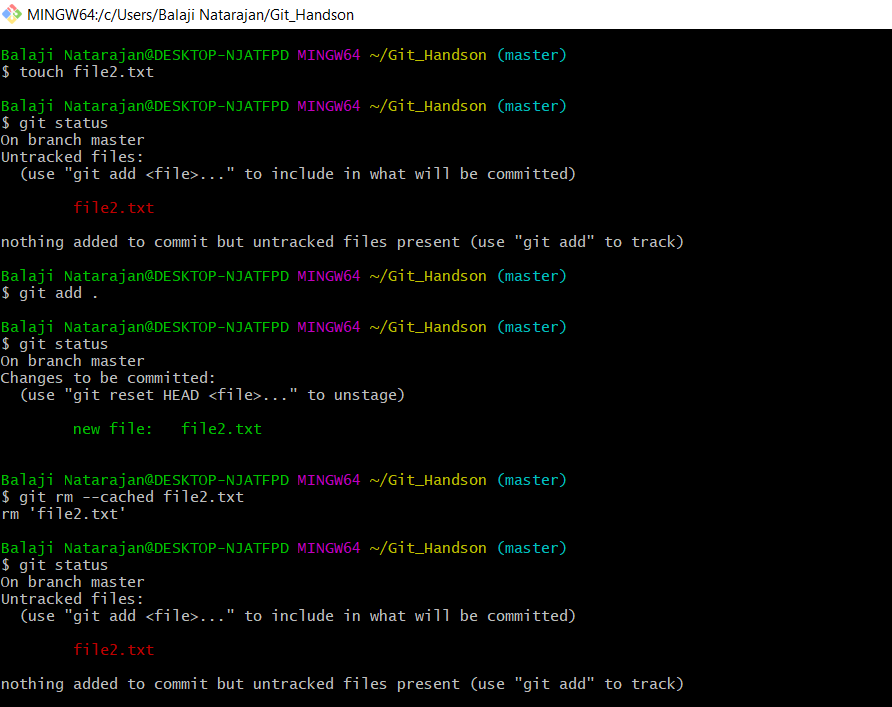
1. You just created a new file, but then you decided that the file is to be removed. How do you delete this untracked file?

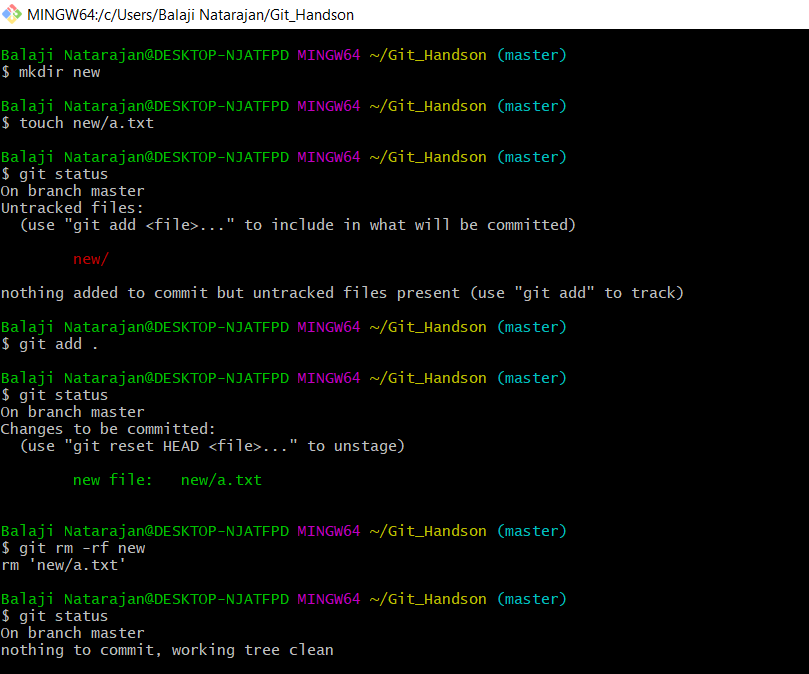
* rm new\_file

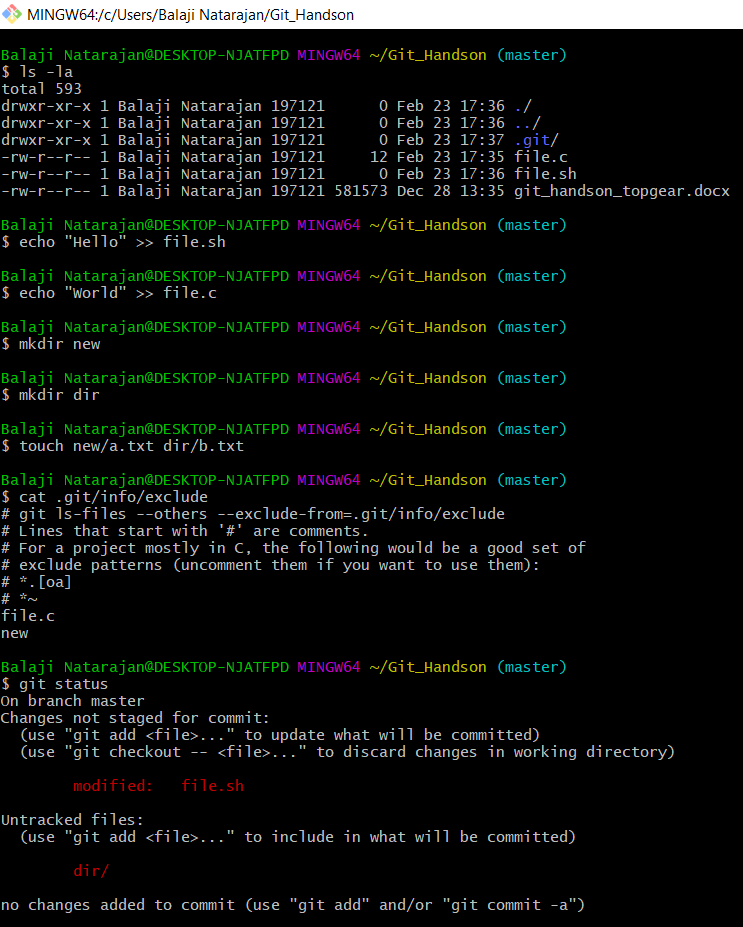


1. Demonstrate the following:
2. delete of a tracked file  
   => git rm -f <filename>



1. backing out staged deletion  
   => git rm –cached <staged filename>  
     
   
2. recursive deletion  
   => git rm -rf <repository name>



1. You do not want to push certain folders/files of your project; how do you manage this?  
   => The files that are to be ignored by Git from being pushed to the project are mentioned in the info/exclude file present in .git directory.

