**Because I love GIT…….**

1. Configure your UserName and Email in GitBash

>git config - -global user.name “Neha Gohil”

>git config - -global user.email nehagohil@gmail.com

1. To get/retrieve the UserName and Email from GitBash

>git config user.name

>git config user.email

1. Basic UNIX Terminal Commands

>ls : List the contents of your current directory.

>start . : It will open the file explorer.

>ls [folder name] : List the contents of the mentioned folder.

>ls [folder\_path] : List the contents of the selected path folder as, we can’t access subfolders directly.

>pwd : It will give you the current path location.

>cd : To change or move between the directory.

>cd .. : To go on the one step backward directory.

>touch [filename]: It will create new file in the directory.

>touch [f1 f2 f3] : It will create multiple files at the same time in the directory.

>touch [path to create file in the specific folder except home directory]

>mkdir [dir name] : It will create new directory on the current path.

>rm [filename] : It will delete file or files permanently.

>rm -rf [directory name] : It will delete the folder permanently.

1. What is GIT repository?

A GIT repo is a workspace which tracks and manages files within the folder. For every new project, we need to create new GIT repo.

Ok…Now enough of basics and move towards the GIT Commands…

>git status : gives information on the current status of a git repo and its content.

>git init : It will create a new git repository with a .git folder. Hidden folder will be created in the folder.

(Always check for the git status before creating any new git repo and do not init a repo inside of a repo)

>ls -a : It will list the hidden folders of the directory.

(Git tracks a directory and all nested subdirectories)

>git add [file1 file2]: It will add the files to be committed. It will add those files to the staging area.

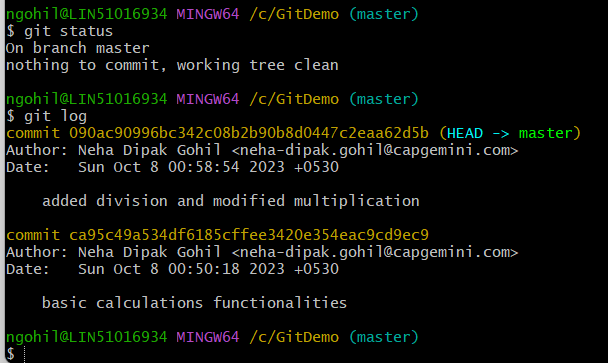
>git add . : To stage all changes at once.

GIT Commit

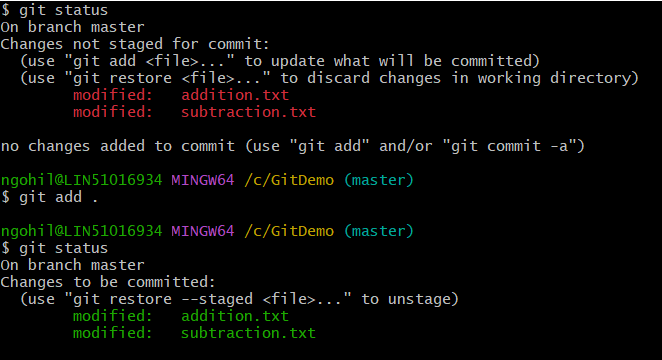
>git commit : commit all staged changes, it also opens text editor and prompts for commit message.

>git commit -m “My message” : -m flag will allow us to pass inline commit message rather than launching a text editor and commit all staged changes.

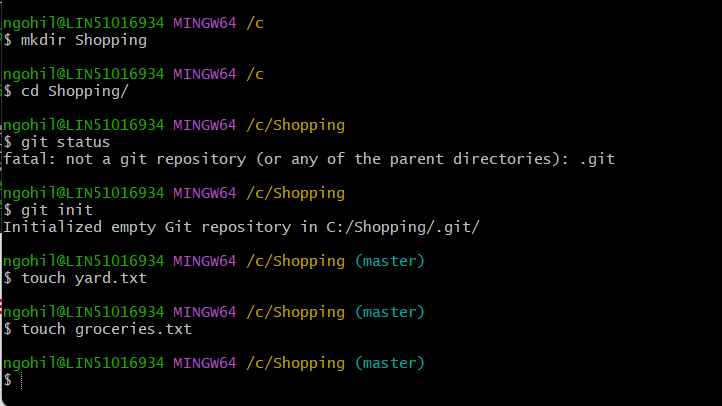
>git log : it lists all the committed history for the repo.

