

Experiment 03 - Git Commands

Roll No.	24
Name	Iyer Sreekesh Subramanian
Class	D15-A
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>

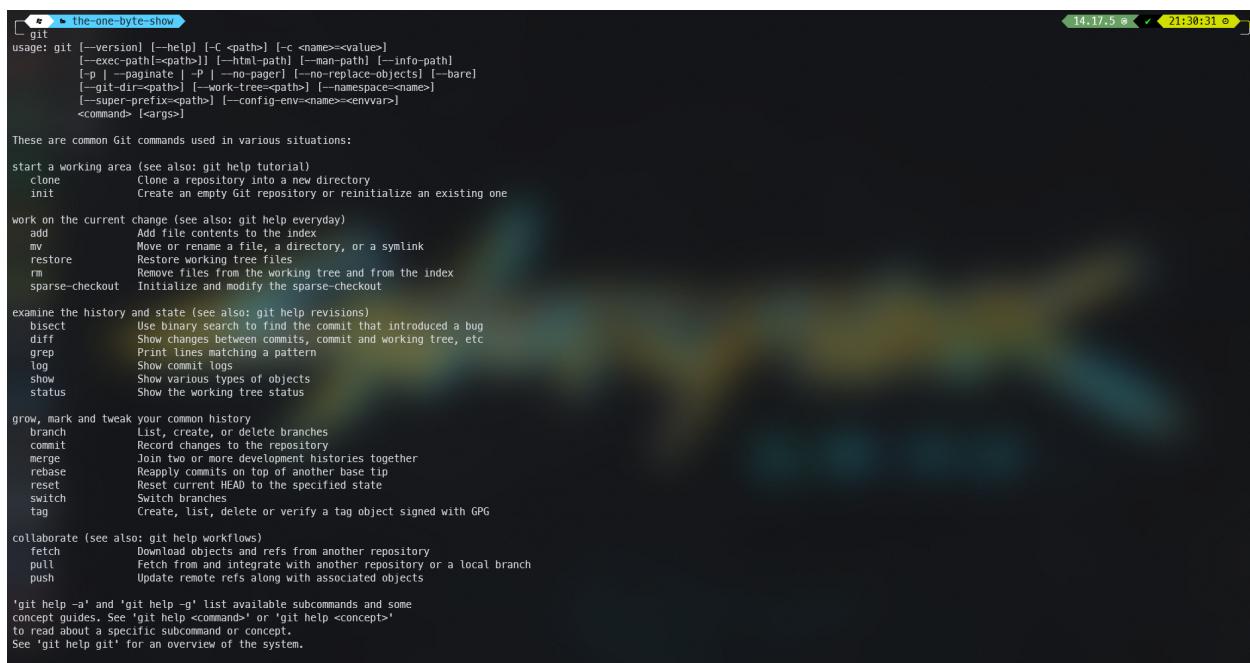
Aim:

To Perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet

Introduction:

Git commands are used like utilities to interact with our version control system

After installing git, one can get the list of all available commands by typing '*git --help*' in the CLI.



```

git --help [options] <command> [<args>]
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
          [--exec-path=<path>] [--html-path] [--man-path] [--info-path]
          [-p] [--paginate] [-P] [--no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          [--super-prefix=<path>] [--config-env=<name>=<envvar>]
          [<command>] [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help everyday)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv        Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm        Remove files from the working tree and from the index
  sparse-checkout Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  diff      Show changes between commits, commit and working tree, etc
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  commit   Record changes to the repository
  merge   Join two or more development histories together
  rebase   Reapply commits on top of another base tip
  reset   Reset current HEAD to the specified state
  switch  Switch branches
  tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch   Download objects and refs from another repository
  pull    Fetch from and integrate with another repository or a local branch
  push    Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'.
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

```

Following is the list of frequently used commands:

1. *git add*

- a. Moves changes from the working directory to the staging area.
- b. This gives you the opportunity to prepare a snapshot before committing it to the official history.

2. *git branch*

- a. This command is your general-purpose branch administration tool.
- b. It lets you create isolated development environments within a single repository.