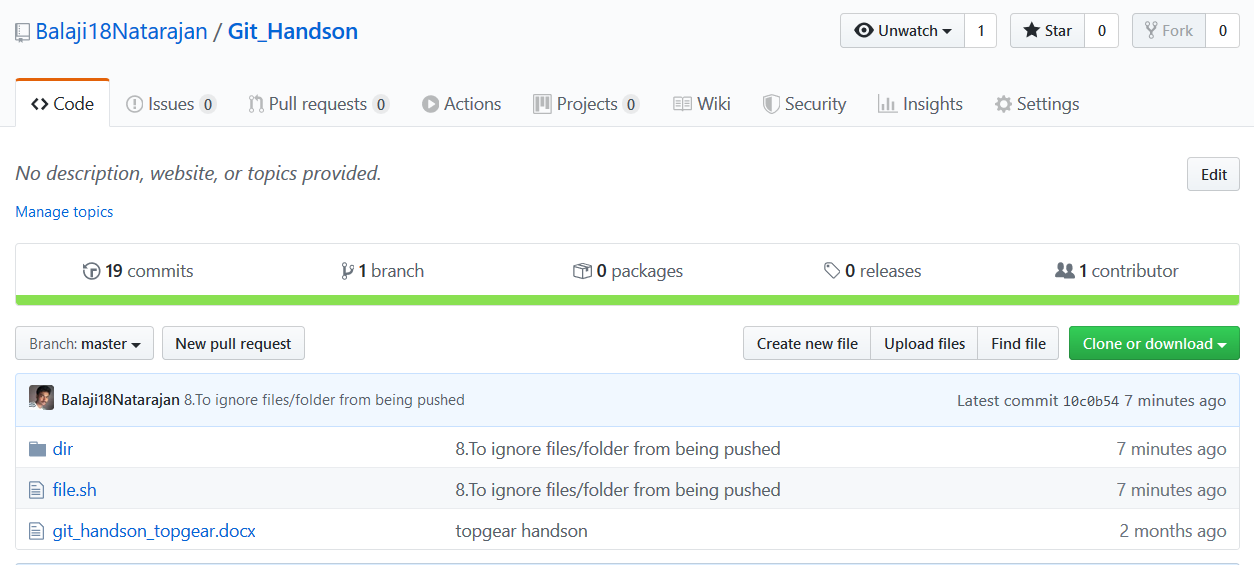
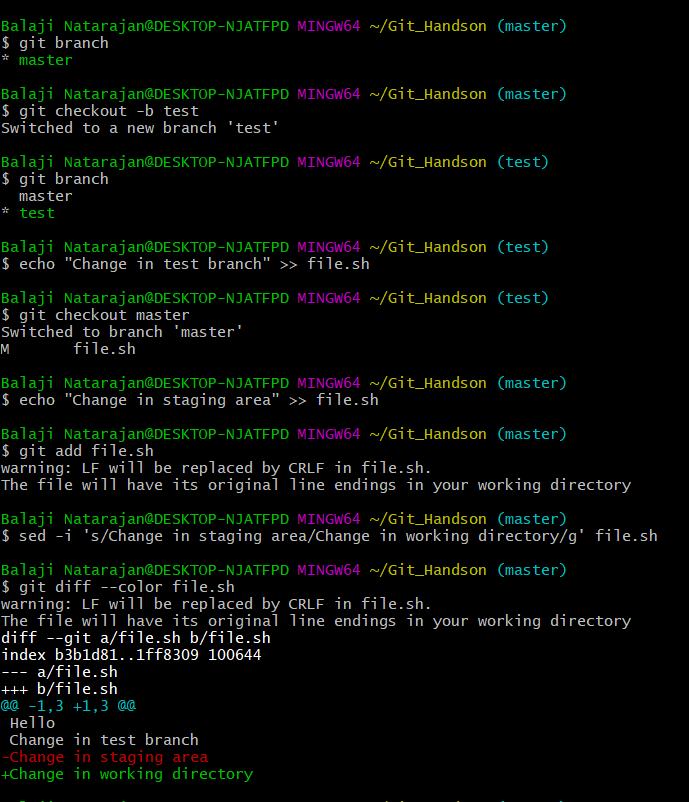


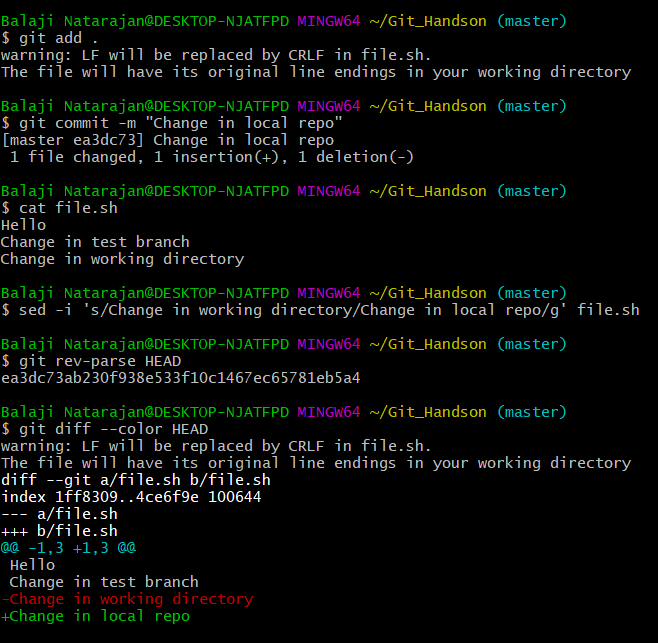
Figure file.c and new directory are not pushed to the project

1. Create a branch called “test”. Make some changes in the master branch. Let there be some changes in the working directory and some in the staging area. Make some changes in the test branch as well. Issue the command to show the differences for

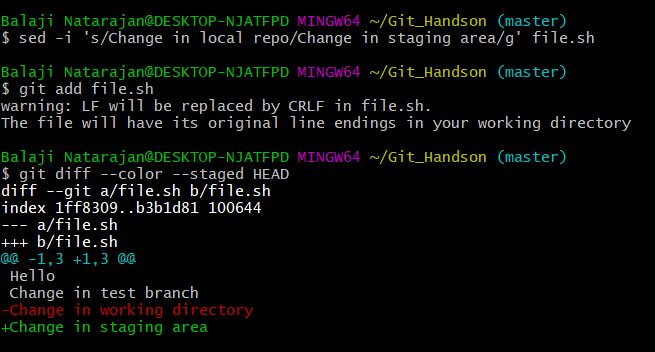
* To create a new branch  
   -> git checkout -b <new branch>
  1. Working directory vs Staging area  
     => git diff --color