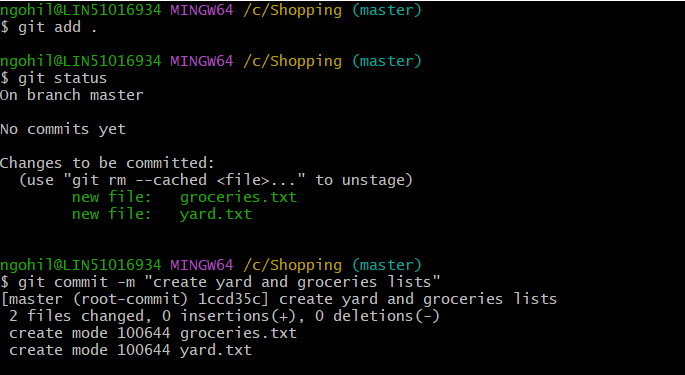


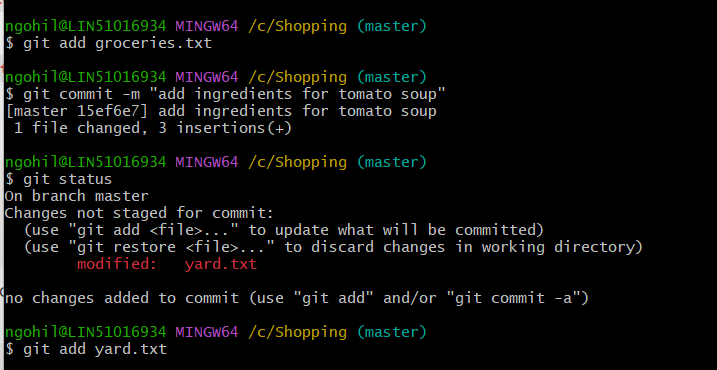
>git add . : to add all the changes at once

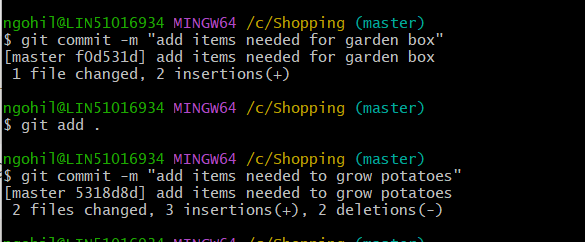


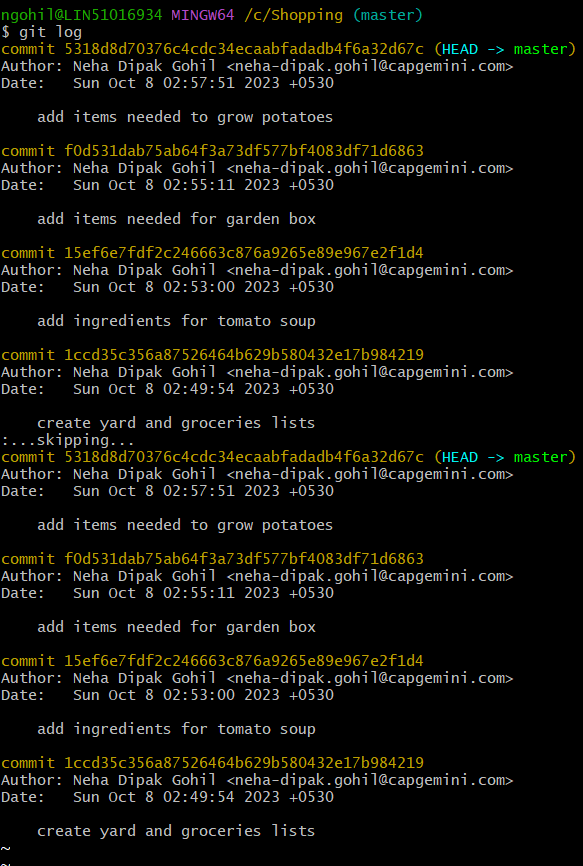
Committing Exercise









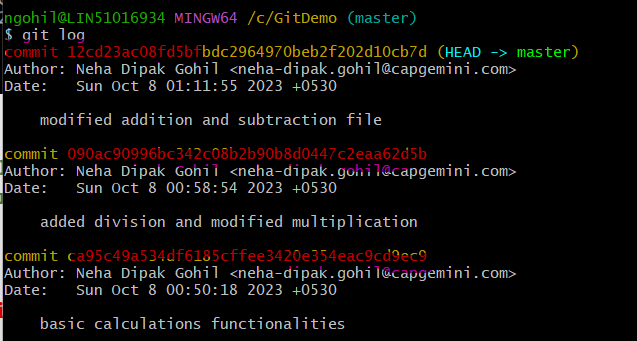


Atomic commits:

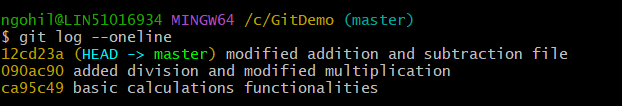
A commit should encompass a single feature, change or fix. This makes rollback or undo changes much easier. Also makes code or project easy to review.

Other Git log variations for easy logging display

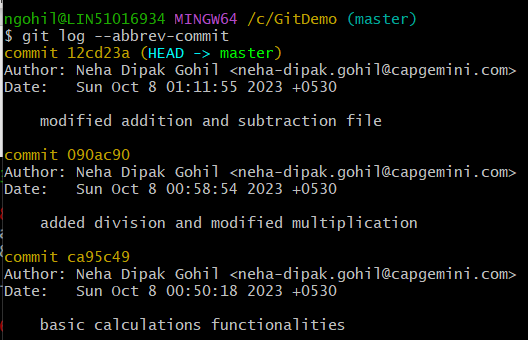
>git log :



>git log - -oneline



>git log - -abbrev-commit



Amending Commits

After commit sometimes we realise that, we forgot to include one file or we wrongly typed a commit message. So, instead of making a new commit we can redo the previous commit [This won’t work after 10 commits, it is only for the just previous commit]

Below are the steps to Amend the commit

>git commit -m “commit message”

>git add <forgotten file>

>git commit - -amend

It will undo and update the previous commit

>git commit -a -m “commit message”

It will add and commit all the unstaged changes.

