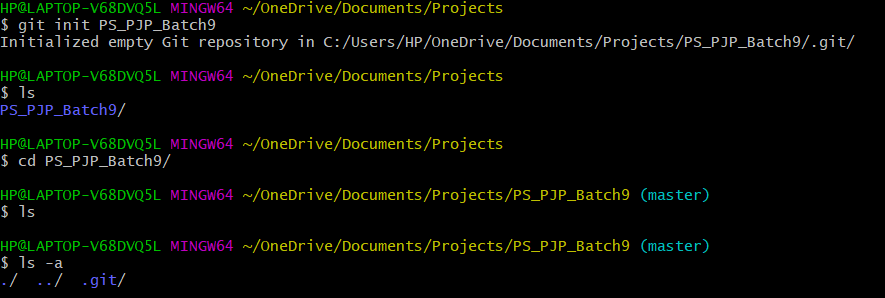
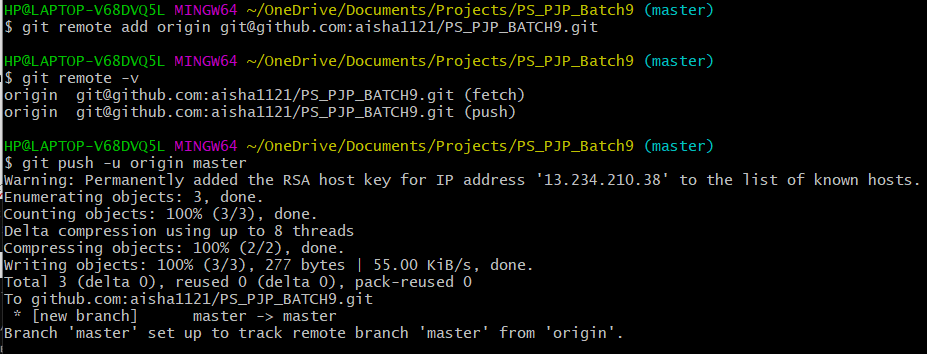
**Submitted by: Aayusha**

GitHub Assignment (Week 1)

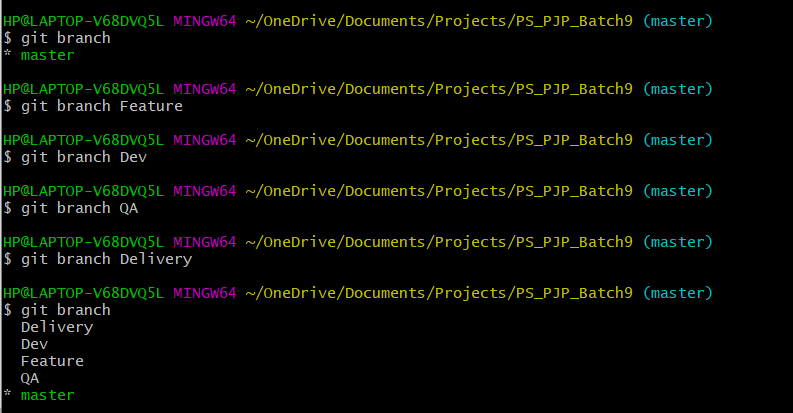
1. **Basic usage using the CLI**
2. Setup a local repository



1. Setup a remote repository



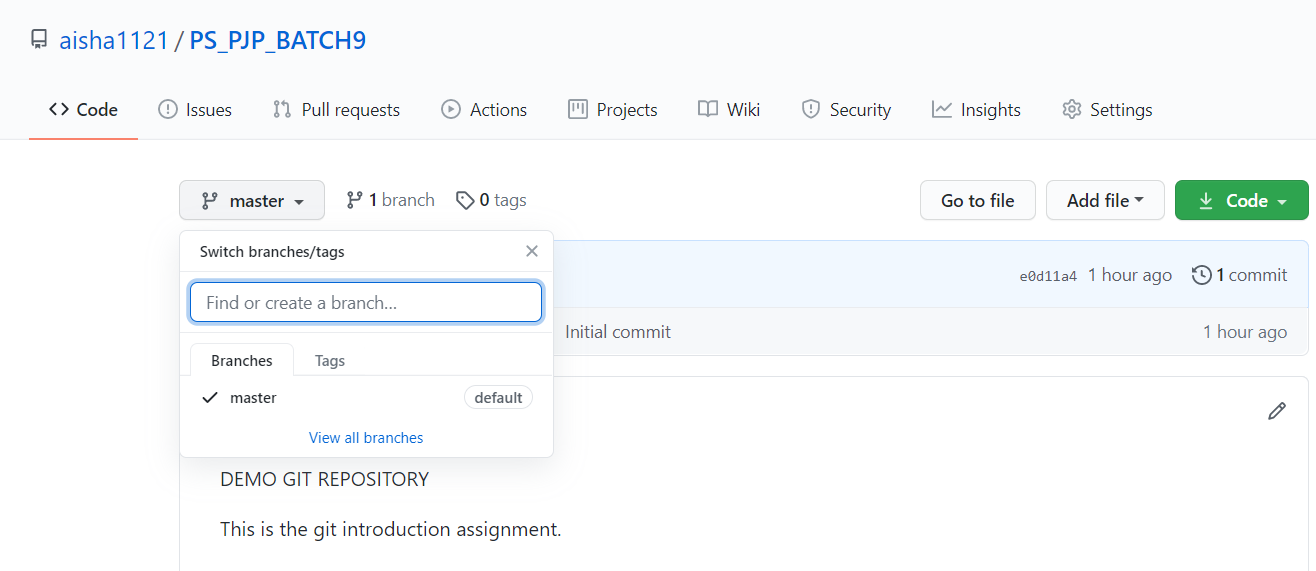
1. Create local branches (Feature, Dev, QA, Master/Prod, Delivery)



