1/17/2022

Software Construction & Development

Assignment#05

**Group members:**

BSE173029 Sehar Iftikhar

BSE173077 Ajwa Naeem

**Submitted to:** Sir Samir Obaid

Contents

[**1.** **What is Git?** 1](#_Toc94310959)

[**2.** **Features of Git** 1](#_Toc94310960)

[**3.** **Git workflow:** 1](#_Toc94310961)

[3.1. Centralized workflow: 2](#_Toc94310962)

[Implementation: 3](#_Toc94310963)

[**3.2.** **Feature branching:** 7](#_Toc94310964)

[Implementation: 8](#_Toc94310965)

[3.3. Gitflow: 9](#_Toc94310966)

[**Figure 1: Git workflow** 2](#_Toc94310967)

[**Figure 2: Git workflow states** 3](#_Toc94310968)

[**Figure 3: Centralized workflow** 3](#_Toc94310969)

[Figure 4: git init command 4](#_Toc94310970)

[Figure 5: Add command 4](#_Toc94310971)

[Figure 6: Adding origin 5](#_Toc94310972)

[Figure 7: Clone command 5](#_Toc94310973)

[**Figure 8: Feature Branching** 8](#_Toc94310974)

[Figure 9: creating branch 9](#_Toc94310975)

[Figure 10: branch created in github 9](#_Toc94310976)

[**Figure 11: Gitflow** 11](#_Toc94310977)

# **What is Git?**

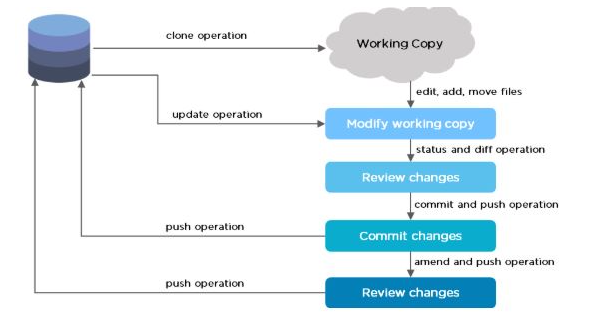
Git is a devOps (devOps is a culture that allows the development and operations teams to work together) tool used for source code managements. It is a free and open source version control system used to handle small to very large project efficiently. It is used to track changes in source code, enabling multiple developers to work together which means it allows multiple developers to work together.

Before git developers had to submit their code in central server. The management and the tracking of code was difficult before git. Now developers have an entire copy of the code on their local system. If the developer changes any code so other can track easily.

# **Features of Git**

* Track history
* Free and open source
* Supports non-linear development
* Creates backups
* Scalable
* Supports collaboration
* Branching is easier
* Distributed development

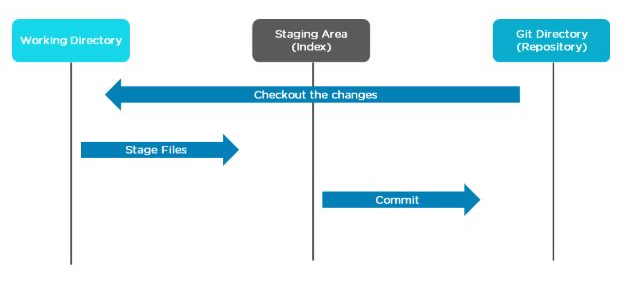
# **Git workflow:**



**Figure 1: Git workflow**

The Git workflow is divided into three states:

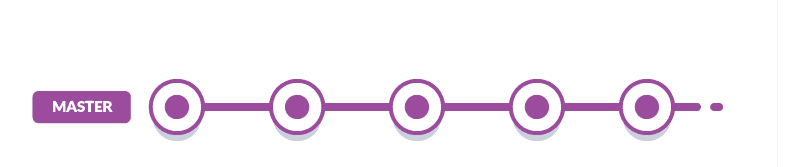
* Working directory - Modify files in your working directory
* Staging area (Index) - Stage the files and add snapshots of them to your staging area
* Git directory (Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.