Git Ignore

We can ignore files like secrets, encrypted files, important files and folders (end with \)using git ignore. We need to create .gitignore file in the root directory and add list of files which we want to ignore.

Then add and commit .gitignore file. If we update any file listed from the .gitignore file , it won’t get tracked in the git .

Let’s go for the Branching….Signature icon of GIT…

Branching enables us to create separate context and we can work on different ideas/functionalities parallelly.

The Master Branch

This is the default branch when we create a new git repo. It doesn’t have any special rights, it is just a simple branch like any other branch.

Head -> master

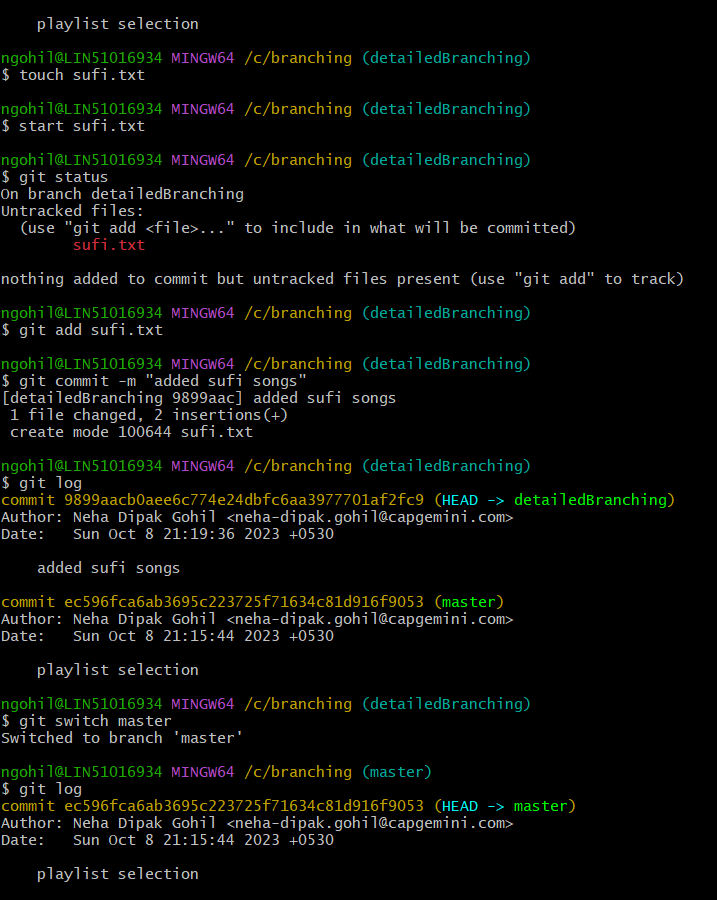
Head is simply the pointer which points to the current location in your repository. It always points to the latest commit we made on the master branch.

>git branch : View existing branches. it gives list of git repos we have in that folder. The active branch is with the asterik sign (\*) ahead of branch name.

>git branch <branch\_name> : create new branch based on the current head pointing branch.(master branch mostly).

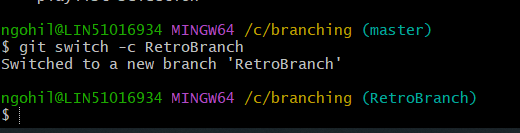
>git branch -v : It will give information of the past commits and last commit to the current branch.

>git switch <branch\_name> : To switch between the branches.



>git checkout <branch\_name> : Does exact same thing like switch.

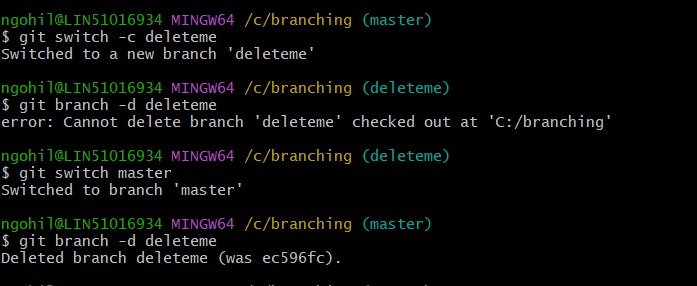
>git switch -c <branch\_name> : It creates new branch and switches to that branch.