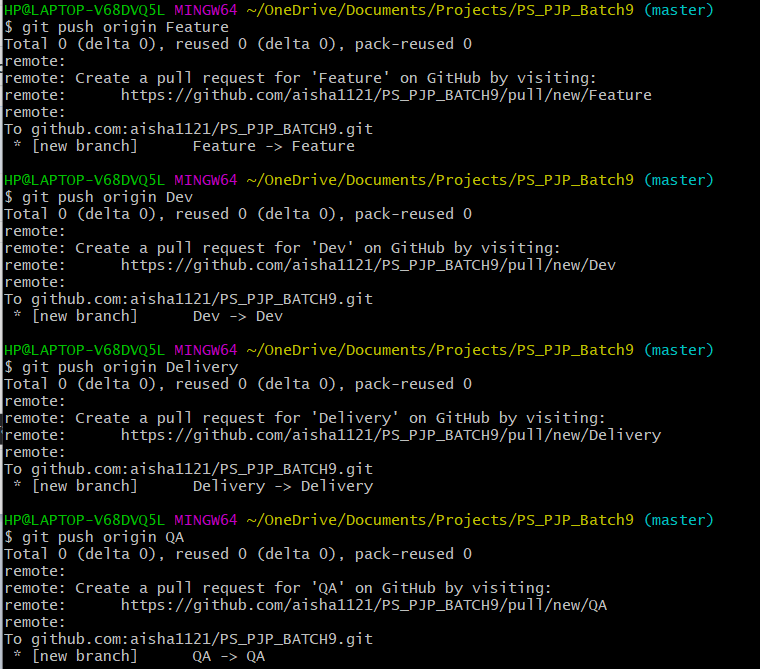
1. Create remote branches

Ans:

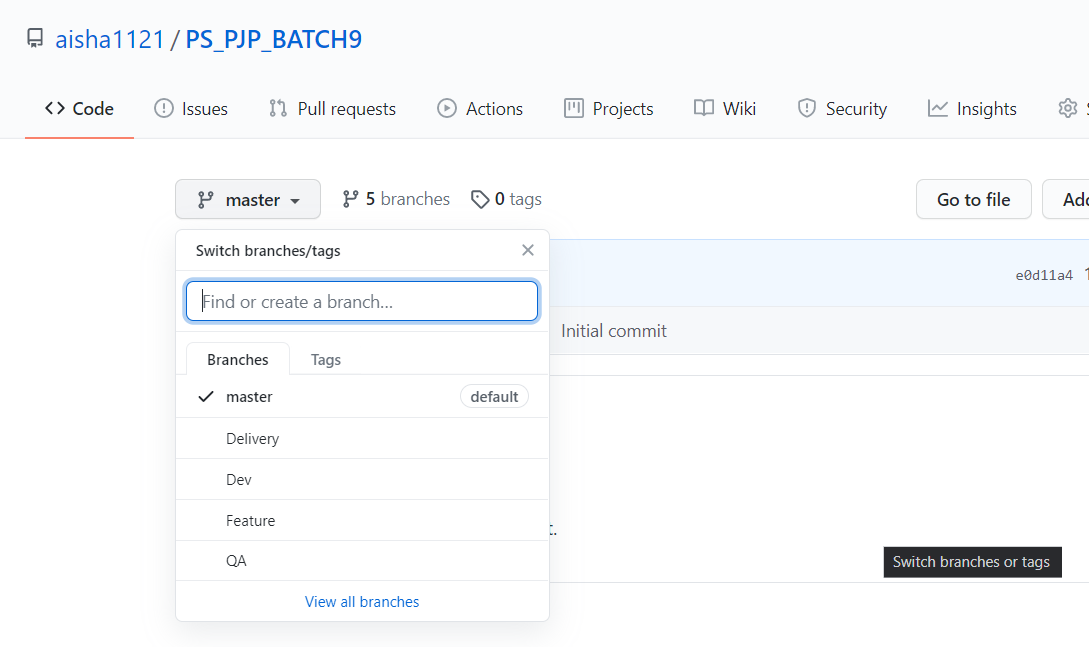
Before creating remote branches



Commands



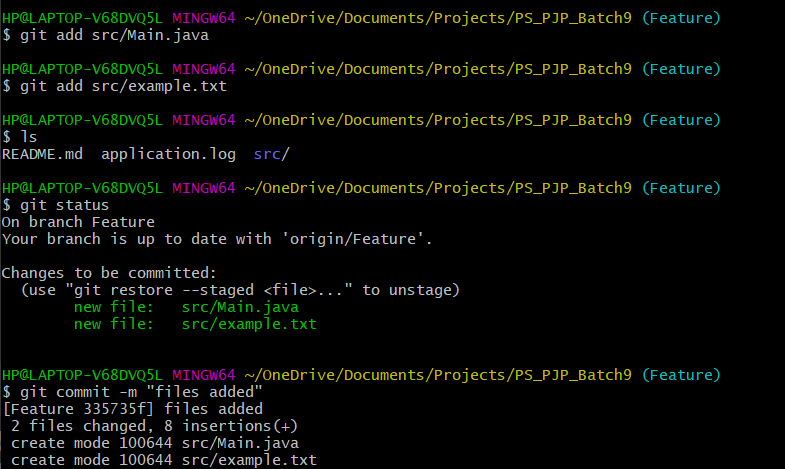
After creating remote branches



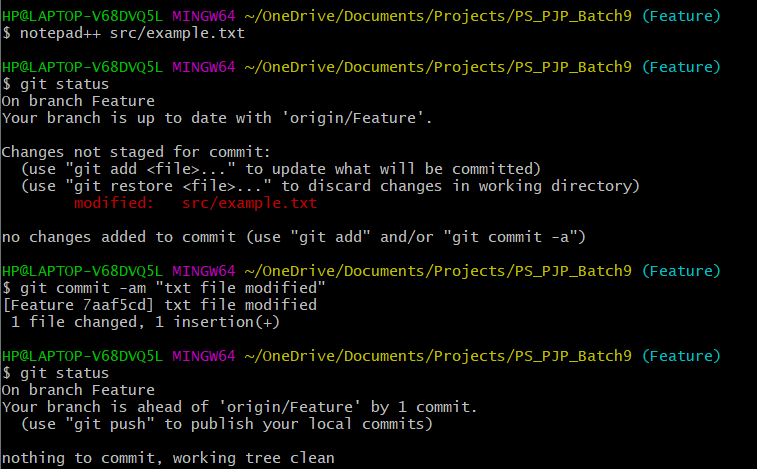
1. Add files, make changes to existing files, add folders, remove folder, remove files

Ans:

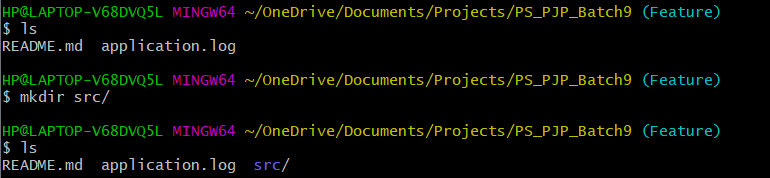
1. Adding a file



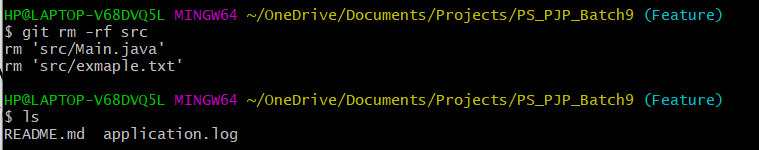
1. Modify a file



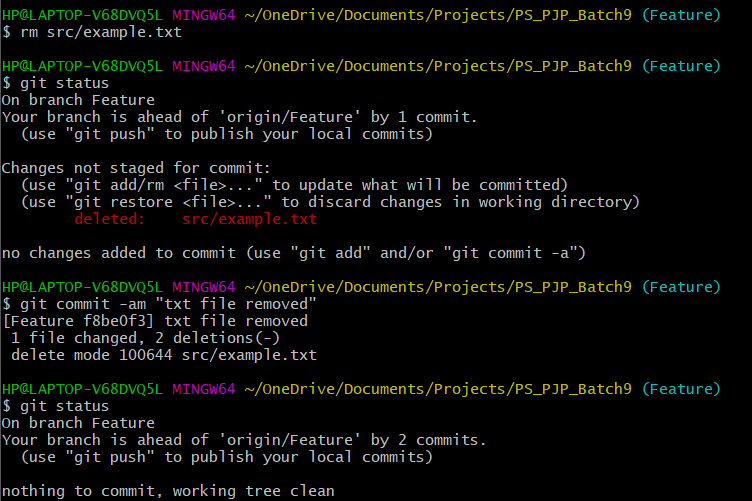
1. Add folder



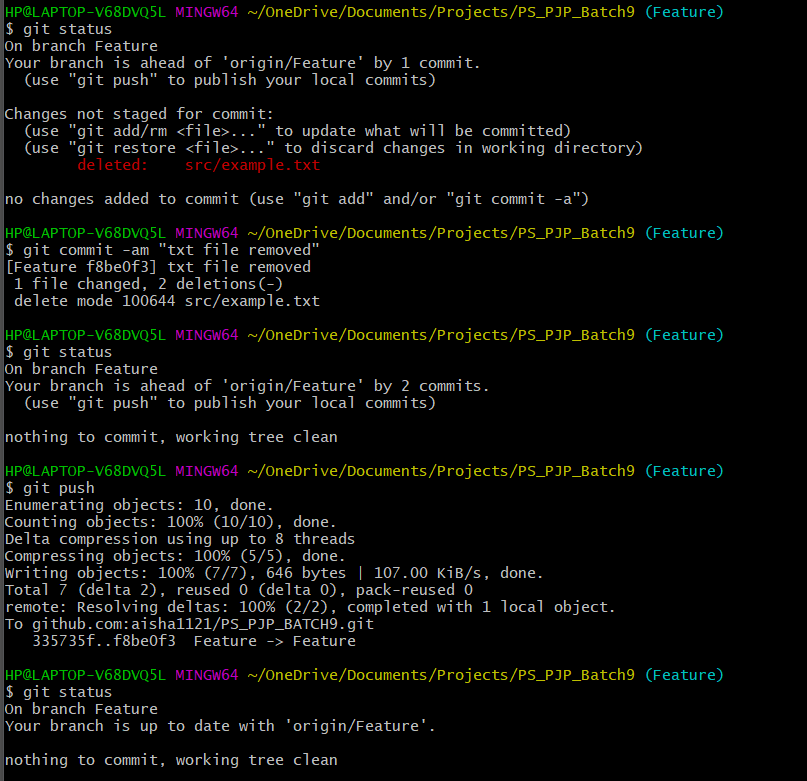
1. Remove folder



1. Remove files

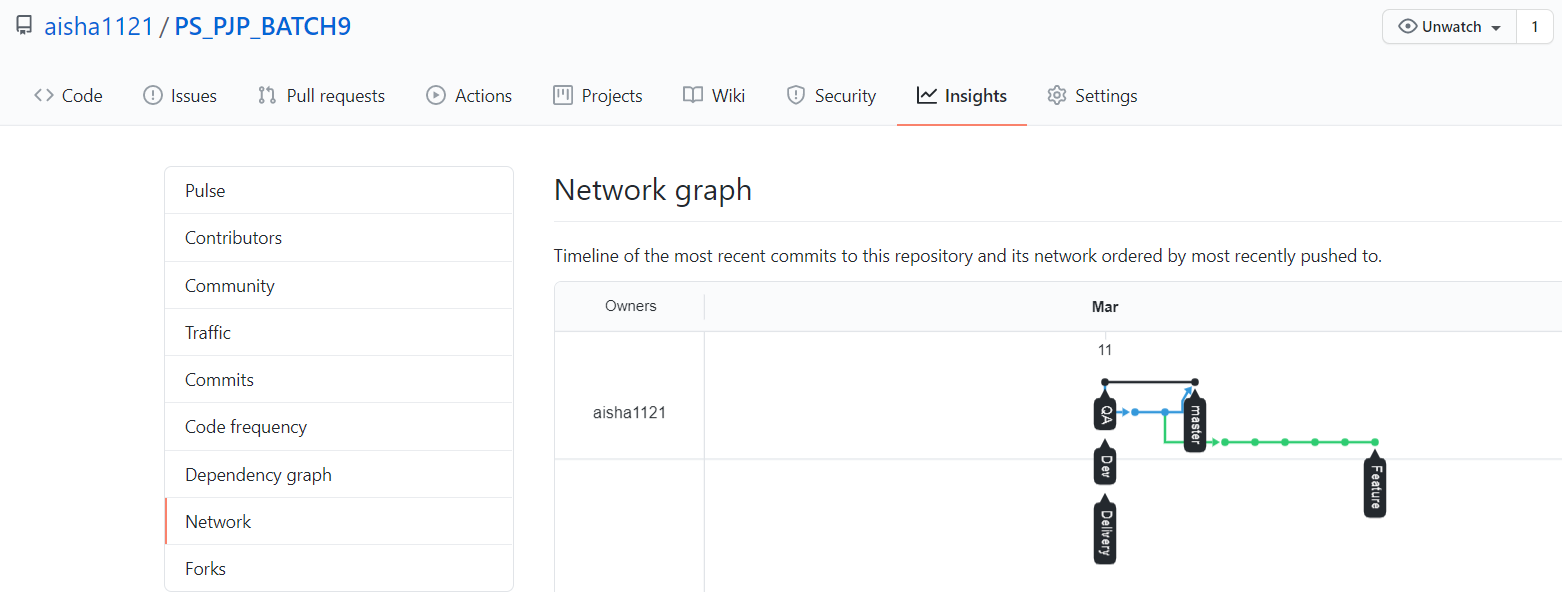


1. Check in, stage, commit, push files into Feature branch