3. *git checkout*

- a. In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches.

- b. Combined with the basic Git commands, it's a way to work on a particular line of development.

4. *git clean*

- a. Removes untracked files from the working directory.
- b. This is the logical counterpart to git reset, which (typically) only operates on tracked files.

5. *git clone*

Creates a copy of an existing Git repository.

- a. Cloning is the most common way for developers to obtain a working copy of a central repository.

6. *git commit*

Takes the staged snapshot and commits it to the project history.

- a. Combined with git add, this defines the basic workflow for all Git users.

7. *git commit --amend*

- a. Passing the --amend flag to git commit lets you amend the most recent commit.
- b. This is very useful when you forget to stage a file or omit important information from the commit message.

8. *git config*

- a. A convenient way to set configuration options for your Git installation.
- b. You'll typically only need to use this immediately after installing Git on a new development machine.

9. *git fetch*

- a. Fetching downloads a branch from another repository, along with all of its associated commits and files. But, it doesn't try to integrate anything into your local repository.
- b. This gives you a chance to inspect changes before merging them with your project.

10. *git init*

- a. Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

11. *git log*

- a. Lets you explore the previous revisions of a project.
- b. It provides several formatting options for displaying committed snapshots.

12. *git merge*

- a. A powerful way to integrate changes from divergent branches.
- b. After forking the project history with the git branch, git merge lets you put it back together again.

13. *git pull*

- a. Pulling is the automated version of git fetch.
- b. It downloads a branch from a remote repository, then immediately merges it into the current branch.
- c. This is the Git equivalent of svn update.

14. *git push*

- a. Pushing is the opposite of fetching (with a few caveats).
- b. It lets you move a local branch to another repository, which serves as a convenient way to publish contributions.
- c. This is like svn commit, but it sends a series of commits instead of a single changeset.

15. *git rebase*

- a. Rebasing lets you move branches around, which helps you avoid unnecessary merge commits.
- b. The resulting linear history is often much easier to understand and explore.

16. *git rebase -i*

- a. The `-i` flag is used to begin an interactive rebasing session.
- b. This provides all the benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way.

17. *git reflog*

- a. Git keeps track of updates to the tip of branches using a mechanism called reflog.
- b. This allows you to go back to changesets even though they are not referenced by any branch or tag.

18. *git remote*

- a. A convenient tool for administering remote connections.
- b. Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.

19. *git reset*

- a. Undoes changes to files in the working directory.
- b. Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.

20. *git revert*

- a. Undoes a committed snapshot.
- b. When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the codebase.

21. *git status*

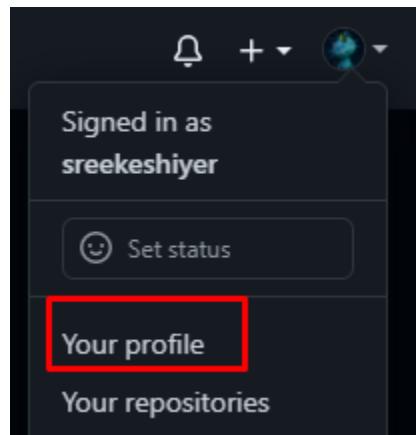
- a. Displays the state of the working directory and the staged snapshot.
- b. You'll want to run this in conjunction with `git add` and `git commit` to see exactly what's being included in the next snapshot.

Just getting started – GitHub

Just using GitHub, please complete each task below. This requires that you have already set up an account.

1. Change your profile picture and name

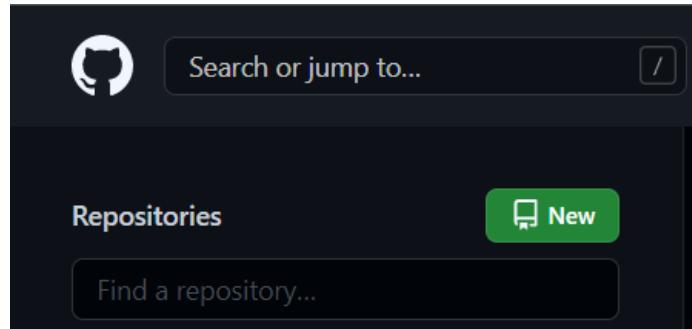
- After logging in to your GitHub account on github.com, click on profile dropdown at the top right corner and select ‘Your Profile’.