Delete Branch

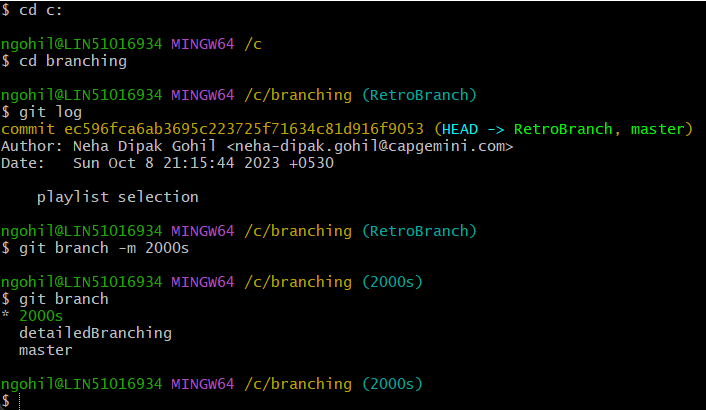
>git branch -d <branch\_name> : It deletes the branch.

>git branch -D <branch\_name> : It allows to delete the branch forcefully.

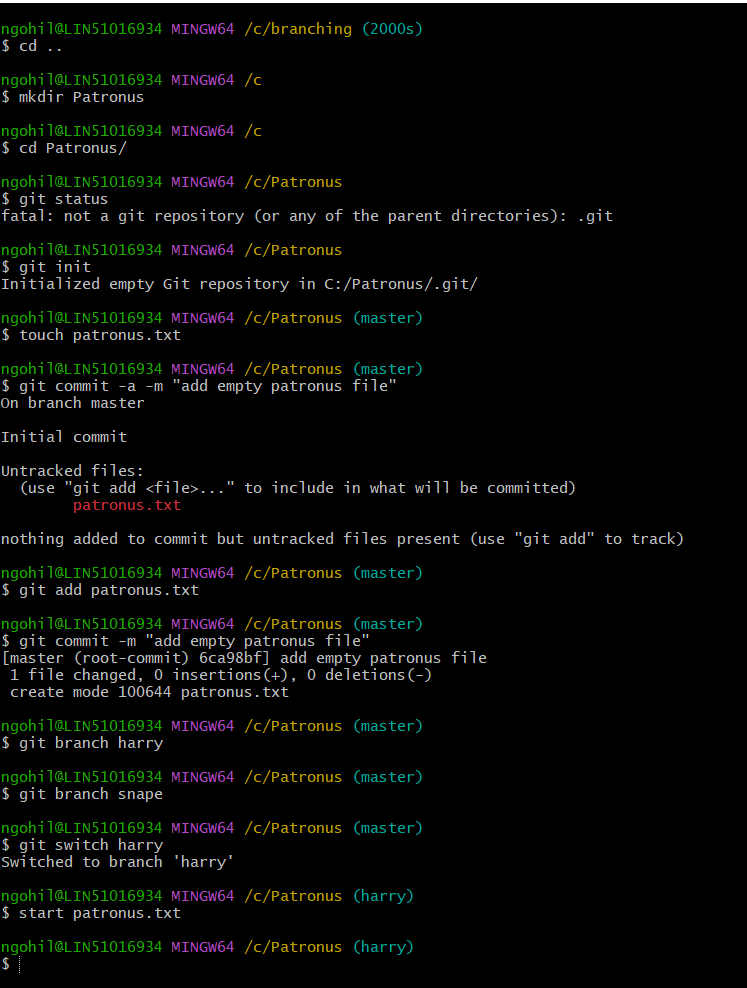


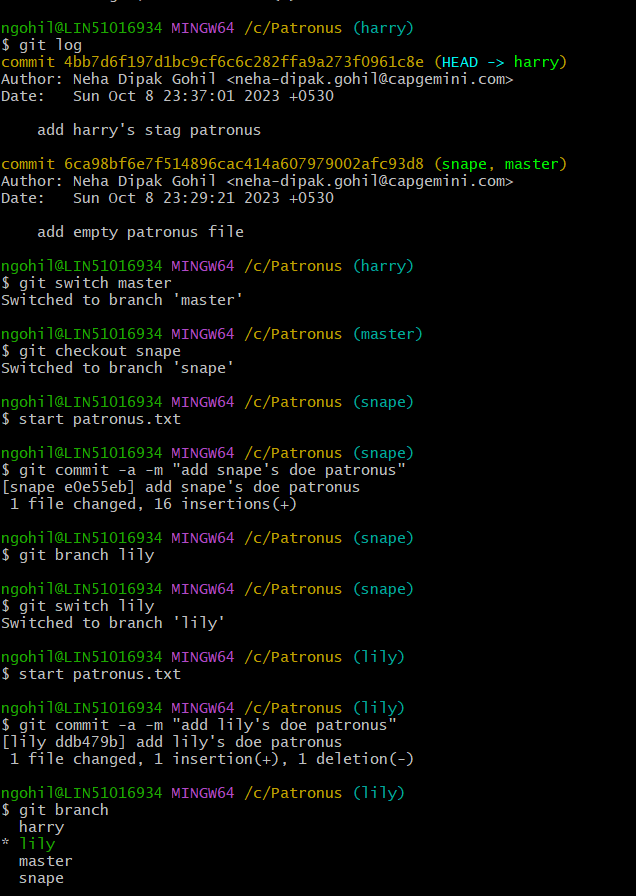
Rename Branch

>git branch -m <branch\_name> : we can rename the branch name. First, we need to switch to that branch and then use this command.