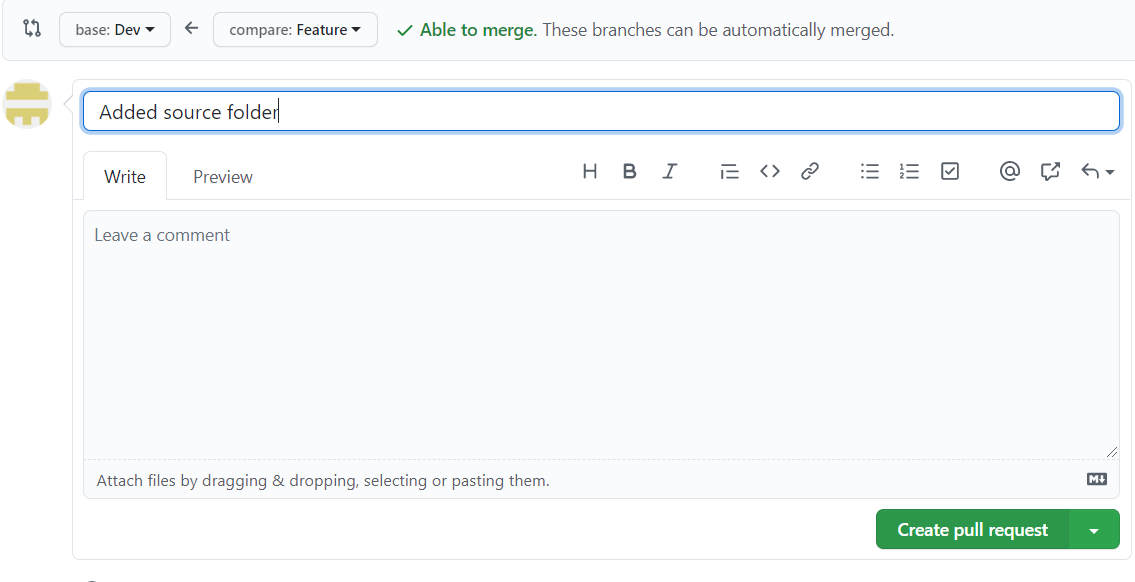


1. Promote code from Feature branch to Dev branch via pull requests

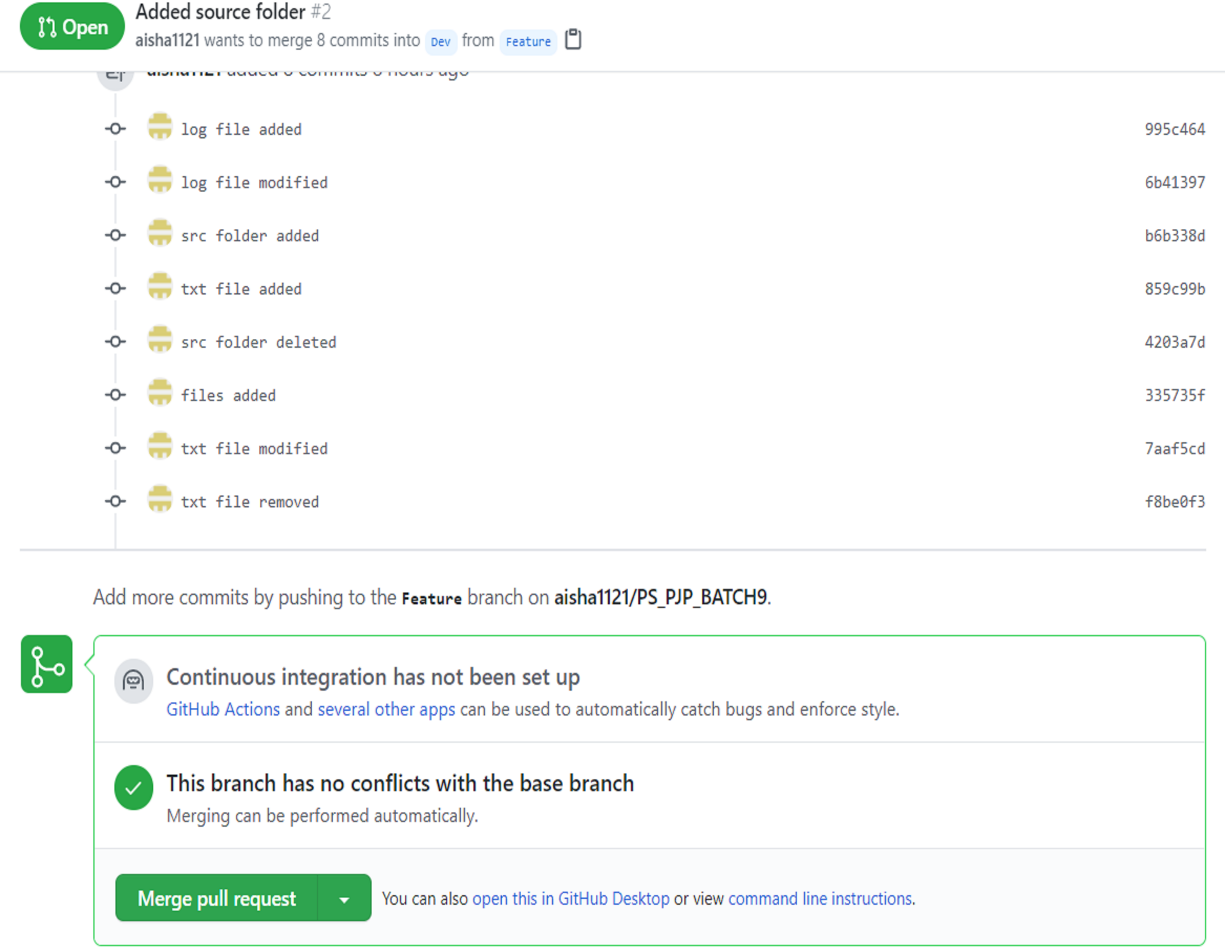
Ans: Step1: Here we can see Feature branch is ahead from Dev branch



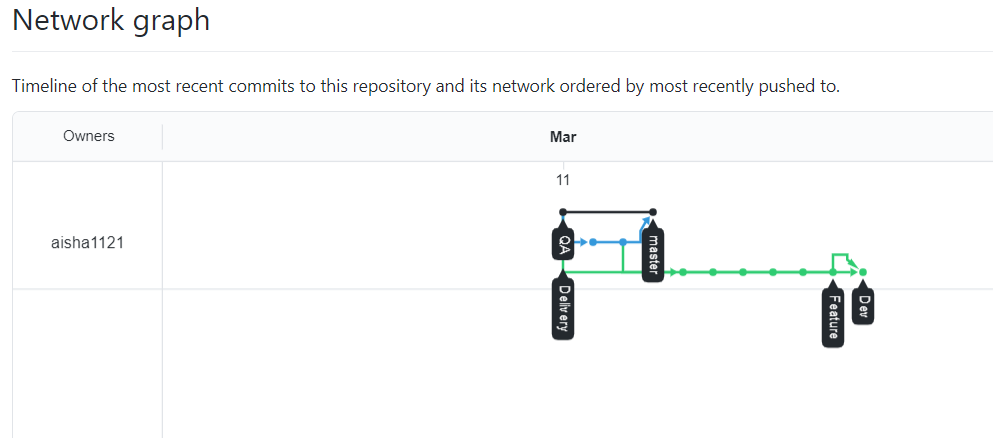
Step2: Creating pull request from Feature branch to Dev branch