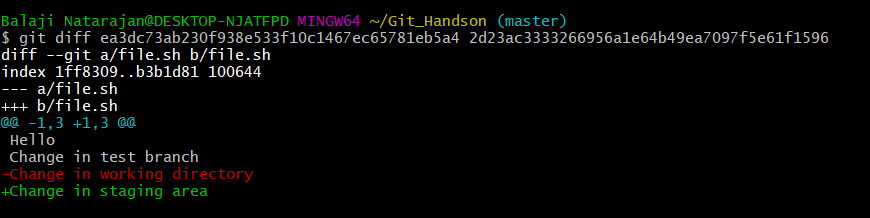


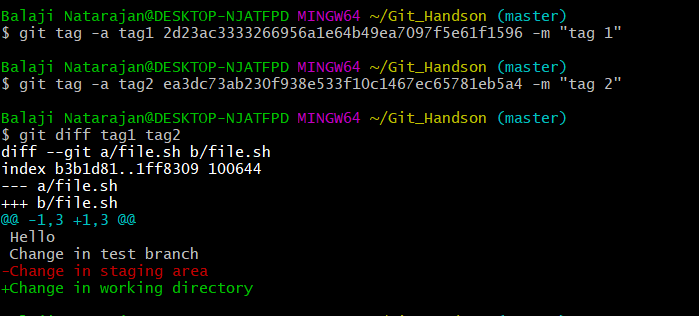
* 1. Working directory vs Local Repository  
     => git diff –color HEAD



* 1. Staging area vs Local Repository  
     => git diff –color –staged HEAD

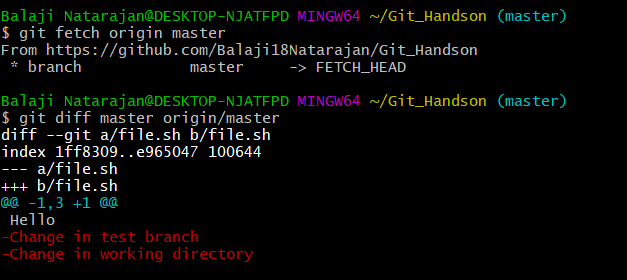


* 1. Between two commits  
     => git diff <commit id1> <commit id2>
  2. Between two tags  
     => git diff <tag 1> <tag2>

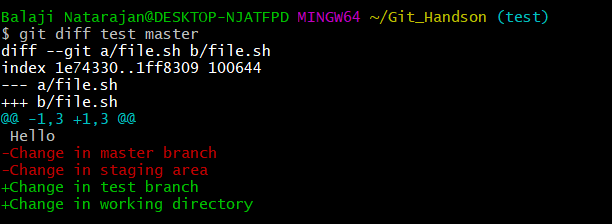


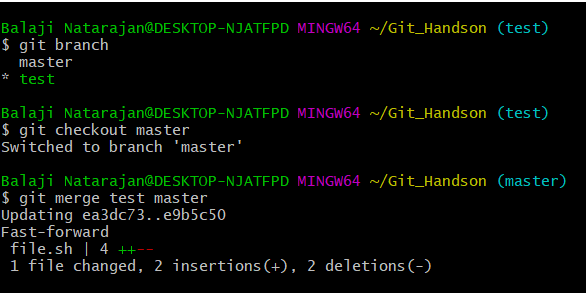
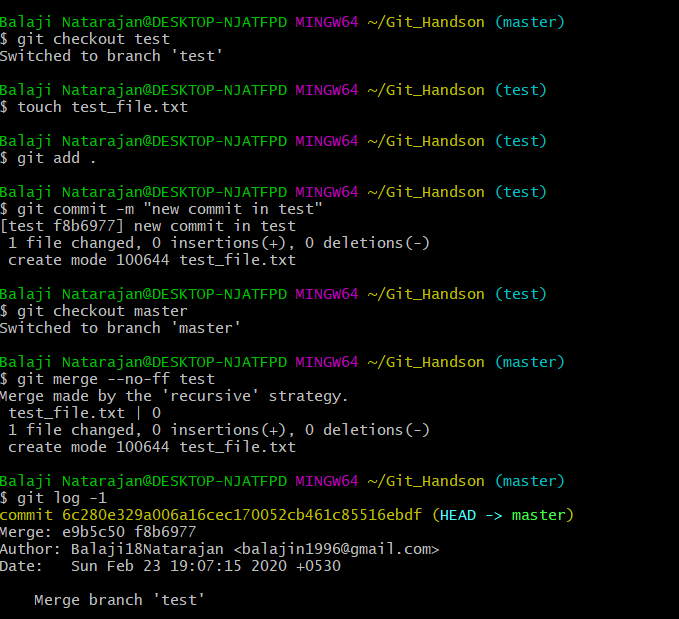
* 1. Local vs Remote Repository  
     => git fetch (to get the remote branch)

=> git diff <branch> origin/<branch>



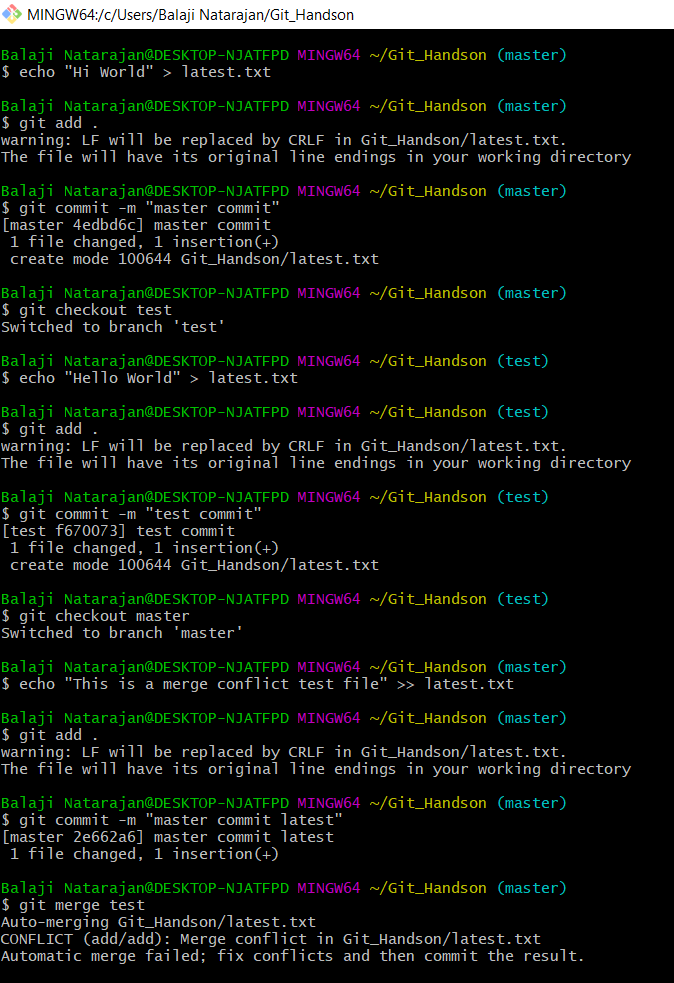
* 1. Master branch vs test branch  
     => git diff <branch1> <branch2>



