



**#Index Task 1.2#**

<b>S. No</b>	<b>Program Title</b>	<b>Teacher Signature</b>
<b>1.</b>	<b>Clone, Fork, and Create Pull Requests</b>	
<b>2.</b>	<b>Using .gitignore and README Files</b>	
<b>3.</b>	<b>Scenario-Based Merge Conflict Resolution</b>	



## Practical No. : 5

**Aim:** Clone, Fork, and Create Pull Requests.

**Theory:**

### 1. Cloning a Repository:

Cloning is the process of creating an exact copy of a remote repository on a local machine. It allows developers to work on a project independently while keeping a link to the original repository for future updates. When a repository is cloned, all its files, branches, and commit history are copied to the local system. This is commonly used when a developer wants to contribute to a project or manage their own repository offline.

### 2. Forking a Repository

Forking is a GitHub feature that allows a user to create a personal copy of another user's repository in their own GitHub account. This is useful for contributing to open-source projects because it enables developers to make changes without affecting the original repository. The forked repository acts as an independent version, where modifications can be made and later proposed for integration into the original project.

### 3. Creating a Pull Request (PR)

A Pull Request (PR) is a mechanism in GitHub that allows developers to propose changes from their forked repository to the original repository. It is commonly used in collaborative development, where multiple contributors suggest improvements or fixes to a project. A pull request must be reviewed and approved before the changes are merged into the main project.

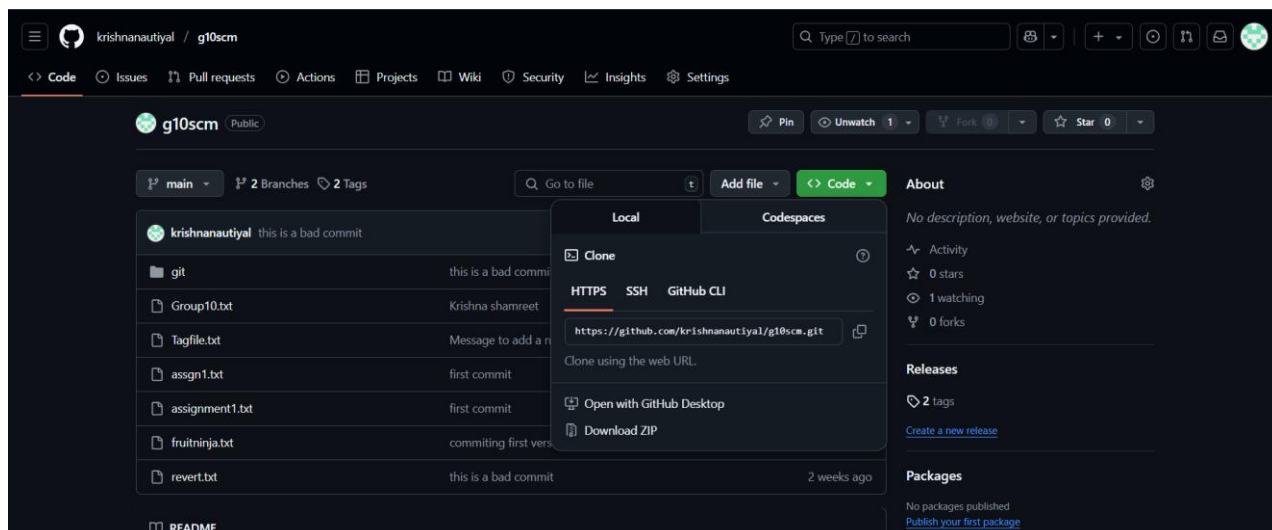
**Procedure:**

### 1. Cloning a Repository:

Cloning a repository allows you to create a local copy of a remote GitHub repository. Steps to

Clone a Repository:

1. Copy the link of the Repository you want to clone.





2. Open Git Bash or a terminal.
3. Navigate to the directory where you want to store the cloned repository.
4. Run the following command:

```
KRISHNA@victus MINGW64 /d/10march
$ git clone https://github.com/krishnanautiya1/g10scm.git
Cloning into 'g10scm'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 36 (delta 7), reused 35 (delta 6), pack-reused 0 (from 0)
Receiving objects: 100% (36/36), done.
Resolving deltas: 100% (7/7), done.
```

The repository is now cloned and ready for local development.

```
KRISHNA@victus MINGW64 /d/10march
$ ls
g10scm/  revert/
```