Branching exercise







Merge..Merge..Merge.. Let’s Merge the branches 😊

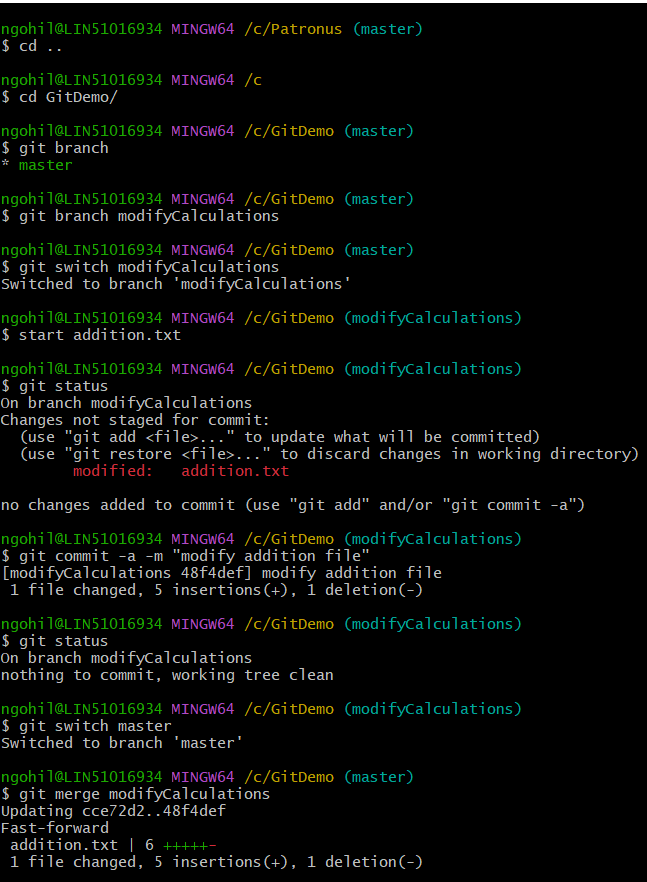
To merge the branches, switch to the branch where you want to merge the changes that is the receiving branch.

>git merge <branch\_name> : merge changes from the specific branch into the current branch.

>git switch master : switch to the master branch.

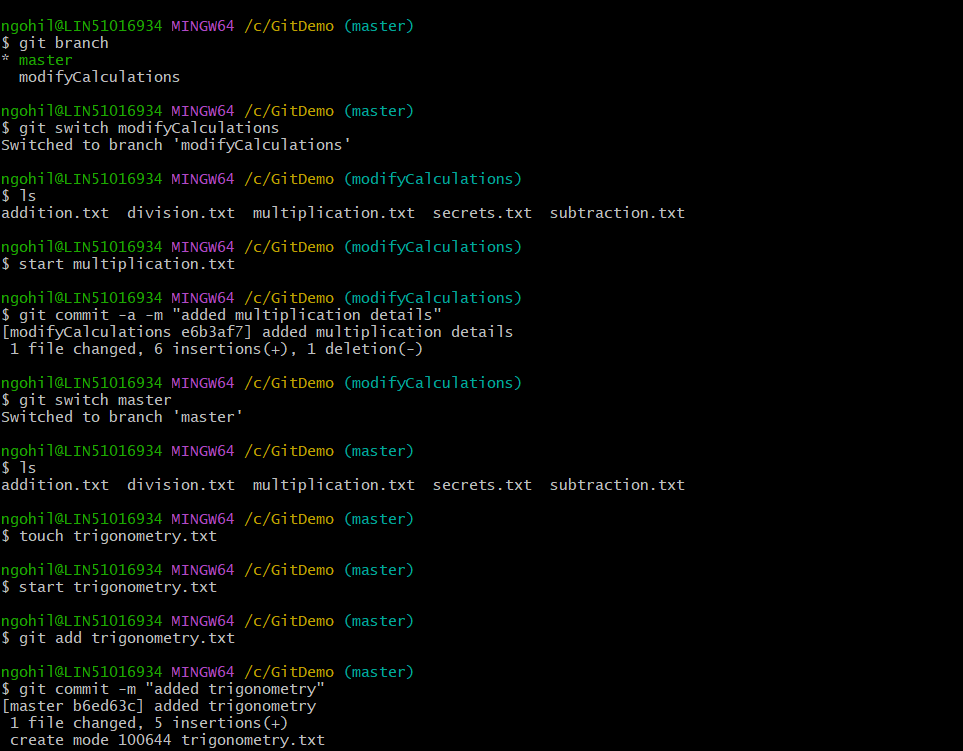
>git merge devfix : merge devfix branch to the master branch.