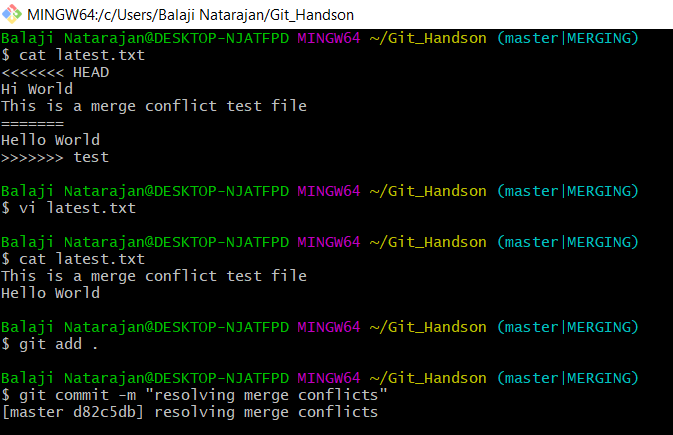
1. Merge the changes from test branch to master branch.
   1. FastForward merge  
      
   2. Disabling FastForward merge  
      => git merge –no-ff <branch>  
      
   3. What is the difference between option a and option b

=> In option (a) with fast forward merge there is no merge commit generated while by disabling it with --no-ff option there is a merge commit generated which can be kept as a record.

11. Create a merge conflict situation. Resolve the conflict and merge the changes between the branches.

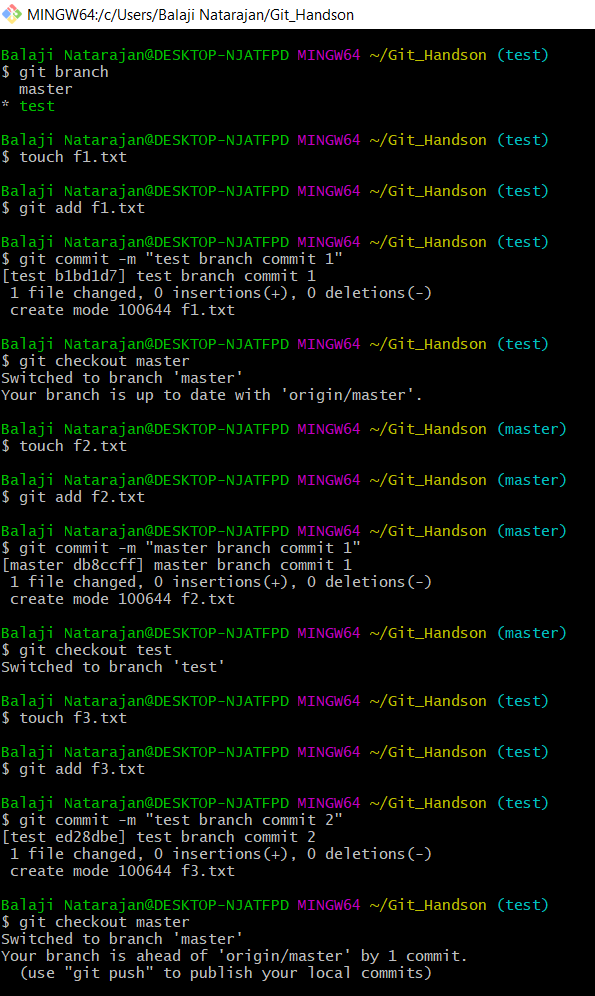
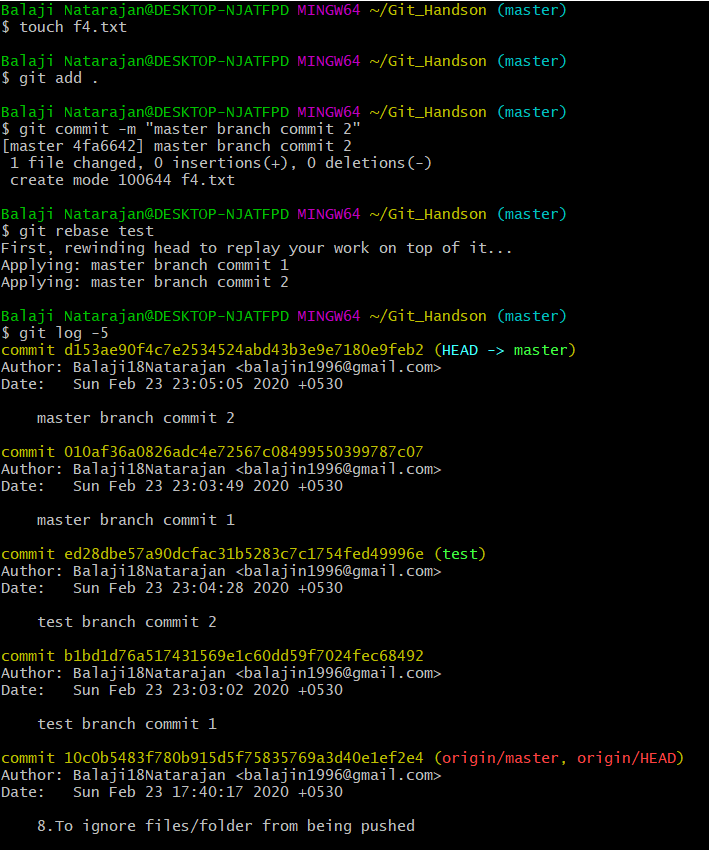