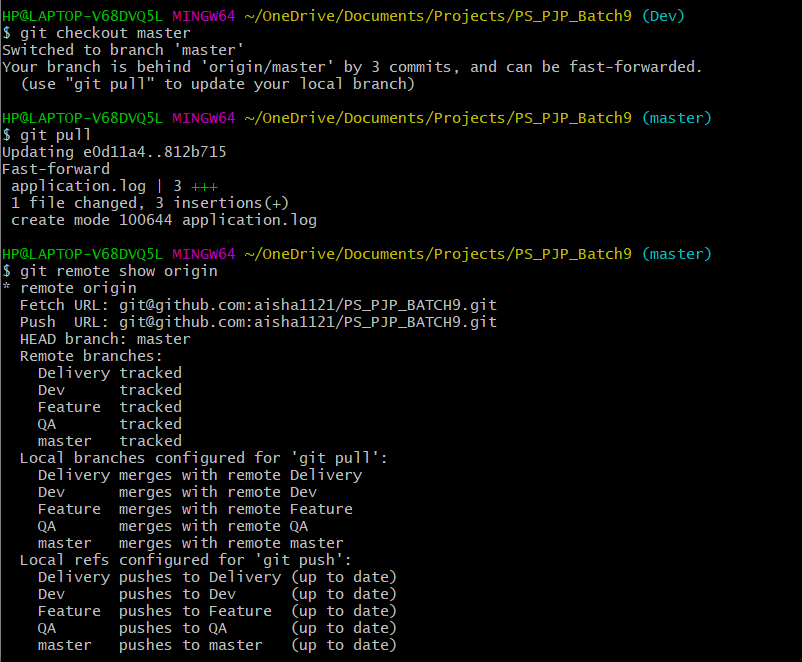
Step3:Since no merge conflicts occur, so we can merge the request.



Step4: Now after merging we can see code have been promoted to Dev branch.



1. Check out the latest code from remote branch to local branch



In a similar way updated Dev branch.

1. Explore the difference between Checkout and Pull.

Ans: Refer to Q8 (used checkout and pull command)

Checkout- **Fetches** the latest changes. We should already have this repo downloaded. It**does not merge** those new changes but makes our working directory reflect them. We can merge them at your leisure later.

Pull- **Fetches** the changes **AND merges** them **into the local branch of the same name.**

1. Get two people to make changes to the same file, check-in and handle merge conflicts.

Ans:

When two or more person modify the same file then git is not able to identify which one is correct. This leads to merge conflicts.

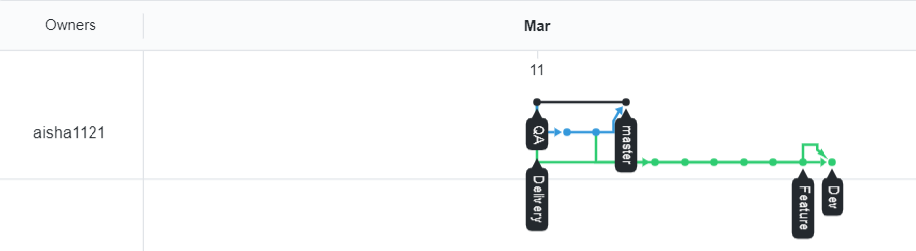
How to resolve merge conflicts:

Git gives a clue to resolve conflicts in its error message. It says Merge conflict in file1, so we know there is a problem with that file. Then it says fix conflicts and then commit the result, so if we follow directions, edit the file, then commit it, everything should work fine.

1. Ensure the code is in sync. With the latest changes across all branches from Prod, QA, Dev, Feature.

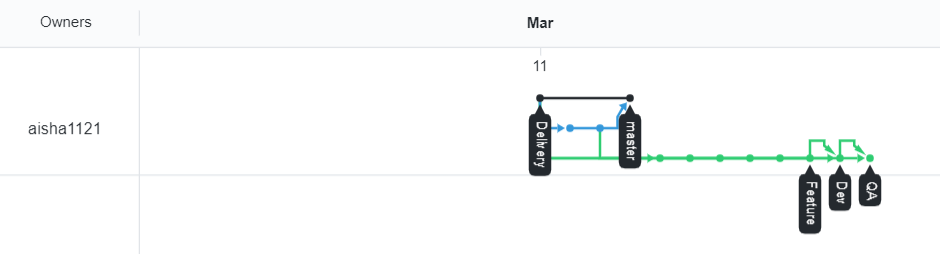
Ans:

Step1: The initial condition of different branches are as follows:



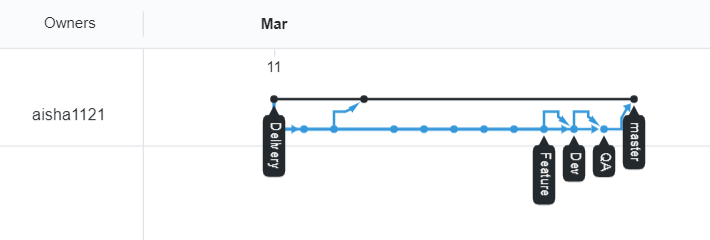
We can see QA and master branch lags Dev branch.