- Click on your default avatar, this will take you to the profiles page from where you can change your profile avatar and add your Name

2. Create a repository named **GitHubExpt03_grno**

- In order to create a new repository, navigate to the homepage while logged in to your account.
- Click on the button 'New'

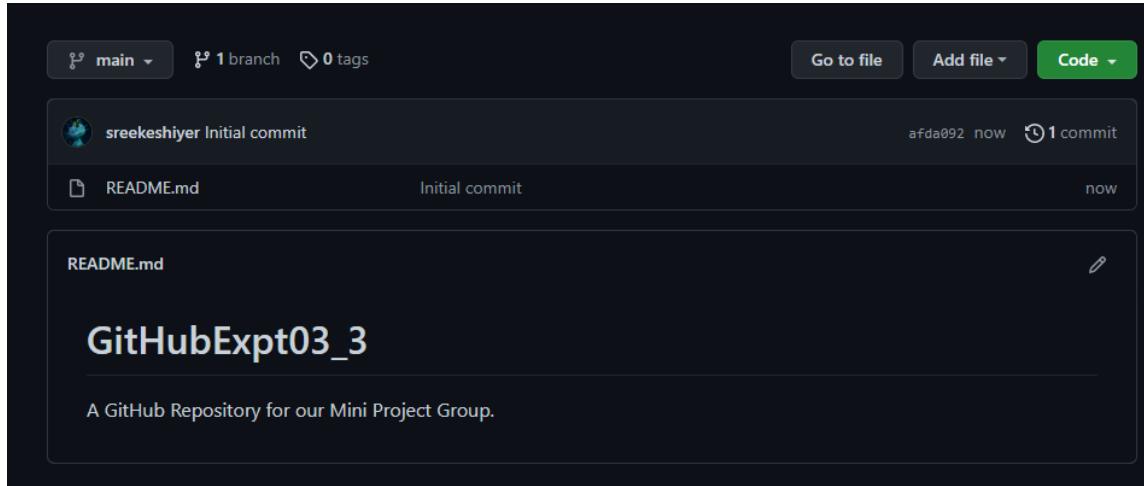


- c. You will be prompted to fill in information like
 - i. Name of the repository
 - ii. Description
 - iii. Visibility of repository (private or public)
 - iv. Initialization options
- d. Fill in the details as per the requirements and click 'Create repository'

A screenshot of the GitHub "Create a new repository" form. The title "Create a new repository" is at the top. Below it is a sub-instruction: "A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository." The "Owner" field shows "sreekeshiyer" with a dropdown arrow. The "Repository name" field contains "GitHubExpt03_3" with a green checkmark icon. A note below says, "Great repository names are short and memorable. Need inspiration? How about verbose-waddle?" The "Description (optional)" field contains "A GitHub Repository for our Mini Project Group.". Below these fields are two radio buttons for visibility: "Public" (disabled) and "Private" (selected). A note next to "Private" says, "You choose who can see and commit to this repository." Under the heading "Initialize this repository with:", there are three checkboxes:

- Add a README file**: "This is where you can write a long description for your project. [Learn more](#)."
- Add .gitignore**: "Choose which files not to track from a list of templates. [Learn more](#)."
- Choose a license**: "A license tells others what they can and can't do with your code. [Learn more](#)."

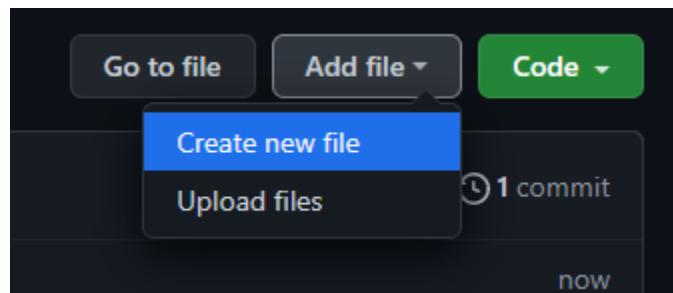
At the bottom is a large green "Create repository" button.