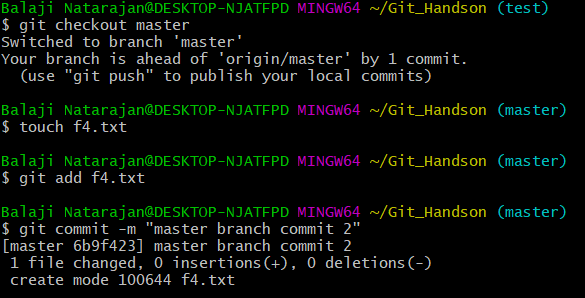
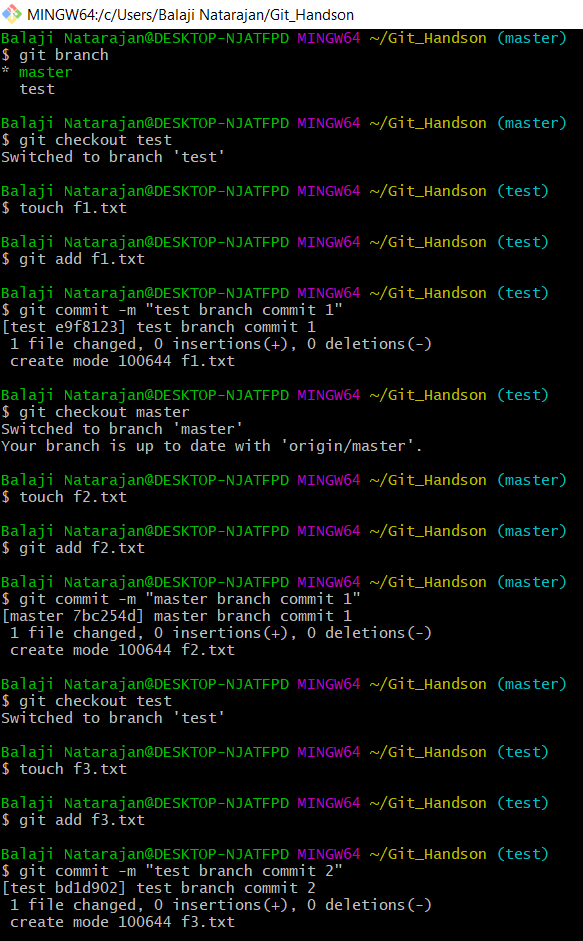
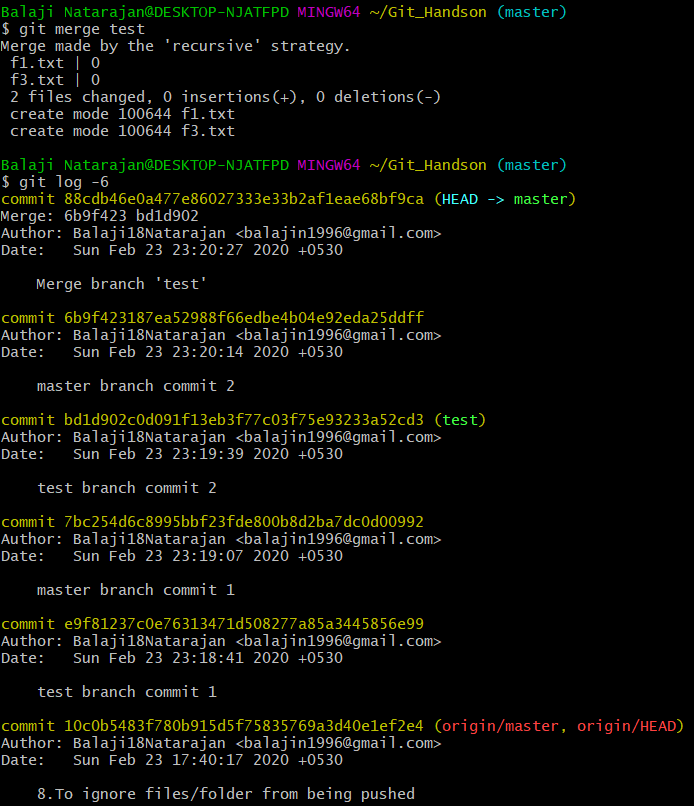
Resolving Merge conflicts



1. What is the difference between merge and rebase, demonstrate with the eg.

=> Merge will result in the combination of commits maintaining the history of commits without changing their timeline. Merge will create an extra commit in the history. While Rebase will put the series of commits done in the feature branch to the end of the master branch latest HEAD without any extra commit providing a linear set of commits. The entire history is rewritten here, the timeline is not followed in the commit history which could be trouble during audit or for other developers.

Rebase :   