Step2: We will make pull request from Dev branch to QA branch and merge the code. After doing this we get.



We can see QA has been updated

Step3: Now we will make a pull request from QA branch to master and merge the code. After doing this we get.



Now code is in sync.

1. **Basic usage using the GUI Client (VS Code or GitHub Client)**
2. Setup a local repository

Ans:

Step1: make a local folder

Step2: Initialise git repository from GUI

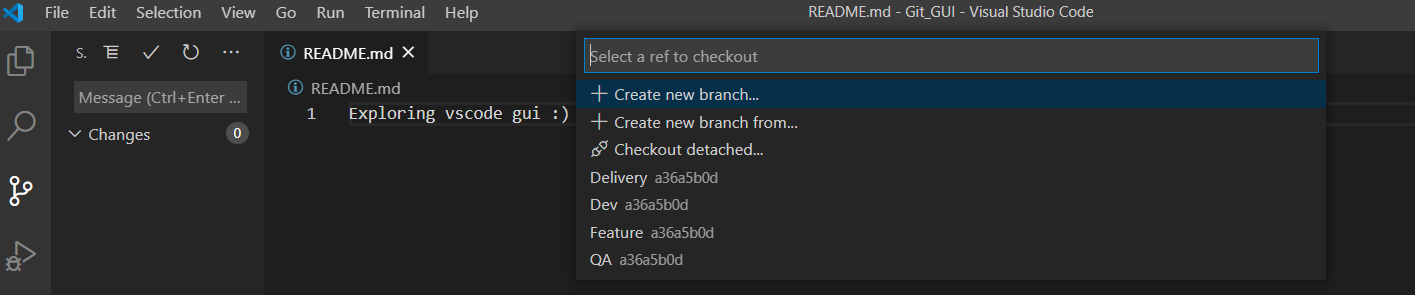
1. Setup a remote repository

Ans: same as in previous section

1. Create local branches (Feature, Dev, QA, Master/Prod, Delivery)

Ans: Follow steps as

Source control->Views and more options->Branch->Create branch



1. Create remote branches

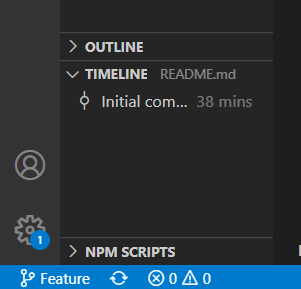
Ans: same as before

1. Add files, make changes to existing files, add folders, remove folder, remove files

Ans:

1. Add folder

Switch to Feature branch



Explorer->Add folder->Name of folder

1. Add files

Explorer->Add file->name of file

1. Modify files

Explorer->select file->make changes

1. Deletion

Right click the file or folder and delete

1. Check in, stage, commit, push files into Feature branch

Ans:

Source code->views and more actions->select commit, push to Feature branch

7,8,9,10,11 same as in part a.

1. **On the ground day to day scenarios**
2. Reset/ revert one or more files to the previous state and ignore the local changes (Soft reset and Hard reset)

Ans:

Reset- Only used when commits are not published remotely ie commits should be done local. We can reset only n numbers of recent commits (cannot reset the nth commit).

Command: git reset HEAD~3

It will reset last 3 commits.

Different types of reset:

**--**soft

Does not touch the index file or the working tree at all (but resets the head to <commit>, just like all modes do). This leaves all your changed files "Changes to be committed", as git status would put it.

**--**mixed

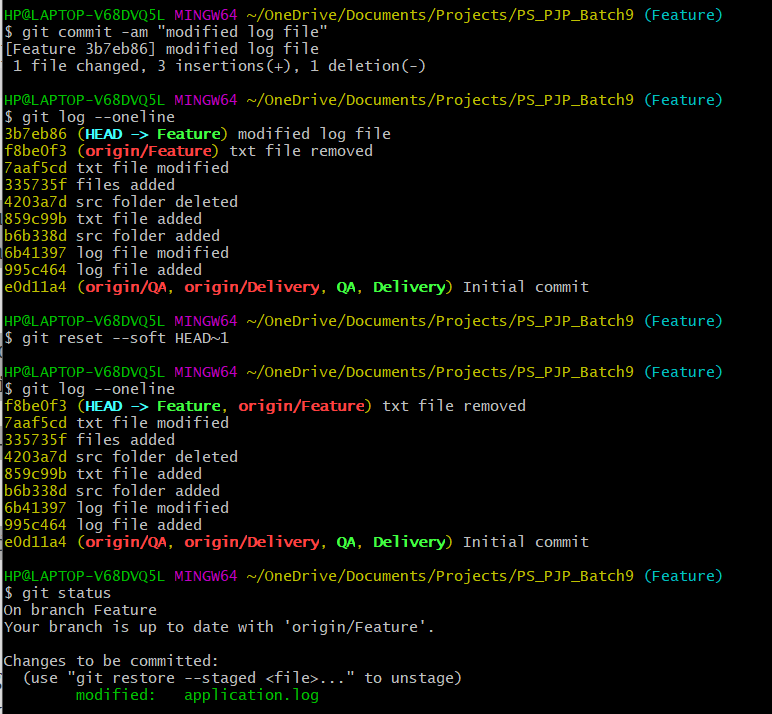
Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated. This is the default action.