[This is also called Fast forward merge]



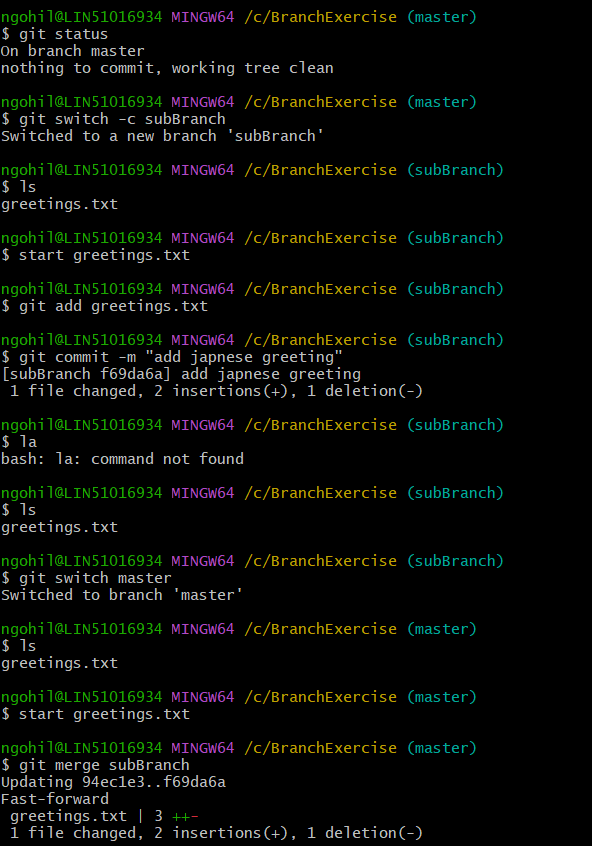
Merge commits

When we commit new changes into the master branch and try to merge the other branch into the master (which don’t have latest master commits) then, Git will automatically commit those changes to the other branch and then, using previous steps we can merge the branches.