**Figure 2: Git workflow states**

## Centralized workflow:

* Uses a central server to store all the files
* Every operation is performed directly on the repository
* All the versions of the file are stored on the Central VCS server
* In case the central server crashes, the entire data of the project will be lost. Hence, distributed VCS was introduced.



**Figure 3: Centralized workflow**

The above diagram shows an example of a centralized workflow. Here it is only creating one branch which is the master branch. All the developers are adding their commits to the master branch. So, the branch will become stable once all the users finish their commits.

## Implementation:

Initializing git bash in mentioned path. Checking the status of files in the folder.

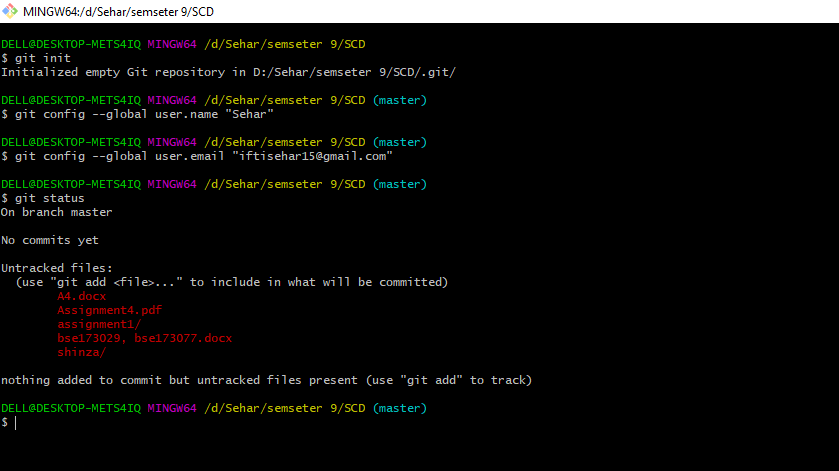


Figure : git init command

Adding different files to implement this flow.