A Repository is created.

3. Create a file in the above directory named “intro.txt” with your name on the first line.

- To create a file, click on ‘Add File’ and select 'create new file'

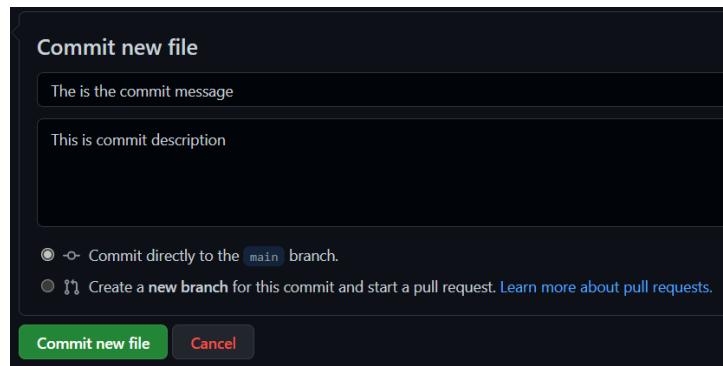


- Enter the name of the file, select the branch you want to create the file in (default is main)
- Enter the content of the files

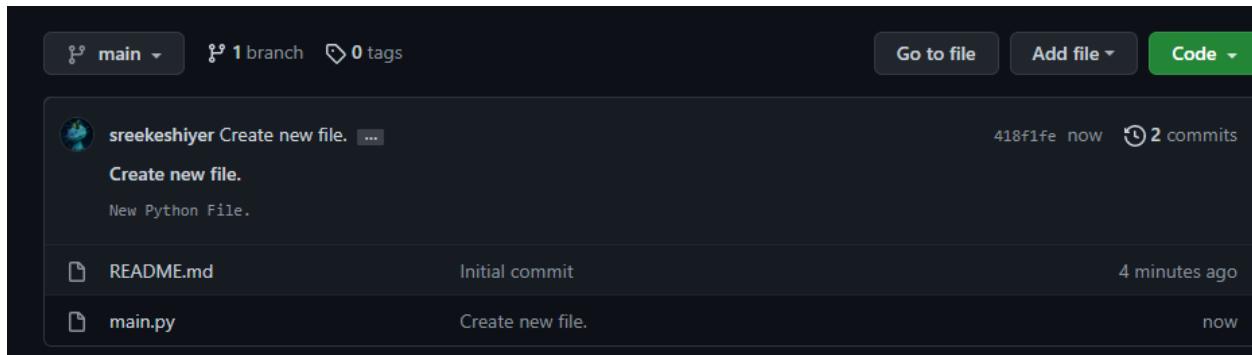


```
GitHubExpt03_3 / main.py in main
Edit new file Preview
1 print('Hello World')
2
3
4 |
```

- d. After adding the content, navigate to the bottom of the page
- e. Add commit message and description, select the branch you wish to add the file then click 'Commit new file'



The new file that was created is now seen in the repository with the latest commit and details



main ▾ 1 branch 0 tags Go to file Add file ▾ Code ▾

sreekeshiyer Create new file. ... 418f1fe now 2 commits

Create new file.
New Python File.

README.md Initial commit 4 minutes ago

main.py Create new file. now

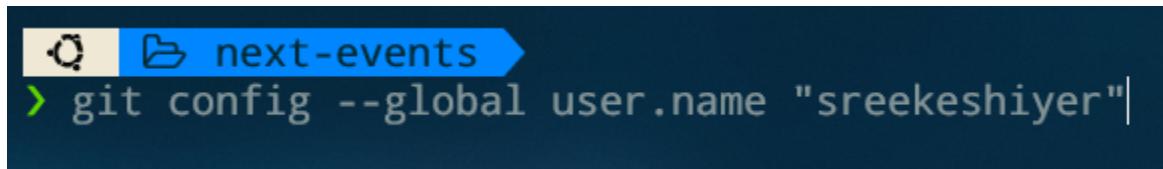
Github repository: https://github.com/Aamir-Ansari-almost/GitHubExpt03_01

Just getting started – Git Bash

Just using Git, please complete each task below and write the answer in the blank box beside the question. This requires that you have already installed Git.

