Exp No. 3	**VERSION CONTROL IN SOFTWARE APPLICATION DEVELOPMENT**	

Aim

The aim is to provide a concise guide for utilizing Git version control system, covering fundamental commands and workflows including tracking changes, committing snapshots, creating branches, and merging changes.

Description

The provided description offers a comprehensive overview of Git, a version control system, detailing its importance in tracking modifications to files and categorizing version control systems into local, centralized, and distributed types. It further outlines key steps in version control, including adding files to tracking, committing changes, and pushing them to remote repositories. Essential Git commands such as `git add` and `git commit` are explained, along with their roles in staging and snapshotting changes. Additionally, it covers branch management, including creating and merging branches, and concludes with instructions for pushing changes to GitHub.

What is a "version control system"?

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

Types of Version Control Systems:

- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Three important steps of version control:

- **git add** changed files to version control tracking.
- **git commit** the changed files to create a unique snapshot of the local repository.
- **git push** those changed files from the local copy of a repository to the cloud

Check the Status of Changes Using GIT Status

Once you start working, you can use the **git status** command to check what changes are being identified by **git**.

To practice working with this command, use the **terminal** to navigate to your git practice repository:

```
$ cd practice-git-skillz
```

Next, run git status.

```
$ git status

On branch main
Your branch is up to date with

'origin/main'.nothing to commit,

working tree clean
```

Notice that when you run git status it returns: **working tree clean**. T his means that there areno changes to any files in your repo - YET.

Next, open and make a small change to the README.md file in a text editor. Then, run the command git status to check that changes have been made to your file(s).

```
git status
On branch main
Your branch is up-to-date with
'origin/main'.Changes not staged for
commit:
```

```
            modified:  README.md

no changes added to commit (use "git add" and/or "git commit -a")
```
The output from the git status command above indicates that you have modified a file (e.g. README.md) that can be added to version control.

Important Git Commands

These two commands make up the bulk of many workflows that use **git** for version control:

- git add: takes a modified file in your working directory and places the modified version in a staging area for review.
- git commit: takes everything from the staging area and makes a permanent snapshot of the current state of your repository that has a unique identifier.

Add Changed Files Using git add

After making changes, you can add either an individual file or groups of files to version control tracking. To add a single file, run the command:

```
git add file-name.extension
```

For example, to add the README.md file, you would use:

```
git add README.md
```

You can also add all of the files that you have edited at the same time using:

```
git add .
```

Commit Changed Files Using git commit

Once you are ready to make a snapshot of the current state of your repository (i.e. move changes from staging area), you can run git commit. The git commit command requires a commit message that describes the snapshot (i.e. changes) that you made in that commit. A commit message should outline what changed and why. These messages:

1. help collaborators and your future self understand what was changed and why.
2. allow you and your collaborators to find (and undo if necessary) changes that were previously made.

When you are not committing a lot of changes, you can create a short one line commit message using the -m flag as follows:

```
git commit -m "Update title and author name in homework for week 3"
```

Creating branches

To keep track of changes to this file using **git**, you need to:

1. Clone the repository.
2. Move into the cloned repository
3. Create a new branch using the command (replace *feature-branch* with your desired branch name).

   git checkout -b feature-branch
4. Make modifications to files in your project.
5. Use git add to add the changes to the staging area
6. Commit the changes with a meaningful message

Merging branches

To keep track of changes to this file using **git**, you need to:

1. Switch back to the main branch.

   git checkout main
2. Merge the branch into the main branch

   git merge feature-branch
3. Resolve conflicts (if necessary)