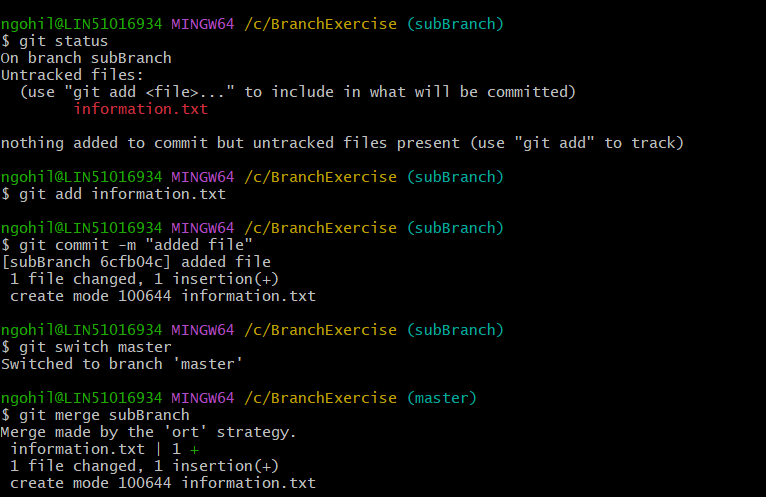


Merging Exercise

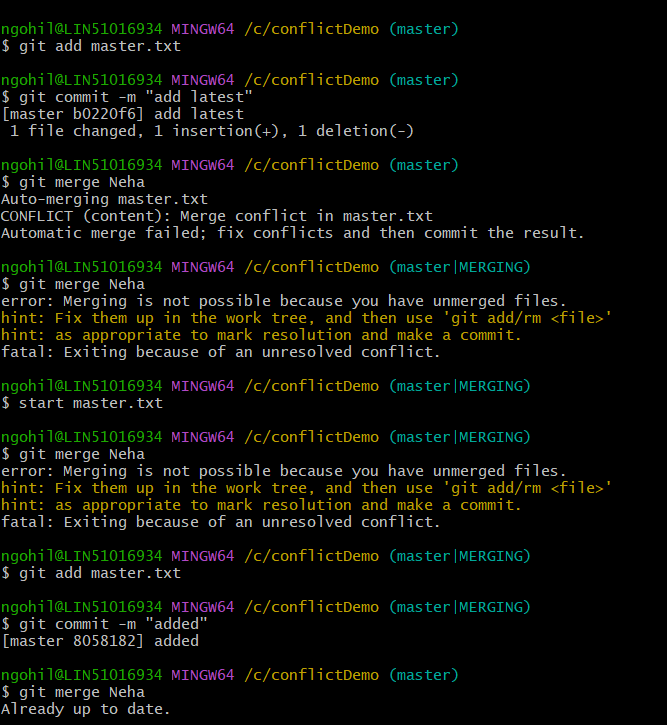
1. Fast forward merge



1. Merge Commit (No Conflicts)



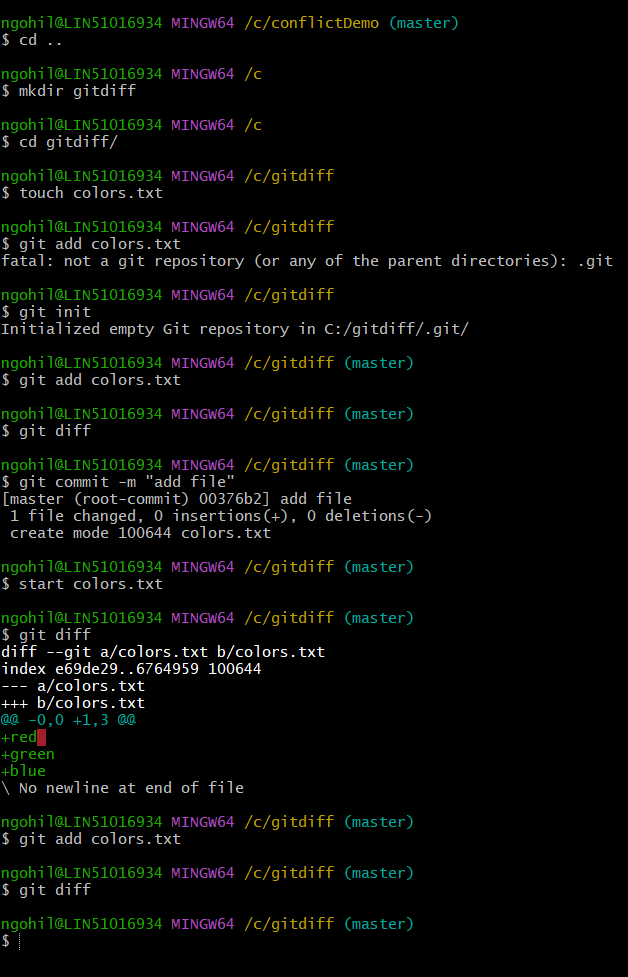
1. Conflicts



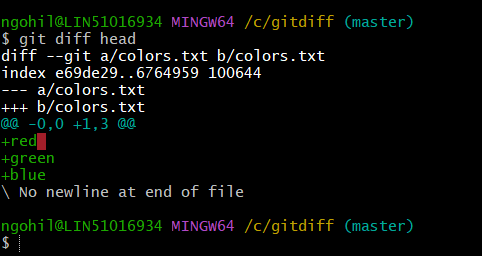
GIT Diff………

We can use this command to view changes between commits, branches, directories, files and more…

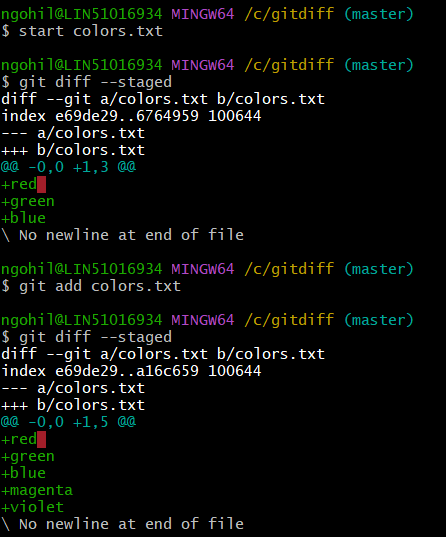
>git diff : Compares staging area and working directory.[shows only Unstaged changes]



>git diff HEAD : Lists all changes in the working tree since last commit.[shows staged and unstaged changes]



>git diff - -staged , >git diff - -cached : will list the changes between the staging area and our last commit.[only staged changes]



>git diff head <filename> : shows staged and unstaged changes from the mentioned file.

>git diff - -staged<filename> : shows staged changes from the mentioned file.

>git diff branch1..branch2 : List the changes between the tips of branch1 and branch2.

>git diff commit1..commit2 : Compare two commits, provide git diff with commit hashes.

Keep calm! Make your life much easier through Stashing 😉

Stashing uncommitted changes.

>git stash : stash will take all uncommitted changes(staged and unstaged) and stash them.

>git stash pop : remove most recently stashed changes in your stash and reapply them to your working copy.

>git stash apply : To apply whatever is stashed away, we can apply stashed changes to the multiple branches.

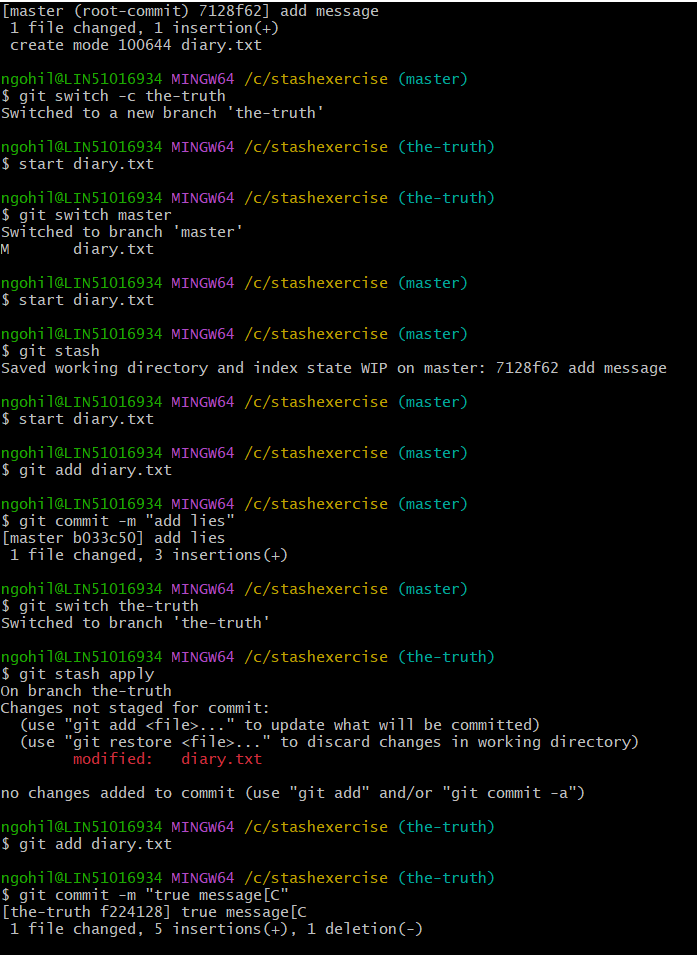
>git stash list : will give list of stashes