1. Setup user name

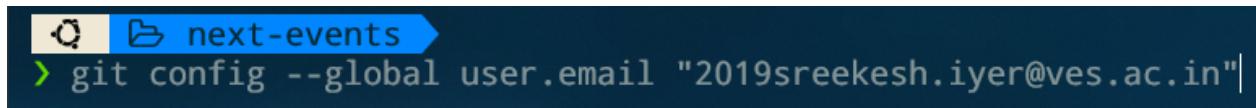
- a. Open git bash cli
- b. Use the following command to configure your username
 - i. `git config --global user.name "Your UserName"`



```
next-events
> git config --global user.name "sreekeshiyer"
```

2. Setup user email

- a. Use the following command to configure your email address
 - i. `git config --global user.email "Your EmailID"`

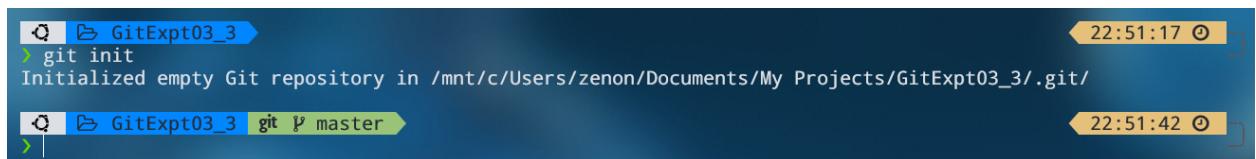


```
next-events
> git config --global user.email "2019sreekesh.iyer@ves.ac.in"
```

*PS: it is advisable to enter github username and password from windows credential. Since there have been some issue with github to add username and email address from command line interface

3. Create your local repository named GitExpt03_grno

- a. To initialise a repository use the following command
 - i. `'git init'`



```
GitExpt03_3
> git init
Initialized empty Git repository in /mnt/c/Users/zenon/Documents/My Projects/GitExpt03_3/.git/
GitExpt03_3 git: master
>
```

- b. To rename repository, cd into the .git directory and put in the required repository name in the 'description' file

```
Q  ⌂ .git  git  ↵ master ?12
> ls
HEAD  branches  config  description  hooks  info  objects  refs
```

4. Add a file in above local repository named file_grno_rollno and add some content to the file Add it to the staging state

- Create a file normally with a text editor, and add contents to the file

```
Q  ⌂ GitExpt03_3  git  ↵ master
> code file_3_24.txt
```

- Use the following command to check if some change is made
 - 'git status'
- Use the following command to add the changes
 - 'git add <filename>'

```
Q  ⌂ GitExpt03_3  git  ↵ master
> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file_3_24.txt

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

Q  ⌂ GitExpt03_3  git  ↵ master ?1
> git add file_3_24.txt

Q  ⌂ GitExpt03_3  git  ↵ master +1
>
```

5. Commit the files

- Use the following command to commit the changes
 - 'git commit -m "<your message>"'

```
Q ➜ GitExpt03_3 git 🌐 master +1
> git commit -m "New Commit"
[master (root-commit) 076ccb3] New Commit
 1 file changed, 1 insertion(+)
 create mode 100644 file_3_24.txt

Q ➜ GitExpt03_3 git 🌐 master
>
```

6. Create a scenario for modifications/additions of the multiple files. Show the advantage of staging state.

- If we make some changes at this point in the process, git will show them as an untracked file.
- A demonstration is shown below

```
Q ➜ GitExpt03_3 git 🌐 master
> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   file_3_24.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

7. Display the log

- Git log is a utility tool to review and read a history of everything that happens to a repository
- Use the following command to view logs
 - 'git log'