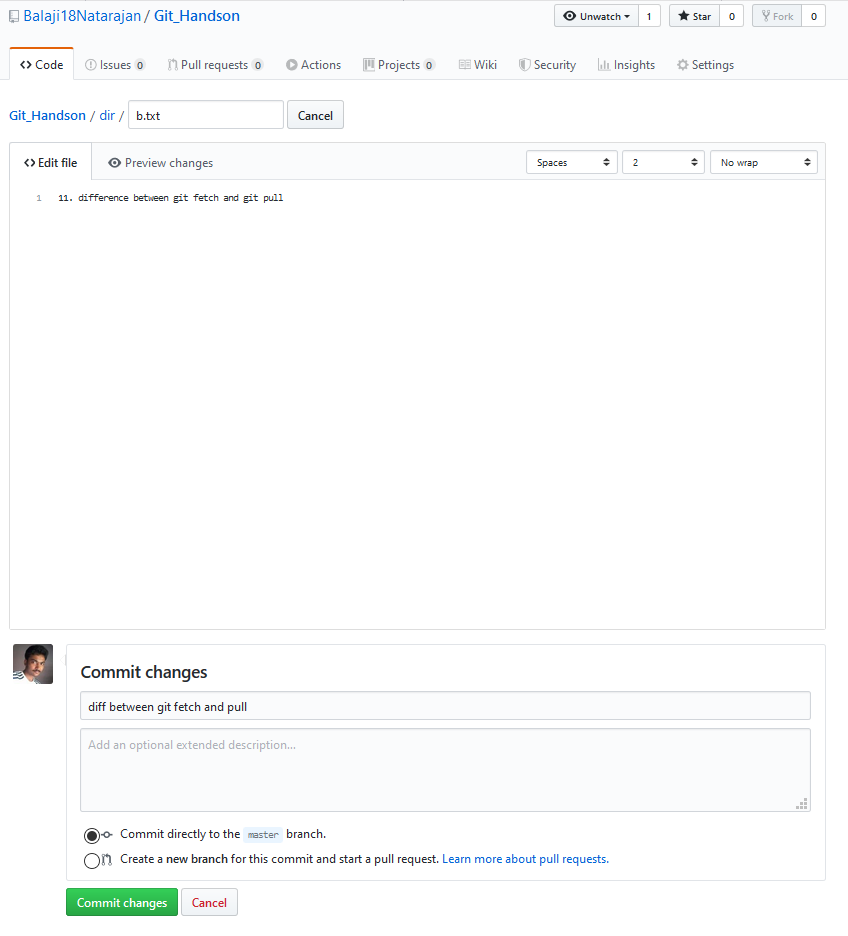
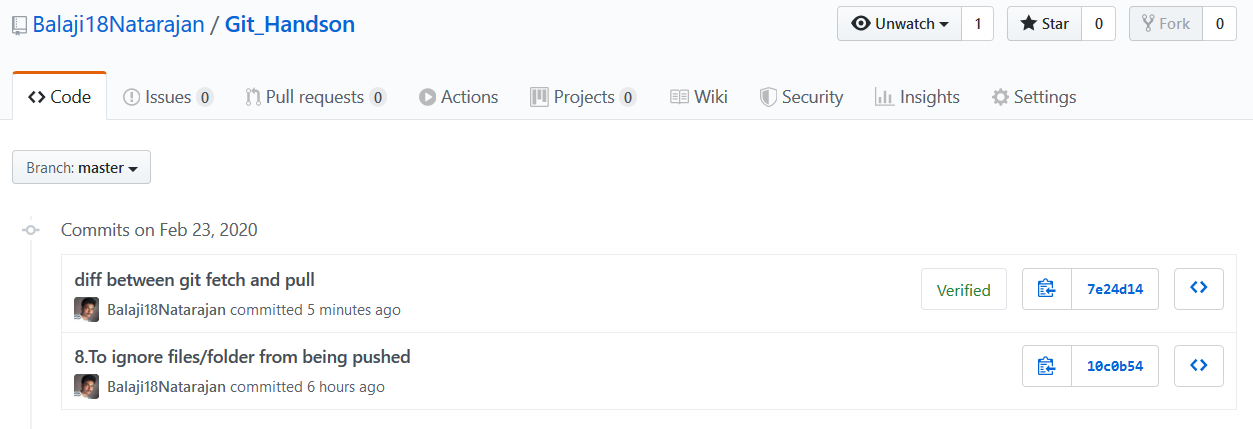
Merge  
  


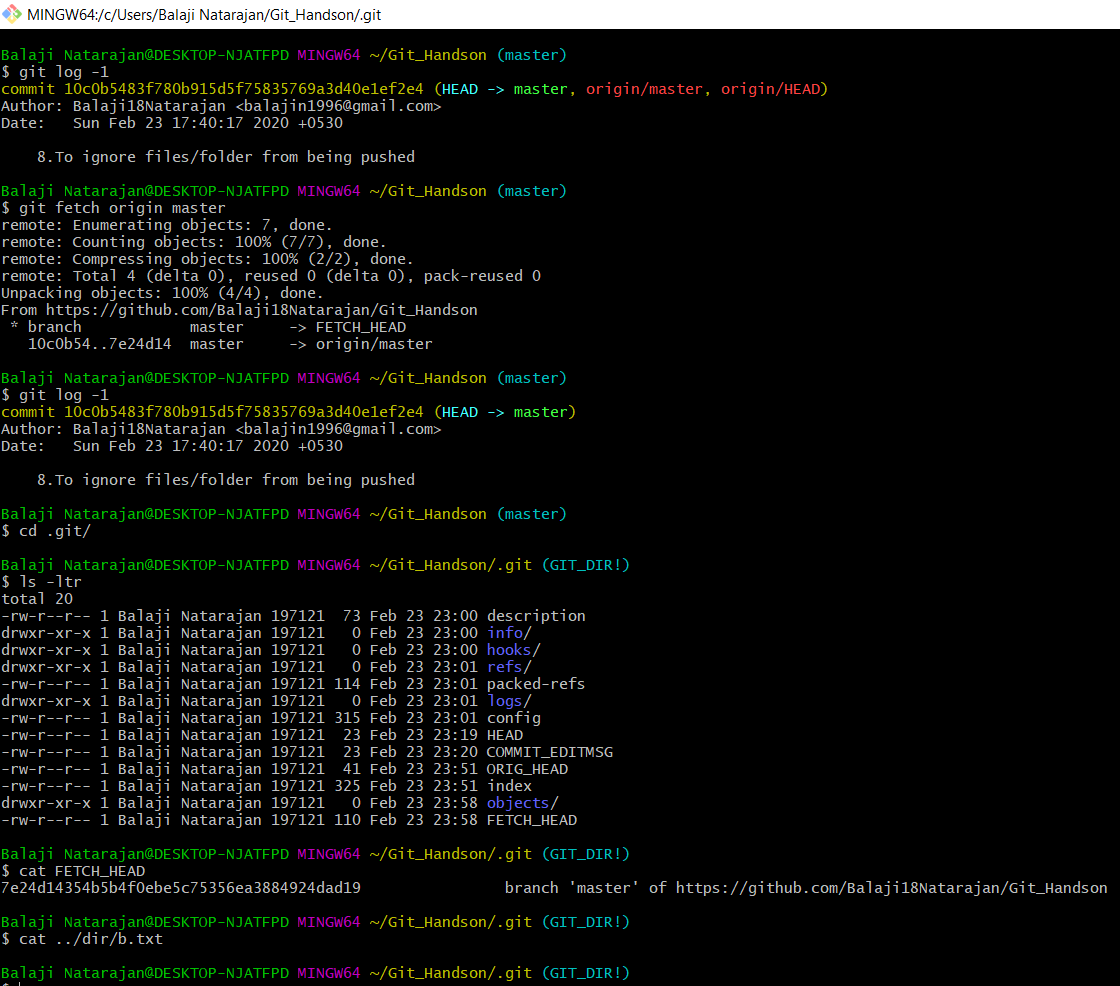
1. With an example, demonstrate fetch, clone and pull. What is the use case for these operations? Are they same, different? Explain

=> fetch, pull and clone commands aren’t the same.   
Fetch downloads the remote data of a repository. It does not affect the local working directory.

While pull does two operations together, i.e., fetch+merge. Git pull command fetches the remote changes and merges them to the local repository.

Fetch or pull as to be given in a local repository where there are references to the remote repository. Git clone command clones a repository into a newly created directory where the default references to the remote repository is maintained. It can also be said that clone is a combination of command like “git init, git remote add url,and git pull”.