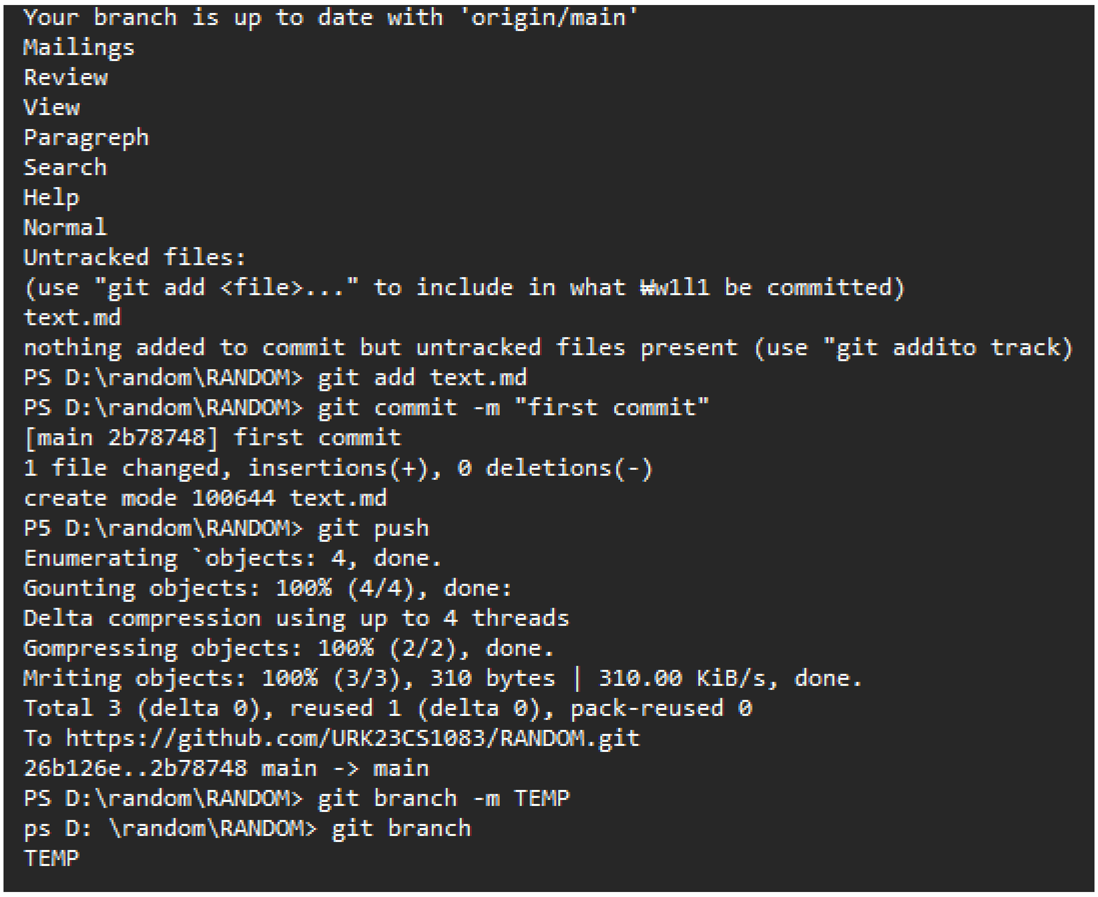
   i. Open the conflicting files and resolve the conflicts manually.

   ii. After resolving conflicts, add the changes to the staging area and commit:

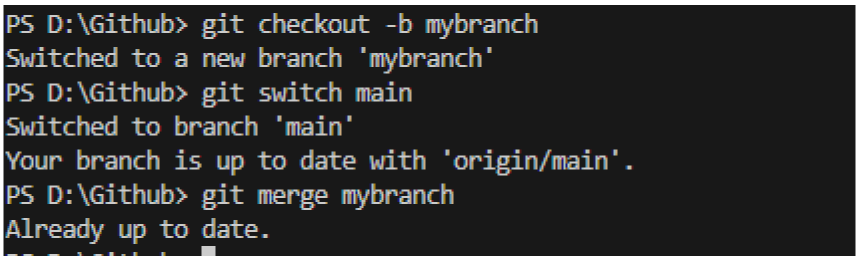
   git add .
   git commit -m "Merge feature-branch into main"
4. Push changes to github using the following command.

   git push origin main

OUTPUT:



```
Your branch is up to date with 'origin/main'
Mailings
Review
View
Paragreph
Search
Help
Normal
Untracked files:
(use "git add <file>..." to include in what Ww1l1 be committed)
text.md
nothing added to commit but untracked files present (use "git addito track)
PS D:\random\RANDOM> git add text.md
PS D:\random\RANDOM> git commit -m "first commit"
[main 2b78748] first commit
1 file changed, insertions(+), 0 deletions(-)
create mode 100644 text.md
P5 D:\random\RANDOM> git push
Enumerating `objects: 4, done.
Gounting objects: 100% (4/4), done:
Delta compression using up to 4 threads
Gompressing objects: 100% (2/2), done.
Mriting objects: 100% (3/3), 310 bytes | 310.00 KiB/s, done.
Total 3 (delta 0), reused 1 (delta 0), pack-reused 0
To https://github.com/URK23CS1083/RANDOM.git
26b126e..2b78748 main -> main
PS D:\random\RANDOM> git branch -m TEMP
ps D: \random\RANDOM> git branch
TEMP
```



```
PS D:\Github> git checkout -b mybranch
Switched to a new branch 'mybranch'
PS D:\Github> git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS D:\Github> git merge mybranch
Already up to date.
```

RESULT :Hence version control in software development of git was completed.