```
commit 076ccb3a745cfcbfeac7b4dd95dcd92db3a5b3a3 (HEAD -> master)
Author: sreekeshiyer <sreekesh.subramanian@gmail.com>
Date:   Fri Aug 27 23:02:31 2021 +0530

  New Commit
(END)
```

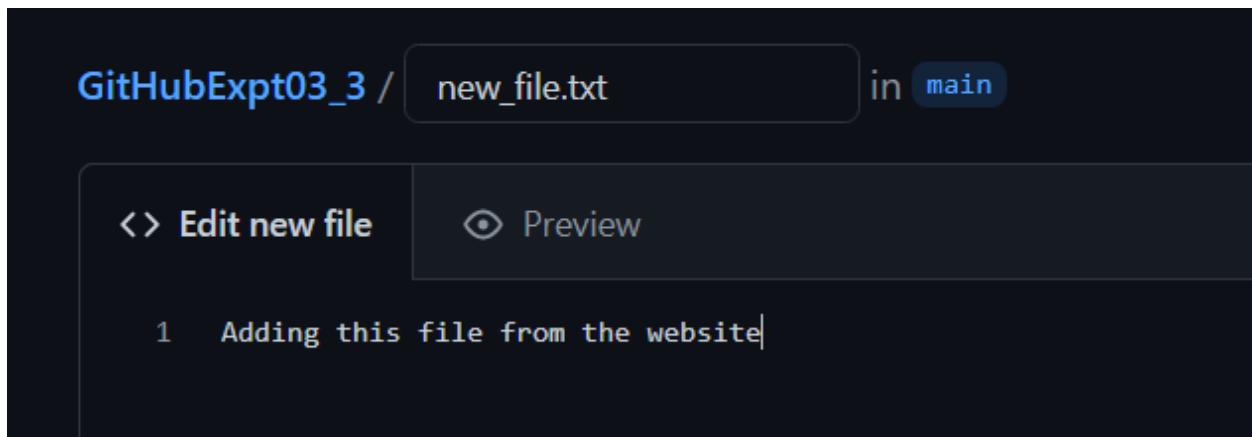
8. Download your GitHub repository created in table-1 named GitHubExpt03_grno

- a. To download a repository use the following command
 - i. `'git clone <URL of repository>'`

```
git clone https://github.com/sreekeshiyer/GitHubExpt03_3.git
Cloning into 'GitHubExpt03_3'...
Username for 'https://github.com': sreekeshiyer
Password for 'https://sreekeshiyer@github.com':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 1.30 KiB | 37.00 KiB/s, done.
```

9. Make some changes in the GitHub file.

- a. Add some file directly through github website as shown before



10. Pull the latest files from your GitHub repository

- a. Use the following command to check if some changes have been made in the repository and pull those changes
 - i. `'git fetch'`
 - ii. `'git status'`
 - iii. `'git pull'`

```
Q GitHubExpt03_3 🐫 main ➔
> git fetch
Username for 'https://github.com': sreekeshiyer
Password for 'https://sreekeshiyer@github.com':
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 730 bytes | 25.00 KiB/s, done.
From https://github.com/sreekeshiyer/GitHubExpt03_3
  418f1fe..8c6b510  main      -> origin/main

Q GitHubExpt03_3 🐫 main ↓1 ➔
> git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

Q GitHubExpt03_3 🐫 main ↓1 ➔
> git pull
Username for 'https://github.com': sreekeshiyer
Password for 'https://sreekeshiyer@github.com':
Updating 418f1fe..8c6b510
Fast-forward
  new_file.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 new_file.txt
```

```
Q GitHubExpt03_3 🐫 main ➔
> ls
README.md  main.py  new_file.txt
```

11. Make some changes in the local file of the GitHub repository

- Make some file normally in your preferred text editor and add content to it
- You can see that the file is created in the local repository

```
Q GitHubExpt03_3 main
> code new_local_file.txt

Q GitHubExpt03_3 main ?1
> ls
README.md  main.py  new_file.txt  new_local_file.txt
```

12. Update the changes back to Github Repository

- Use the following commands to push all local changes to a remote repository
 - 'git add <filename>'
 - 'git commit -m "<your message>"'
 - 'git push'