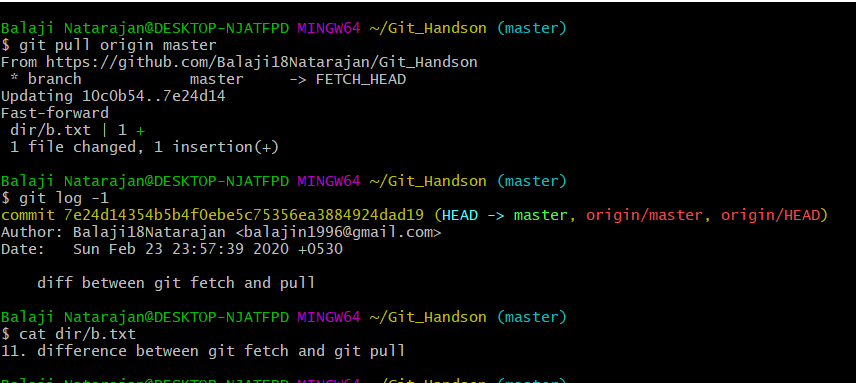
Example:  
A file in the remote repository is being updated with a commit.  
 



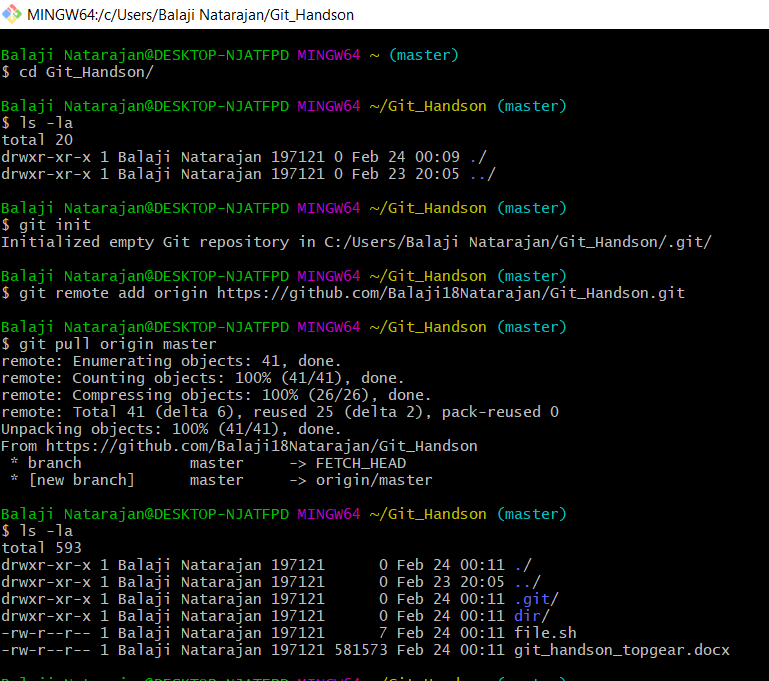
On performing a git fetch the local working directory isn’t affected while the HEAD is and .git directory is updated with the remote changes, but is not merged with the local copy.

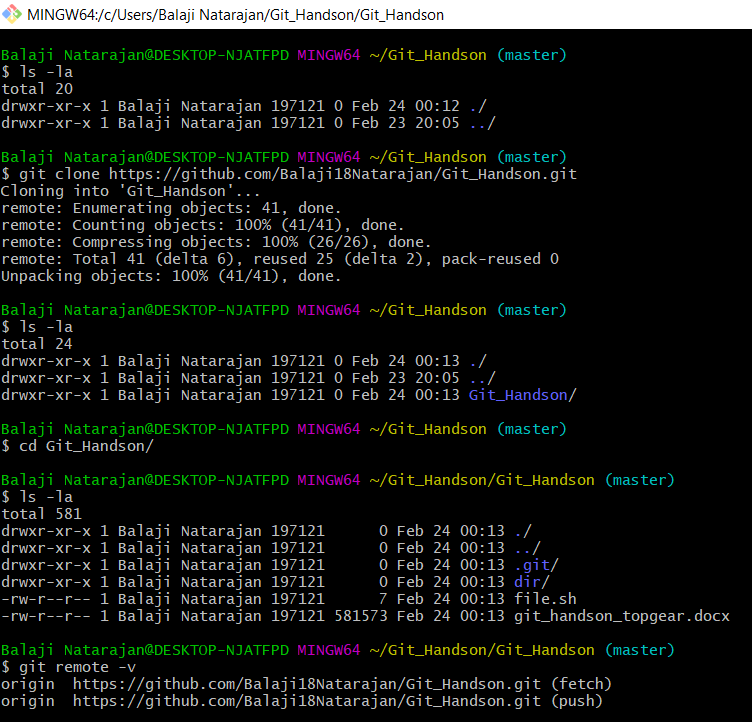
Git pull:

Git pull = “git fetch+git merge”



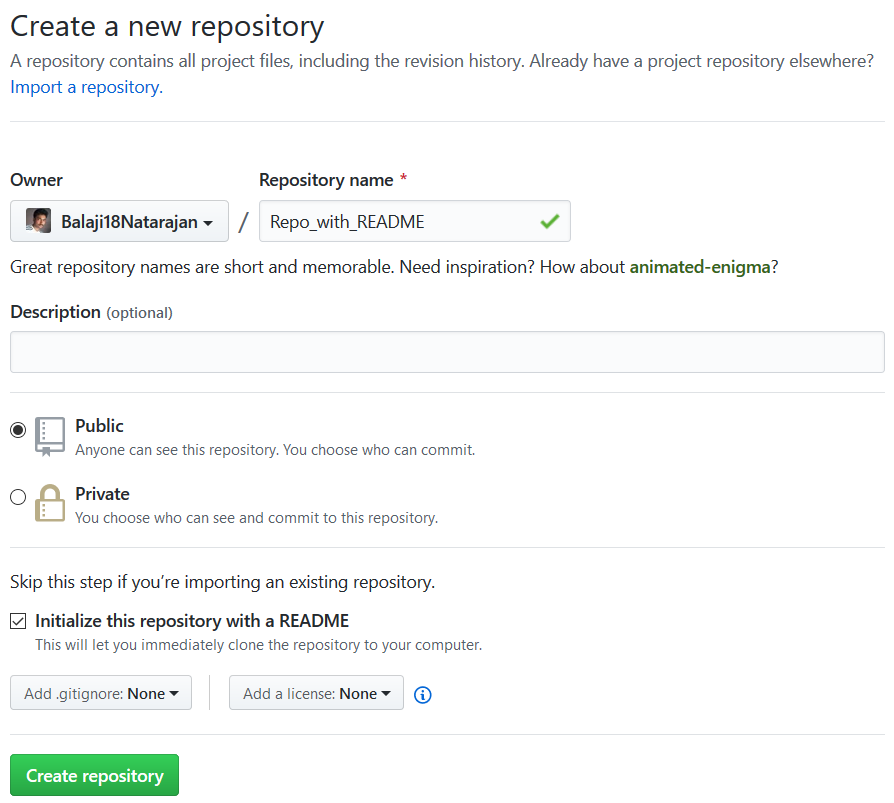
Git clone = git init + git remote add origin <url> + git pull