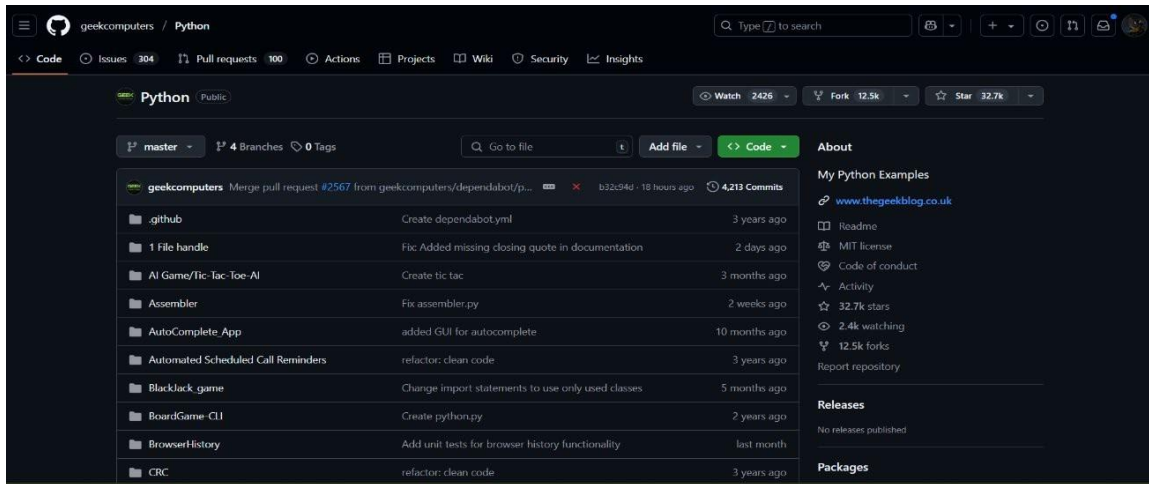


## Forking a Repository:

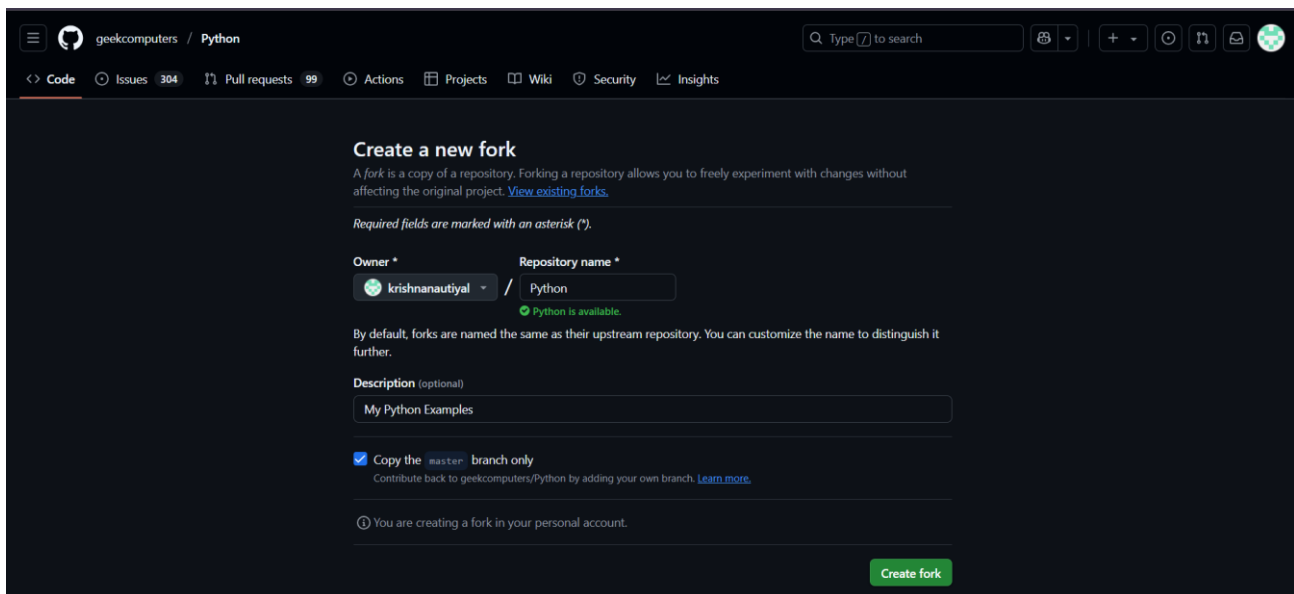
Forking creates a copy of someone else's GitHub repository under your own GitHub account, allowing you to modify it independently.