```
Q GitHubExpt03_3 main ?1
> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
(use "git add <file>..." to include in what will be committed)
  new_local_file.txt

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

Q GitHubExpt03_3 main ?1
> git add new_local_file.txt

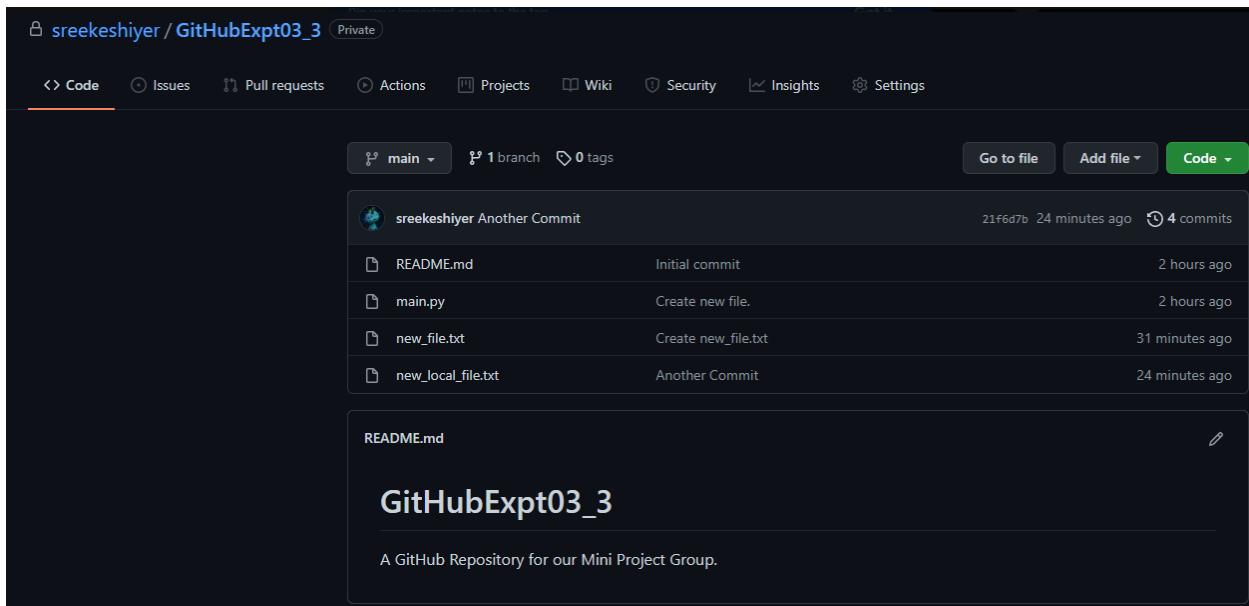
Q GitHubExpt03_3 main +1
> git commit -m "Another Commit"
[main 21f6d7b] Another Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 new_local_file.txt

Q GitHubExpt03_3 main ?1
> git push
Username for 'https://github.com': sreekeshiyer
Password for 'https://sreekeshiyer@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 95.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/sreekeshiyer/GitHubExpt03_3.git
  8c6b510..21f6d7b  main -> main