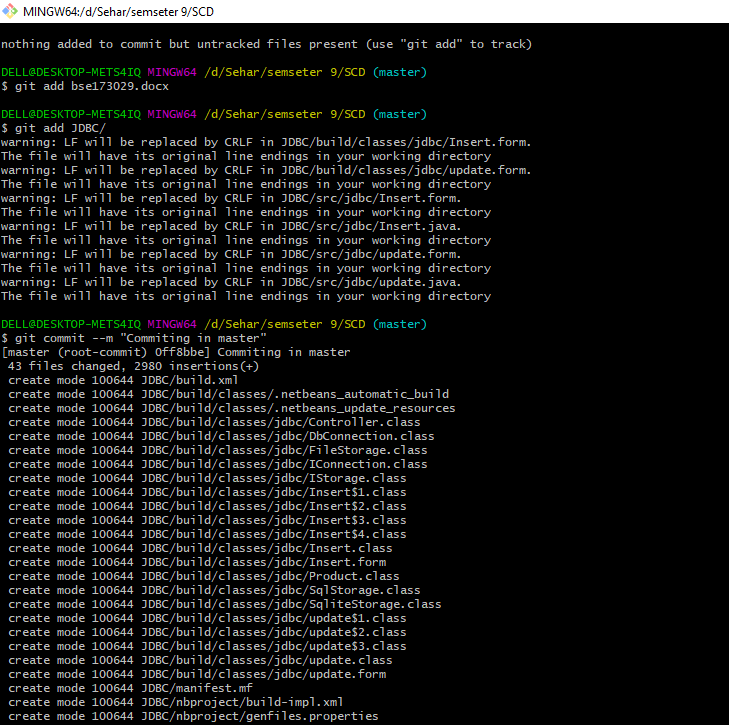


Figure : Add command

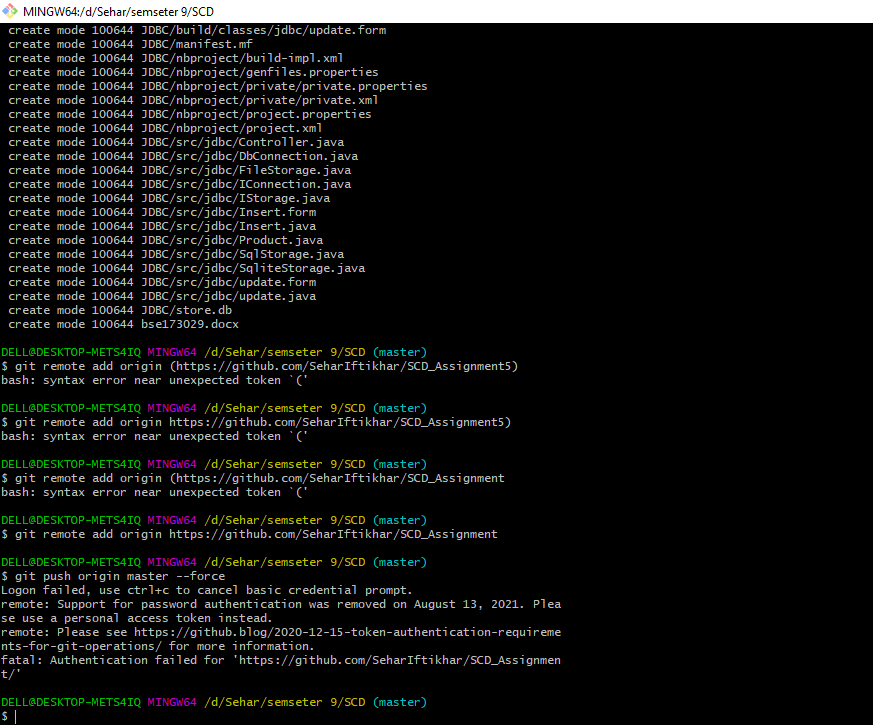
Adding remote origin 

Figure : Adding origin

Now committing without informing another collaborator. Ajwa is making new changes in files and committing it without informing other members.



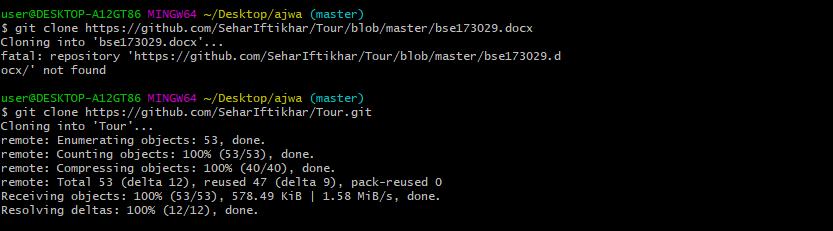
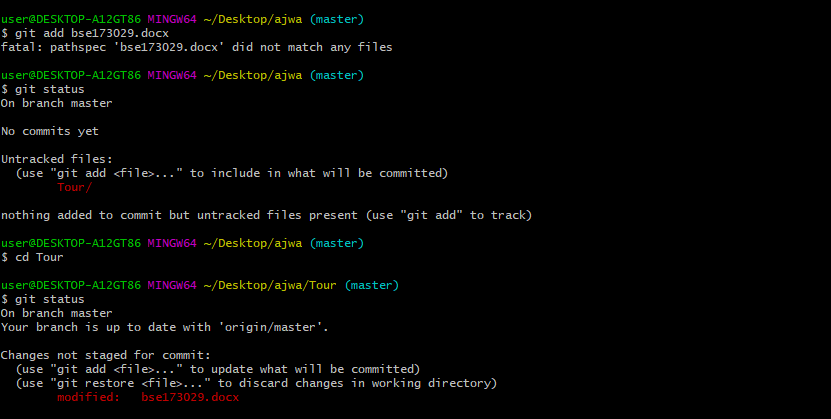
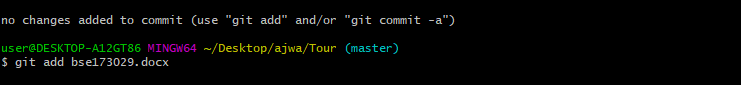
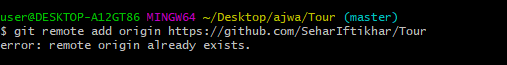
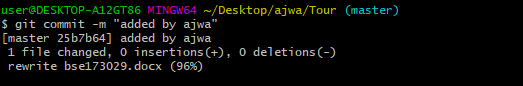
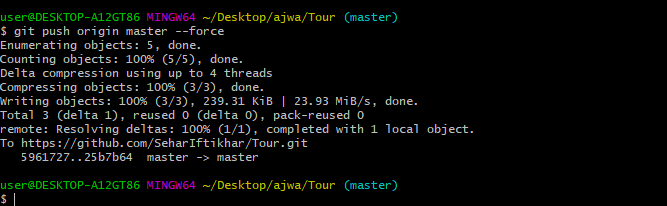
Cloning the project.

Figure : Clone command

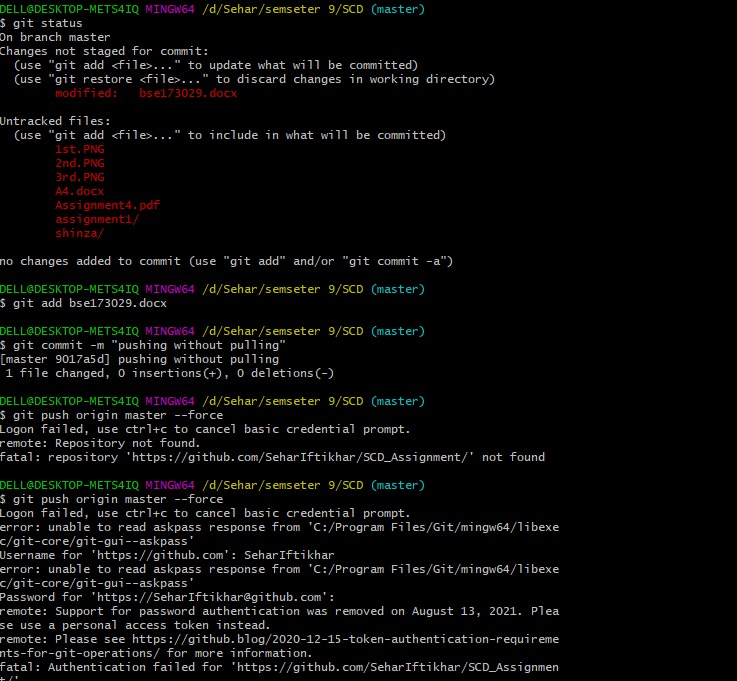




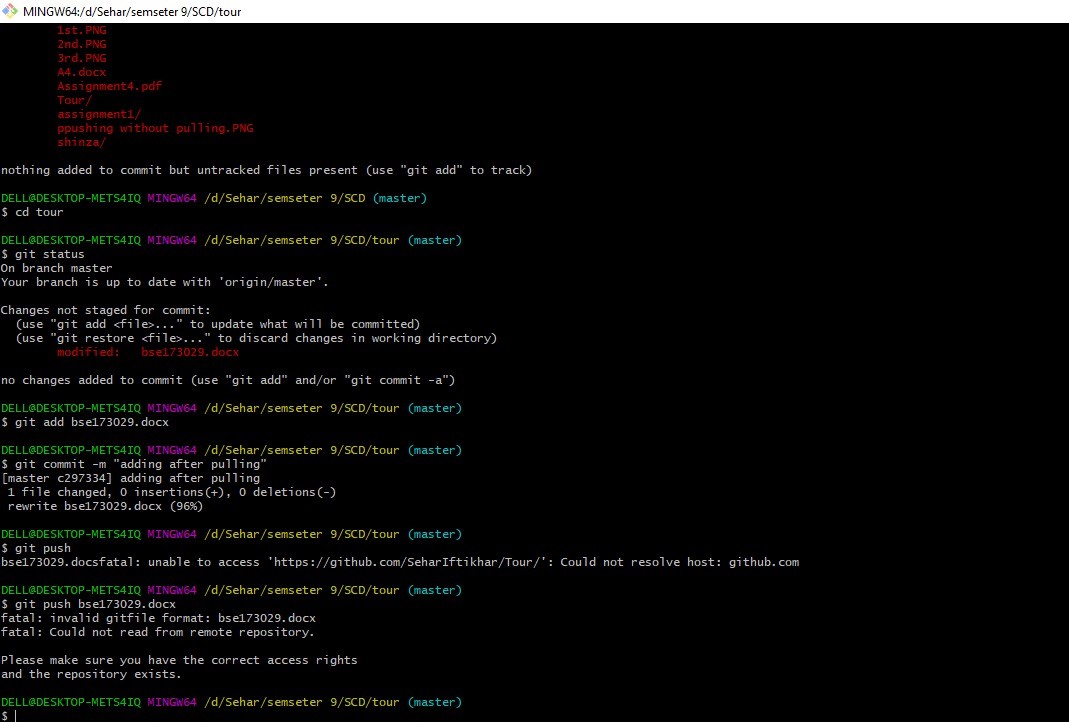


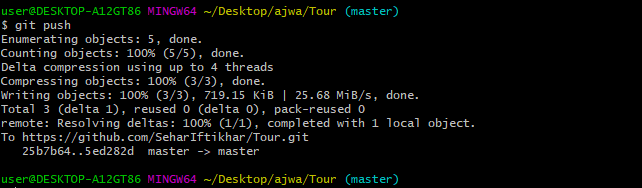
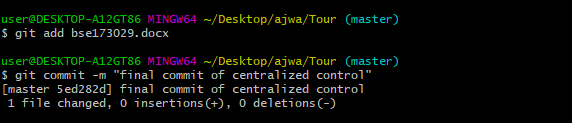


Now committing without pulling first.



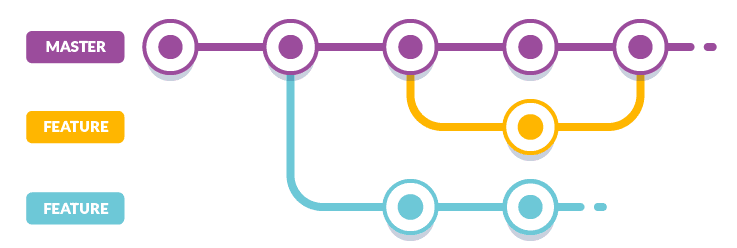
Now pushing after pulling the project.





# **Feature branching:**

The Feature Branch Workflow assumes a central repository and master represents the official project history. Instead of committing directly on the master branch, developers can create a new branch every time they start work on a new feature. Feature branches should have descriptive names.



**Figure 8: Feature Branching**

* Anything in the master branch is deployable and is stable.
* To work on something new, create a branch off from master and given a descriptive name (ie: UI-upgrade).
* Also, you can create branches from the existing feature branch.
* Commit to that branch locally and regularly push your work to the same-named branch on the server.
* When you need feedback or help, or you think the branch is ready for merging, open a pull request.
* After someone else has reviewed and signed off on the feature, you can merge it into master.
* Once it is merged and pushed to master, your feature is ready for deployment.

## Implementation:

Creating branch at first.

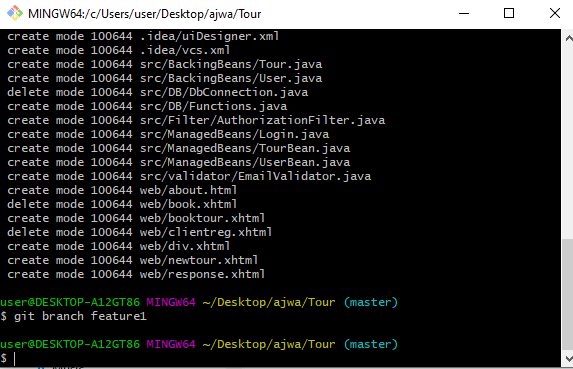


Figure : creating branch

Branch created.