>git stash apply <stash reference number> : we can apply particular stash with the help of stash reference number.

>git stash drop < stash reference number > : to delete the stash.

>git stash clear : to delete all the stashes.

Stash Exercise



Checking out old commits:

>git checkout <commit-hash> :

1. To view previous commit by giving 7 characters from the command hash. It leads to the detached head as head will point to the checkout commit.

2. To remove the detached head, we can again go back to our master branch and can see all the commits.

3. You can go to any commit and make a new branch based on it and then, you can start working on that newly created branch.

4. Again you can go back to your master branch.[all the commits will be there]

>git checkout head~1 : It will checkout to one commit previous

>git checkout head<filename> : Discard changes in that file and reverting to HEAD

>git checkout - - file1 file2 : Discard changes in that file and reverting to HEAD

GIT RESTORE

>git restore <filename> : does exact same thing like git checkout head<filename> Discard changes in that file and reverting to HEAD

>git restore - -source head~1 <filename> : It will restore the contents from one commit prior to head for the mentioned filename.

>git restore - - staged<filename> : If we have accidentally added file to the staging and we don’t want to commit that file then, we can unstaged that file using this command.

GIT RESET

>git reset <commit\_hash> : Will reset the repo back to the specific commit. The commits are gone but associated changes will be intact.

>git reset - - hard <commit> : will delete the commit and associated changes.

GIT REVERT

>git revert <commit\_hash> : create new commit which will undo the changes from the mentioned commit. As a result, you will be prompted to enter the new commit message. This is useful when multiple people has copy of the same commit. So instead of losing the whole commit we can use revert.

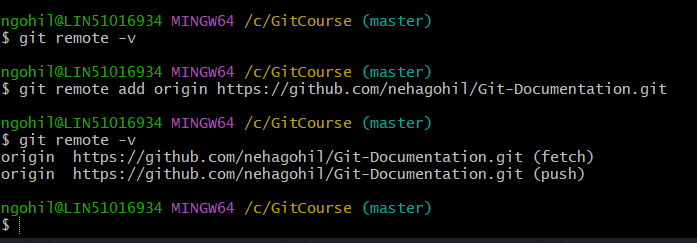
GITHUB : THE BASICS

>git clone <URL> : retrieve all files associated with this repository and will copy them to your local machine. Git initializes the new repo giving you access to the full git history of the cloned project.

GIT REMOTES

>git remote / git remote -v : to view existing remotes for your repository.

>git remote add <name> <URL> : To add a new remote we need to provide new label and remote.



>git remote rename <old> <new> : To rename the remote name

>git remote remove <name> : To delete remote if needed.

>git push <remote> <branch> : To Push the changes from the master (branch) to the remote(origin). >git push origin master

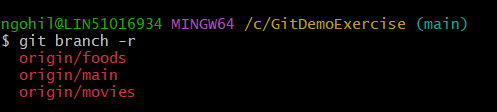
>git push <remote> <local branch>:<remote branch> : to push the changes from the local branch to the different branch on the remote. >git push origin cats(local) : Animal(remote)

>git push -u origin master : Sets the upstream of the local master branch so that it tracks the master branch on the origin repo. Once we have set the upstream, we can use >git push shorthand which will push our current branch to the upstream.

>git branch -M main : It will rename the master branch to main.

GIT FETCHING AND PULLING

>git branch -r : to view the remote branches our local repository knows about.(remote tracking branch)



>git switch <remote\_branch\_name> : To create a new local branch from the remote branch with the same name.

>git fetch <remote> : fetches branches and history from a specific remote repository. It only updates remote tracking branches. Git fetch origin would fetch all changes from the remote repository.

>git fetch <remote> <branch> : >git fetch origin master , we can get the changes from the remote to the local. And if we want to see them, then we need to checkout to that branch.

>git pull <remote> <branch> : It matters where we run this command. Because where the changes will be merged to. Would fetch the information from the origin’s master branch and merge those changes into our current branch. >git pull origin movies/master/food

>git pull : This command is a shorthand command for pulling the changes. If we switched to the food branch and then use git pull command, it will automatically pull the changes from the remote to the local branch.it pulls from origin/food automatically.(not recommended when you have uncommitted changes on the branch)