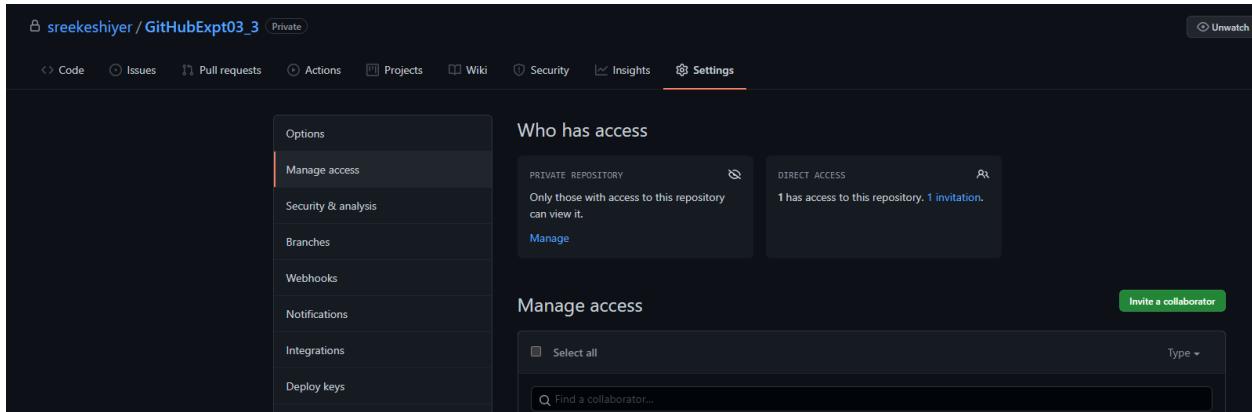
Q GitHubExpt03_3 main
```

Team set up in GitHub

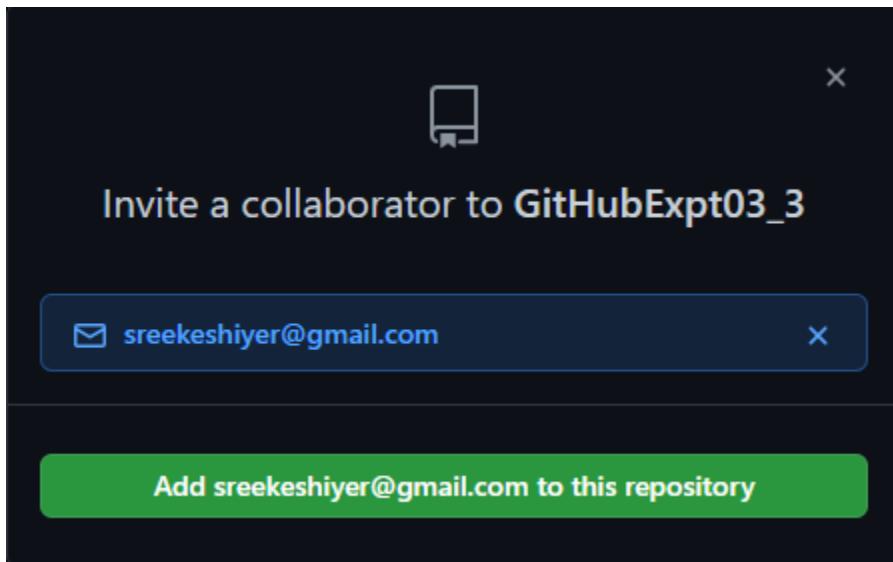
1. Using one person (“lead developer”) on your team, determine who will house the files in your project.
 - a. Create a new repository on GitHub as shown previously
 - b. Make sure to set the repository as Private, by doing the following we are ensuring that anyone outside our team can not access the repository



2. That “lead” will create a directory “Project_GrNo” using GitHub.
 - a. Now you have created the following repository
3. Make other team members as collaborators.
 - a. Go to setting
 - i. Manage access
 - ii. Click on the 'Invite a collaborator' button



- b. Enter GitHub username or email address to add collaborators



4. Using Git, have the others create and add a unique file to the lead's Project_GrNo directory.
a. The collaborator will get an email and prompt to accept the invitation



- b. Newly added collaborators can now access the repository and add files just like shown before

A screenshot of a GitHub repository page for "sreekeshiyer/GitHubExpt03_3". The page header shows the repository name and a "Private" status. Below the header, a message says "You now have push access to the sreekeshiyer/GitHubExpt03_3 repository." The main navigation bar includes links for Code (which is underlined in red), Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights.

The Lead Developer *sreekeshiyer* added *sreekesh-iyer* to the *Team*. We can see that there is a commit from a collaborator in the repository created by the lead developer.

Conclusion

In this experiment, we learned how to use Git Commands using the Git CLI and we learned how to create repositories, set up teams and collaborate with people on GitHub.