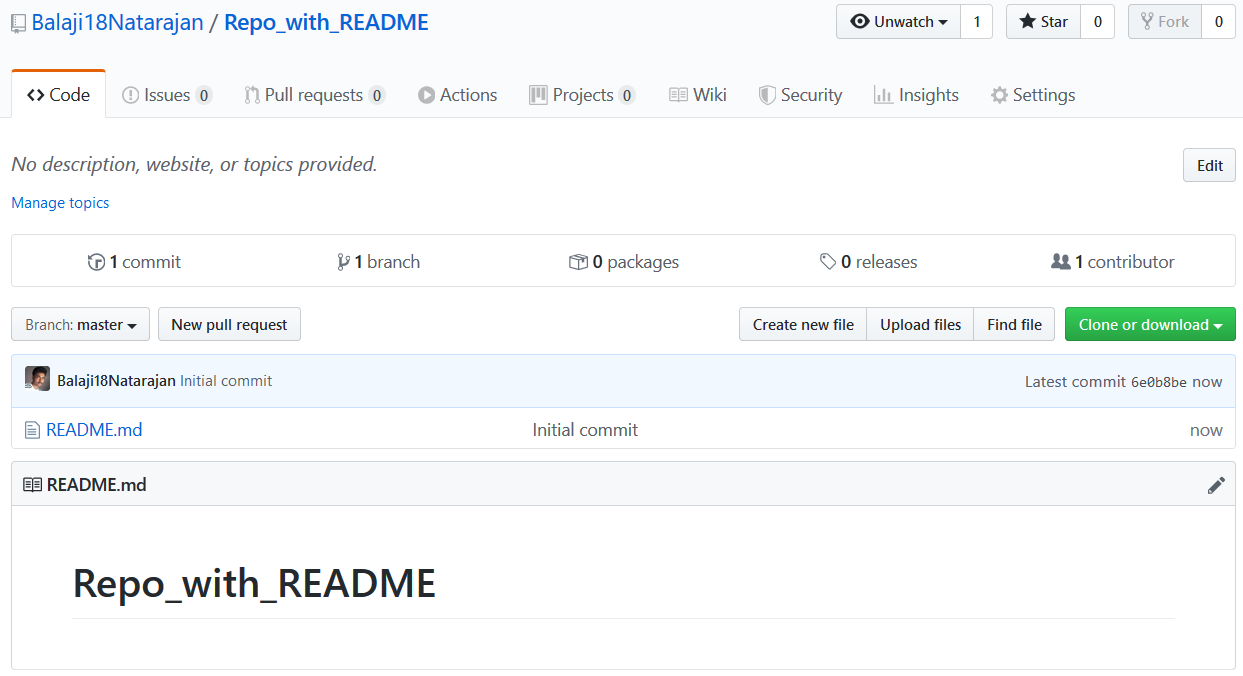




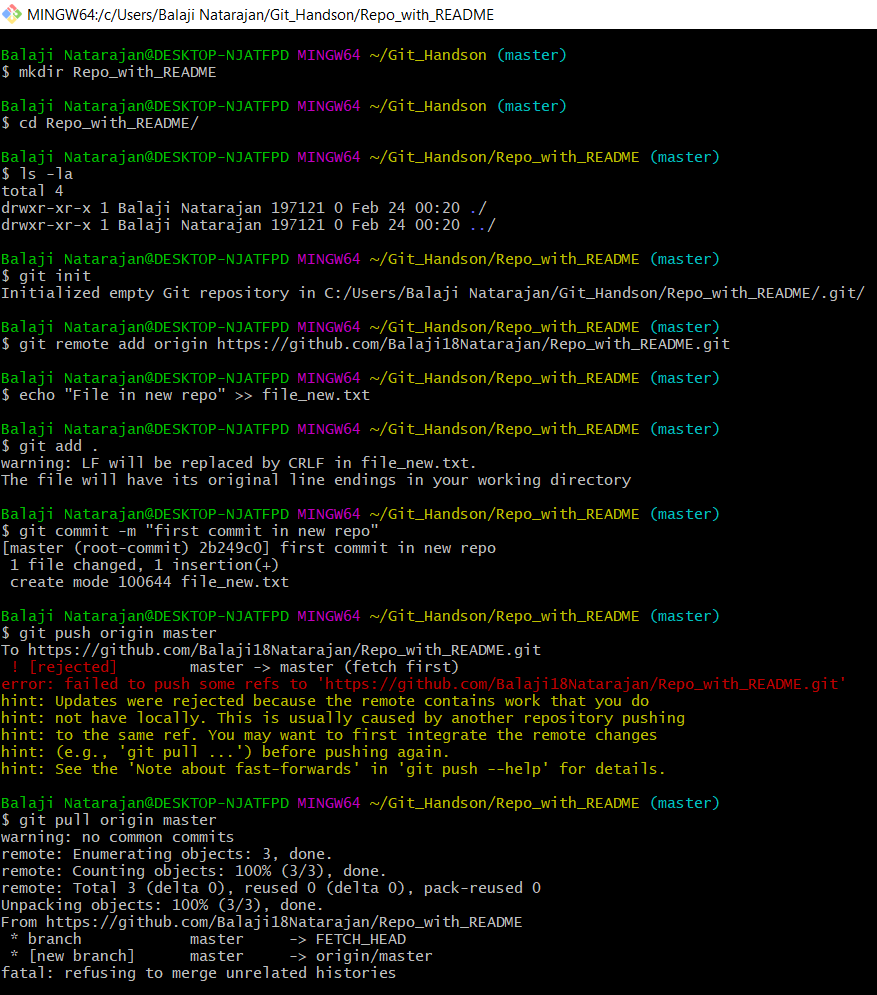
1. Create a new repository in Github, with a README file. While pushing to the remote repository, if the remote branch is ahead of the local repository (new file is added in remote repository, which is not there in local repository) and pull is failing, how do you solve this problem?

=> git pull –allow-unrelated-histories origin master will solve the problem





Recreating the problem statement in local



Solution:

