

## Experiment 04 - Advanced Git Commands

Roll No.	37
Name	Mikil Lalwani
Class	D15-B
Subject	DevOps Lab
LO Mapped	<p>LO1: To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements</p> <p>LO2: To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub</p>

**Aim:** To implement Feature Branch workflow strategies in real-time scenarios

### **Introduction:**

1. git remote add [alias] [url] - add a git URL as an alias
2. git fetch [alias] - fetch down all the branches from that Git remote
3. git merge [alias]/[branch] - merge a remote branch into your current branch to bring it up to date
4. git push [alias] [branch] - Transmit local branch commits to the remote repository branch
5. git pull - fetch and merge any commits from the tracking remote branch
6. git branch - list your branches. a \* will appear next to the currently active branch
7. git branch [branch-name] - create a new branch at the current commit
8. git checkout - switch to another branch and check it out into your working directory
9. git merge [branch] - merge the specified branch's history into the current one
10. git log - show all commits in the current branch's history

### **Workflow**

#### **1. Basic workflow**

The most popular Git development workflow and the entry stage of every project.

The idea is simple: there is one central repository. Each developer clones the repo, works locally on the code, creates a commit with changes, and pushes it to the central repository for other developers to pull and use in their work.

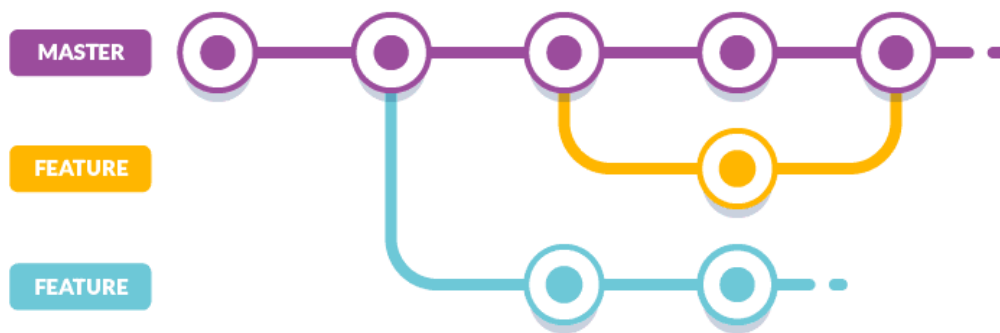


#### **2. Feature Branch workflow**

The basic workflow is great for developing a simple website. However, once two developers start working on two separate functionalities within one project, problems begin to appear.

Let's say one of the developers finished their functionality and wants to release it. However, they cannot do that because the second feature isn't ready. Making a release at this moment could result in messing everything up (to say the least).

This is where branches - the core feature of Git - come in handy. Branches are independent "tracks" of project development. For each new functionality, a new branch should be created, where the new feature is developed and tested. Once the feature is ready, the branch can be merged to the master branch so it can be released to the production server.



### 3. Forking workflow

The owner of the repository decides who can push to the repo. This is done by forks: any time a developer wants to change something in an open-source project, they don't work directly on the project's repository. Instead, they fork it, effectively creating a copy of the entire repo. Then, the developer is free to work on the new feature in whatever fashion they please. It's worth mentioning that forking opens up possibilities of creating custom versions of certain components tuned for specific applications, too. A developer or a company can fork a repository and take the code in an entirely new direction which wouldn't be allowed by the owner(s).

In most of cases, however, when the work is done, a pull request is created, which compares the changes the developer introduced to their fork to the state of the forked repository. From there, the community and the project's owners can review, discuss, and test the changes. The final call remains in the hands of the project's skipper and their deputies.

### Feature Branch Workflow

If dev1 wants to add a new feature he creates a new branch dev1-feature.

He starts with the development and makes some progress. Later he decides to take a break and commits his features to his local repository.

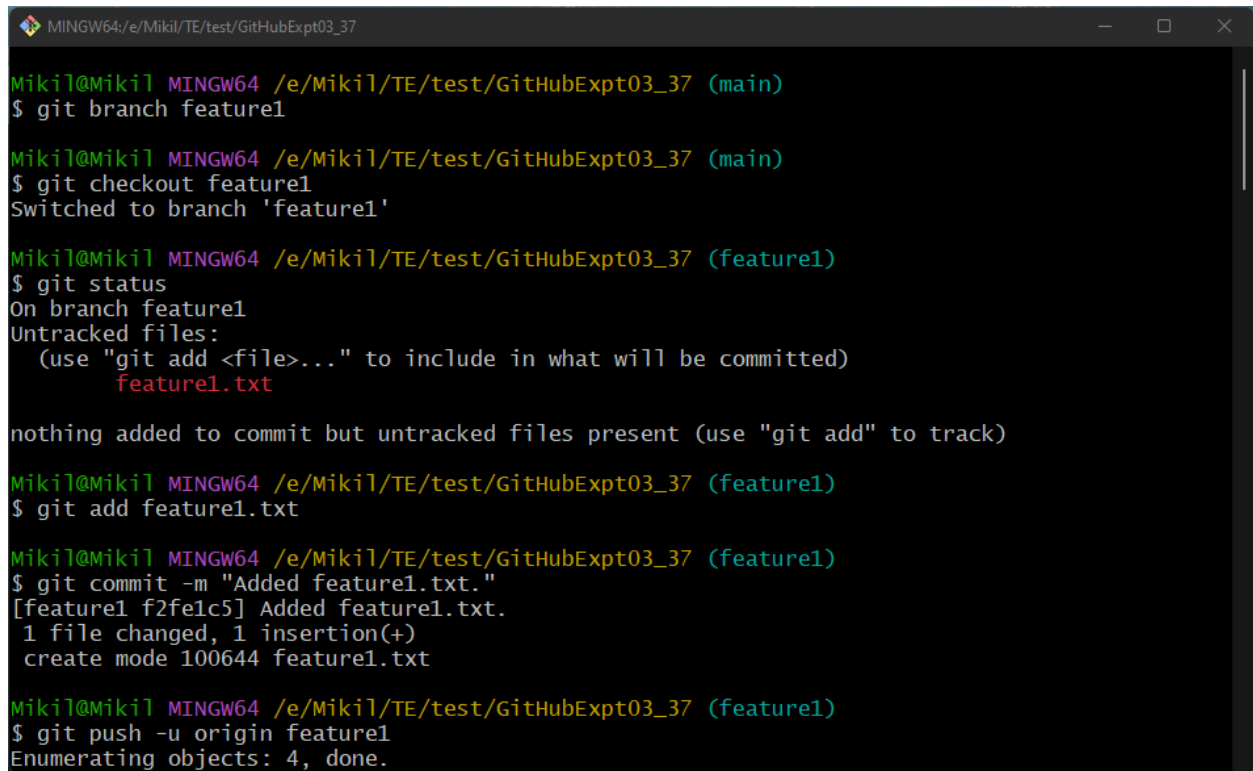
Then he starts again and completes the feature. He then commits and pushes the feature to remote dev1-feature branch.

He then creates a pull request. Senior developer dev2 checks the feature and suggests some changes few times.

Later the code is merged with main repository.

If the code was committed directly to the main repository then making the changes would be difficult.

So have a dedicated branch for development of feature makes sure that the feature is perfect before finalization.



```
MINGW64:/e/Miki/TE/test/GitHubExpt03_37
Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (main)
$ git branch feature1

Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (main)
$ git checkout feature1
Switched to branch 'feature1'

Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (feature1)
$ git status
On branch feature1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    feature1.txt

nothing added to commit but untracked files present (use "git add" to track)

Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (feature1)
$ git add feature1.txt

Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (feature1)
$ git commit -m "Added feature1.txt."
[feature1 f2fe1c5] Added feature1.txt.
1 file changed, 1 insertion(+)
create mode 100644 feature1.txt

Miki1@Miki1 MINGW64 /e/Miki/TE/test/GitHubExpt03_37 (feature1)
$ git push -u origin feature1
Enumerating objects: 4, done.
```

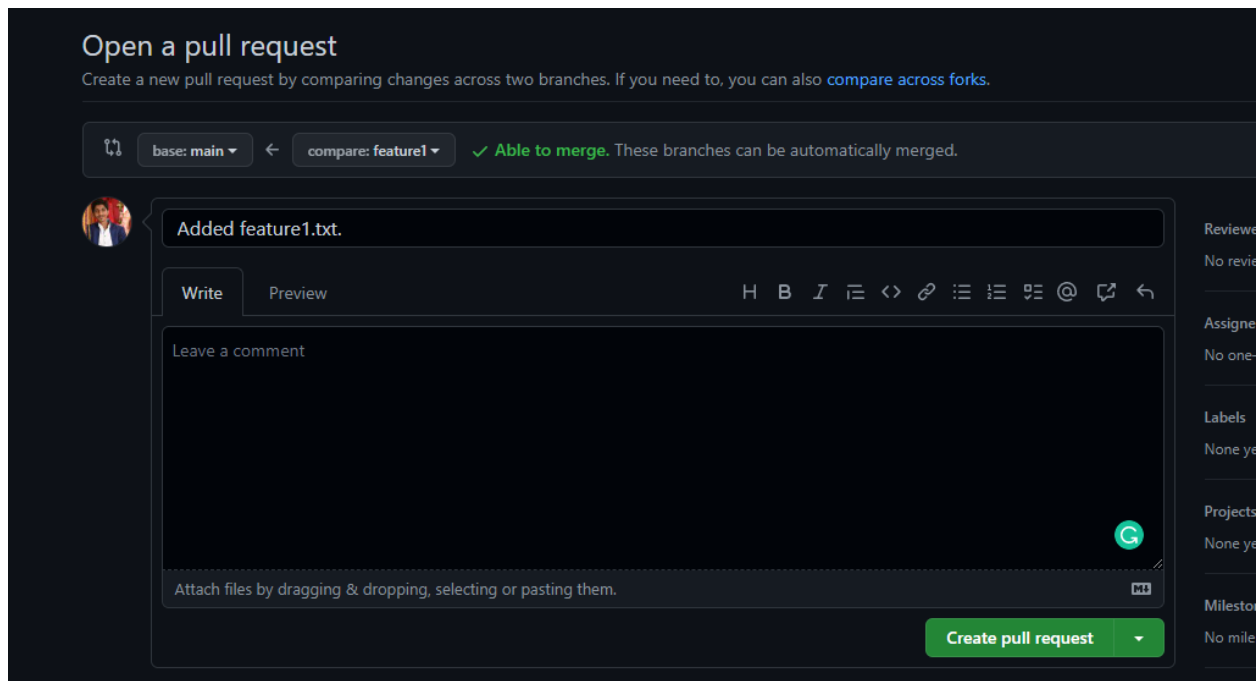
```
MINGW64: e/Mikil/TE/test/GitHubExpt03_37
create mode 100644 feature1.txt

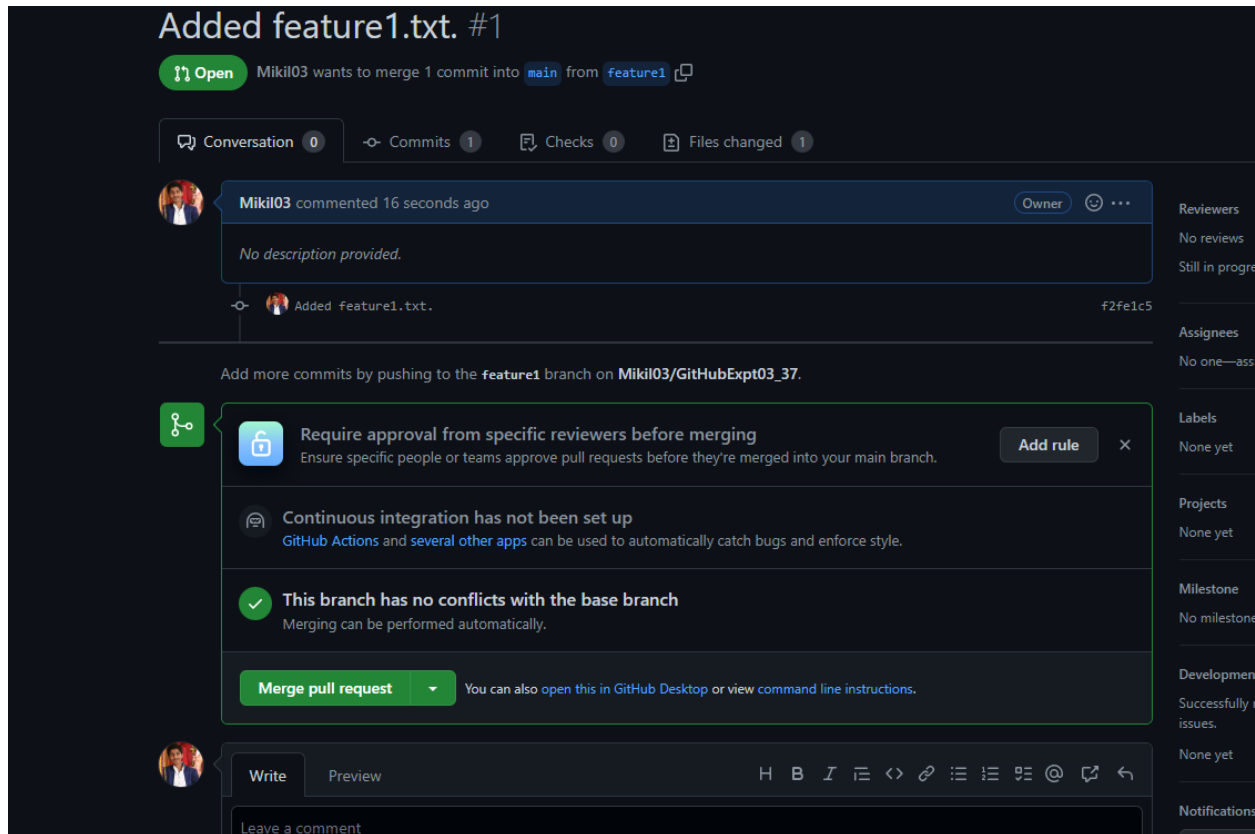
Mikil@Mikil MINGW64 /e/Mikil/TE/test/GitHubExpt03_37 (feature1)
$ git push -u origin feature1
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes | 303.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature1' on GitHub by visiting:
remote:   https://github.com/Mikil03/GitHubExpt03_37/pull/new/feature1
remote:
To https://github.com/Mikil03/GitHubExpt03_37.git
 * [new branch]      feature1 -> feature1
branch 'feature1' set up to track 'origin/feature1'.

Mikil@Mikil MINGW64 /e/Mikil/TE/test/GitHubExpt03_37 (feature1)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Mikil@Mikil MINGW64 /e/Mikil/TE/test/GitHubExpt03_37 (main)
$ git checkout feature1
Switched to branch 'feature1'
Your branch is up to date with 'origin/feature1'.

Mikil@Mikil MINGW64 /e/Mikil/TE/test/GitHubExpt03_37 (feature1)
```





## Conclusion

We have successfully completed the experiment.