### Steps to Fork a Repository:

1. Open **GitHub** and go to the repository you want to fork.



2. Click on the **Fork** button at the top-right corner.
3. You can change the name for the forked repository, and then click on the **Create Fork** Option.



4. The forked repository will now appear under your GitHub account.



## Creating a Pull Request (PR):

A **Pull Request (PR)** is used to suggest changes from a forked repository to the original repository. Steps to

### Create a Pull Request:

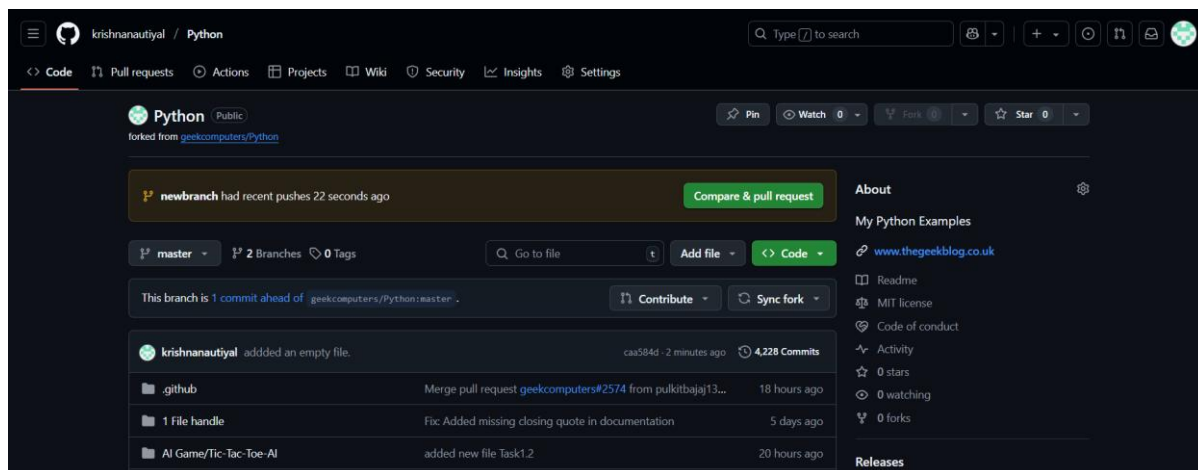
1. Open the cloned repository folder in a code editor.
2. Make necessary changes and save them.
3. Open **Git Bash** or terminal and navigate to the repository.
4. Stage the changes:
5. Commit the changes with a message
6. Push the changes to your forked repository:

```
KRISHNA@victus MINGW64 /d/Python (master)
$ git add .

KRISHNA@victus MINGW64 /d/Python (master)
$ git commit -m "added an empty file."
[master caa584d] added an empty file.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 newtxtfile.txt

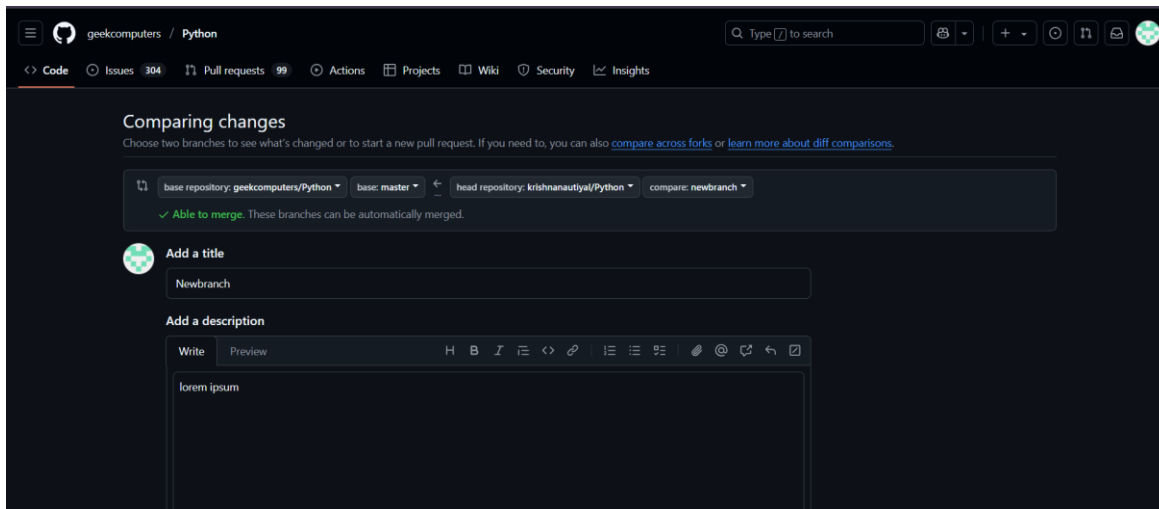
KRISHNA@victus MINGW64 /d/Python (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 261 bytes | 261.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/krishnanauiyal/Python.git
6469a9c..caa584d master -> master
```