If -N is specified, removed paths are marked as intent-to-add

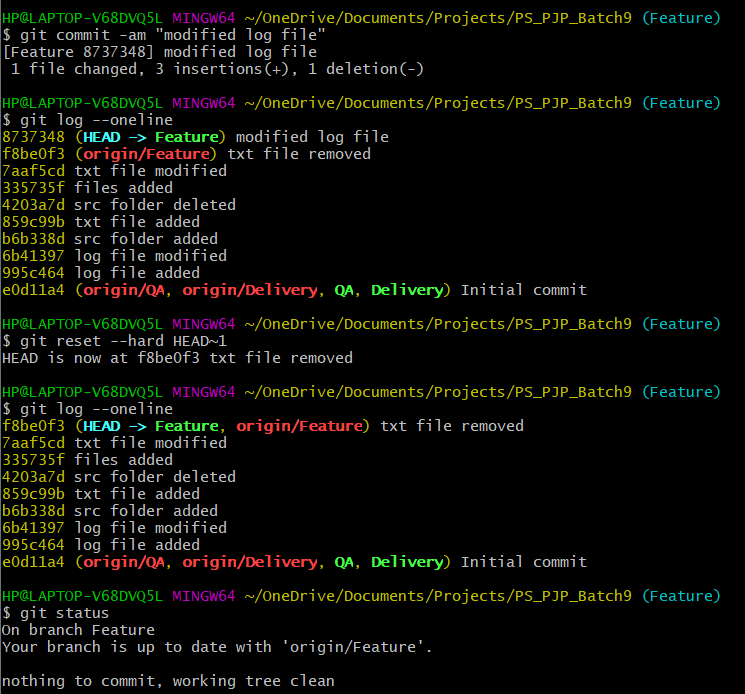
**--**hard

Resets the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded.

1. Reset soft



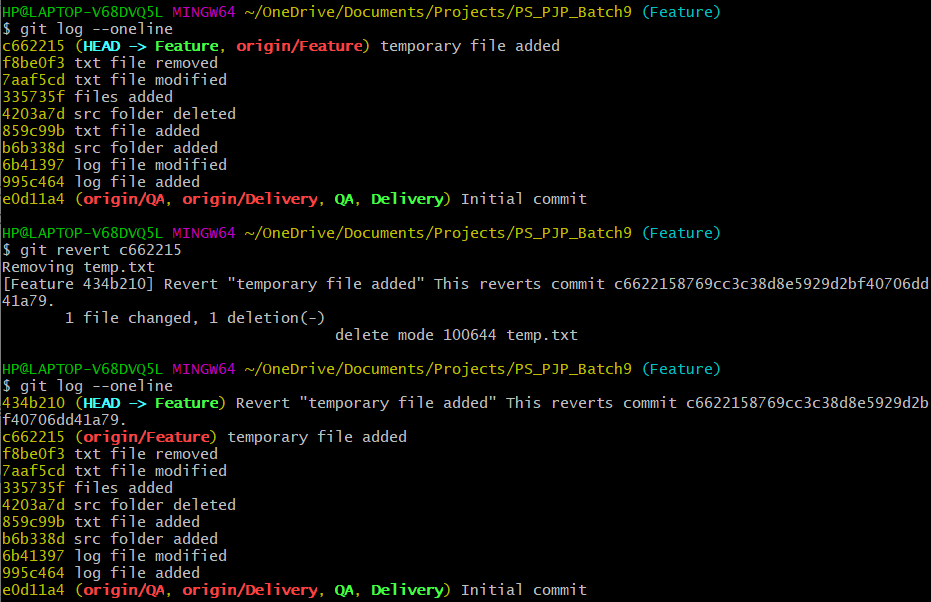
1. Reset hard



Revert-

When we have to remove commits from remote repository.

This makes a new commit and revert back the previous commit’s action.



In this way temp.txt is removed from remote as well as local repository.

1. Stash the local changes during merge conflicts

Ans: When working on a branch and suddenly have to work on another branch leaving the incomplete work. In such situations its not a good practice to commit changes and in such situations stash is used. It saves the partial work.

Commands: git stash

Git stash apply

Git stash list

1. Rebasing (reword, edit, squash, fixup, exec, drop)

Ans: Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualised in the context of a feature branching workflow.

1. Reword- used to change commit message and not the commit contents.
2. Edit- used to change both commit message and commit content
3. Squash- It takes a series of commits and squash them into a single commit
4. Fixup- It discards the message of commits to be merged and the older commits message is used to describe both the changes
5. Exec- allows to run any shell command after any commit
6. Drop- removes a commit

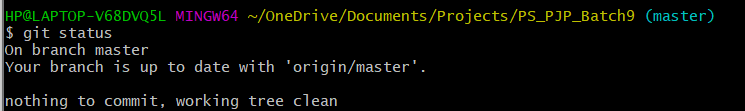
1. Git log, status, reflog

Ans:

1. Log-shows the commit logs



1. Status- shows the working tree status



1. Reflog- manages reflog information