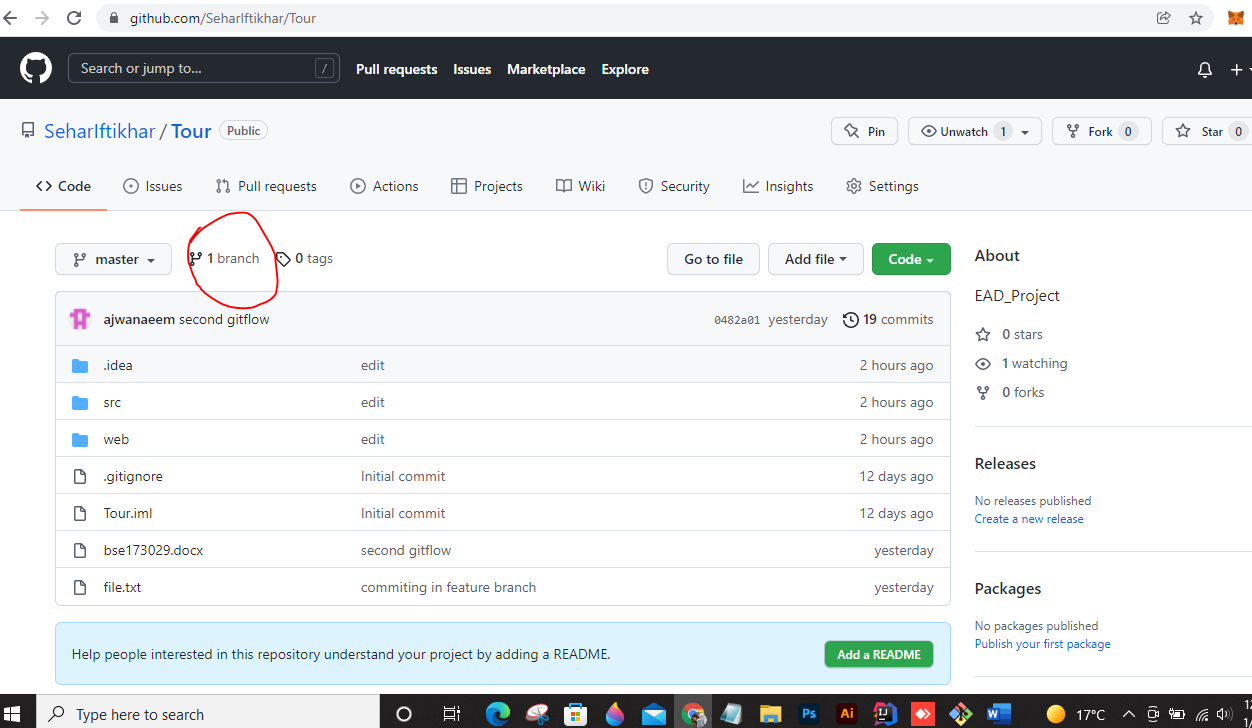
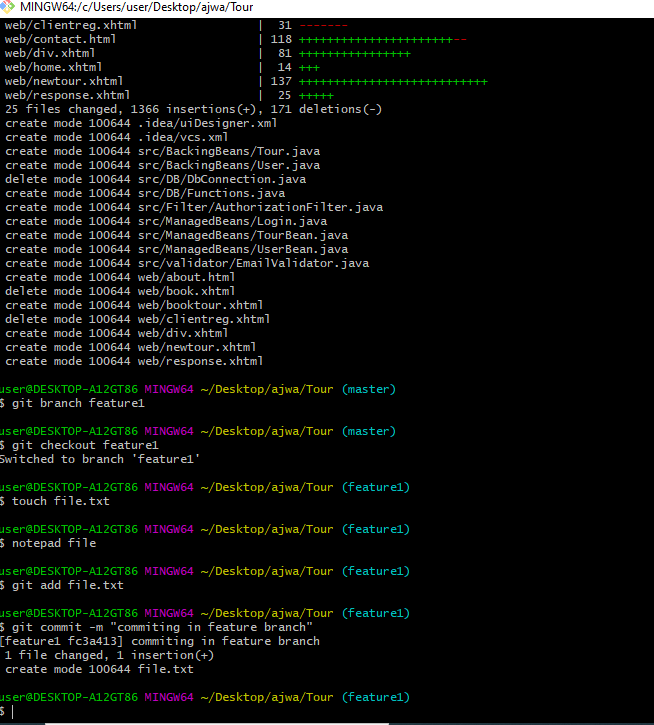


Figure : branch created in github

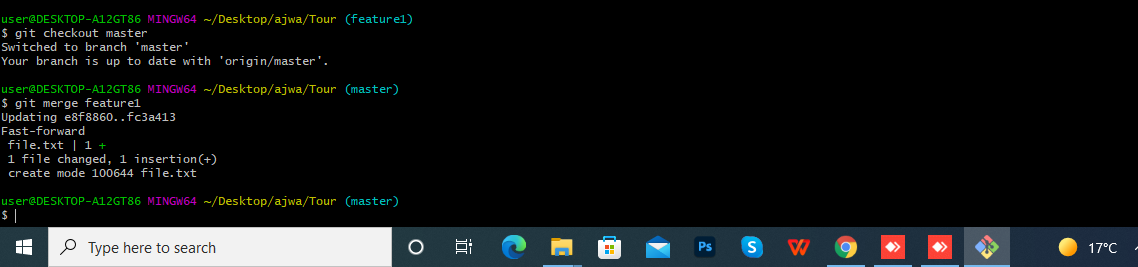
Moving toward branch.



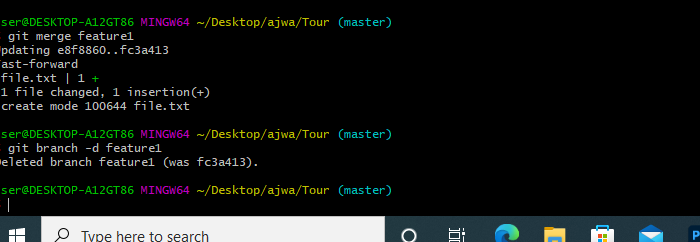
Creating file in branch 1, opened file with notepad, edit this and then add file command is used in last we commit.



Moving back to master branch to merge feature1



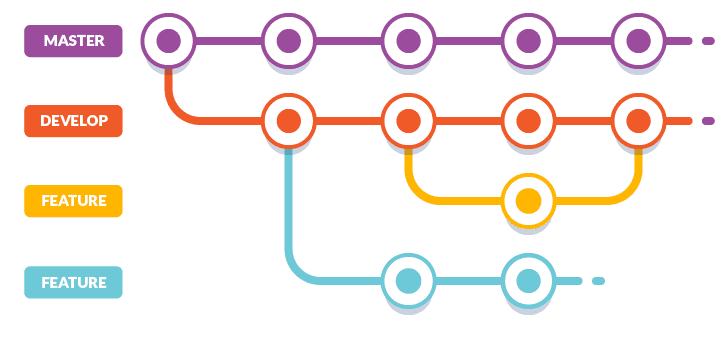
Now will delete feature 1.



After deleting we use push command to push every addons in the repository.

## Gitflow:

The Git Flow is the most known workflow on this list. It is almost similar to the feature branch workflow. But the difference is the developers are creating branches from the develop branch and it is a branch of master branch. Developers are not allowed to create branches directly from master branch. This flow eliminates buggy code from the master branch.



**Figure 11: Gitflow**

* master — this branch contains production code. All development code is merged into master in sometime.
* develop — this branch contains pre-production code. When the features are finished then they are merged into develop.