7. Go to **GitHub**, open your forked repository, and click "**Compare & pull request**".





8. Review the changes and click "Create Pull Request".



9. The repository owner will review your request and decide whether to merge it.

## Practical No. : 6

**Aim:** Using gitignore and README Files.

### **Theory:**

#### **1. .gitignore File:**

A .gitignore file is used in Git to specify files and directories that should be ignored by Git and not tracked in the repository. This is useful for excluding sensitive data, compiled files, or temporary files from version control.

#### **Common Use Cases of .gitignore:**

- Ignoring log files (.log), temporary files, and cache files.
- Excluding compiled files like .class (Java), .exe (Windows executables).
- Preventing tracking of personal configuration files such as config.json.

#### **2. README File:**

A README.md file is the main documentation file of a repository, usually written in Markdown format. It provides essential details about the project, including its purpose, installation steps, usage, and contributions.

#### **Common Sections in a README:**

- **Project Title & Description** – What the project does.
- **Installation Steps** – How to install dependencies.
- **Usage** – Instructions on how to use the project.
- **Contributions** – Guidelines for contributing.

### **Procedure:**

#### **1. Creating and Using a .gitignore File:**


##### **Steps:**

1. Open the terminal or Git Bash in your project directory.
2. Create a .gitignore file:

```
KRISHNA@victus MINGW64 /d/scm (main)
$ touch .gitignore

KRISHNA@victus MINGW64 /d/scm (main)
$ nano .gitignore
```

3. Open the file in a text editor and add the files/directories to ignore. Example:



```
MINGW64:/d/scm
GNU nano 8.2 .gitignore Modified
# Ignore compiled files
*.class

# Ignore log files
*.log

# Ignore node_modules
node_modules/
```



4. Save the file and commit it to the repository:

```
KRISHNA@victus MINGW64 /d/scm (main)
$ git add .gitignore
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the ne
xt time Git touches it

KRISHNA@victus MINGW64 /d/scm (main)
$ git commit -m "added .gitignore file"
[main 8caa290] added .gitignore file
1 file changed, 9 insertions(+)
create mode 100644 .gitignore

KRISHNA@victus MINGW64 /d/scm (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:krishnanautiyal/g10scm.git
8ffe3dc..8caa290 main -> main
```



## 2. Creating a README File:

### Steps:

1. Create a README.md file in the repository:

```
KRISHNA@victus MINGW64 /d/scm (main)
$ touch README.md

KRISHNA@victus MINGW64 /d/scm (main)
$ nano README.md
```

2. Open the file in a text editor and write basic project details. Example:

```
GNU nano 8.2 README.md
#SCM task 1.2

# Practicals

1.
Clone, Fork, and Create Pull Requests

2.
Using .gitignore and README Files

3.
Scenario-Based Merge Conflict Resolution

# lorem ipsum |
```

3. Save the file and commit it to the repository

```
KRISHNA@victus MINGW64 /d/scm (main)
$ git add README.md
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it

KRISHNA@victus MINGW64 /d/scm (main)
$ git commit -m "added README.md"
[main f42e9de] added README.md
1 file changed, 14 insertions(+)
create mode 100644 README.md

KRISHNA@victus MINGW64 /d/scm (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 422 bytes | 211.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:krishnanautiyal/g10scm.git
8caa290..f42e9de main -> main
```

## Practical No. : 7

**Aim:** Scenario-Based Merge Conflict Resolution.

### **Theory:**

#### **What is a Merge Conflict?**

A merge conflict occurs in Git when two branches have made changes to the same line of a file, and Git cannot automatically determine which change to keep. This usually happens when multiple developers work on the same repository and merge their changes.

#### **Common Scenarios Causing Merge Conflicts:**

##### 1. Concurrent Changes in the Same File:

- Two users edit the same line in a file in different branches and try to merge them.

##### 2. File Deleted in One Branch, Modified in Another:

- One branch deletes a file while another branch modifies it.

##### 3. Different Changes in the Same File Structure:

- Changes in indentation or formatting differences can also trigger conflicts.

### **Procedure:**

#### **Scenario : Editing the Same Line in Two Branches**

##### Steps to Reproduce:

1. Create a repository and a new branch.
2. Modify a file (**example.txt**) in both the master and feature-branch separately.
3. Commit changes in both branches.
4. Try merging the feature-branch into master:

```
KRISHNA@victus MINGW64 /d/newfolder (master)
$ git init
Reinitialized existing Git repository in D:/newfolder/.git/

KRISHNA@victus MINGW64 /d/newfolder (master)
$ echo "hello world" >> newfile.txt

KRISHNA@victus MINGW64 /d/newfolder (master)
$ git add .
warning: in the working copy of 'newfile.txt', LF will be replaced by CRLF the n
ext time Git touches it

KRISHNA@victus MINGW64 /d/newfolder (master)
$ git commit -m "added newfile.txt"
[master d63129d] added newfile.txt
1 file changed, 1 insertion(+)
```

```

KRISHNA@victus MINGW64 /d/newfolder (master)
$ git checkout feature
Switched to branch 'feature'

KRISHNA@victus MINGW64 /d/newfolder (feature)
$ echo "changed newfile" >> newfile.txt

KRISHNA@victus MINGW64 /d/newfolder (feature)
$ git add .
warning: in the working copy of 'newfile.txt', LF will be replaced by CRLF the n
ext time Git touches it

KRISHNA@victus MINGW64 /d/newfolder (feature)
$ git commit -m "changed newfile in feature branch."
[feature db47be3] changed newfile in feature branch.
1 file changed, 1 insertion(+)

KRISHNA@victus MINGW64 /d/newfolder (feature)
$ git checkout master
Switched to branch 'master'

KRISHNA@victus MINGW64 /d/newfolder (master)
$ git merge feature
Auto-merging newfile.txt
CONFLICT (content): Merge conflict in newfile.txt
Automatic merge failed; fix conflicts and then commit the result.

```

### Resolution:

- Git will show a merge conflict message.
- Open the conflicting file in a text editor, and you will see conflict markers:

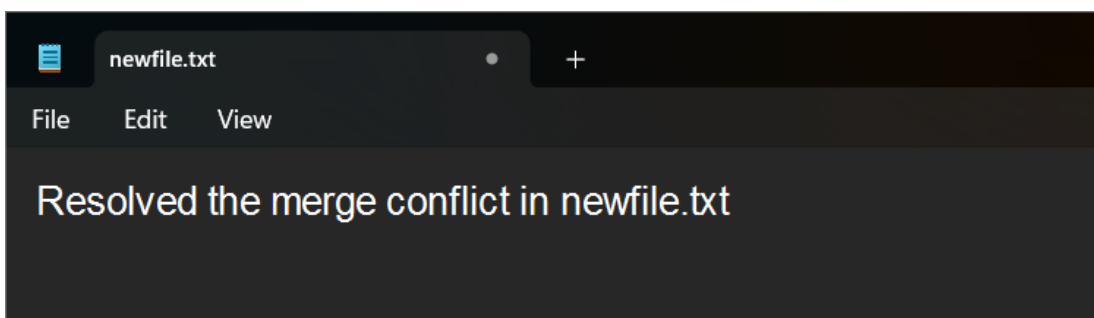


```

<<<<<< HEAD
hello world
=====
changed newfile
>>>>>> feature

```

- Manually edit the file to keep the desired changes, remove the conflict markers, and save the file.



```

Resolved the merge conflict in newfile.txt

```



- Stage, commit, and push the resolved file into the remote repository:

```
KRISHNA@victus MINGW64 /d/newfolder (master|MERGING)
$ git add .

KRISHNA@victus MINGW64 /d/newfolder (master|MERGING)
$ git commit -m "resolved the merge conflict in newfile.txt"
[master d53cd64] resolved the merge conflict in newfile.txt

KRISHNA@victus MINGW64 /d/newfolder (master)
```

```
KRISHNA@victus MINGW64 /d/newfolder (master)
$ git push origin master
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 1.05 KiB | 214.00 KiB/s, done.
Total 12 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/krishnanautiyal/newfolder.git
 * [new branch]      master -> master
```