Subject Name: **Source Code Management**

Subject Code: **22CS003**

Session: **2023-2024**

Department: **DCSE**

CHITKARA UNIVERSITY

Submitted To:

Dr. Sharad Chauhan
Assistant Professor
Chitkara University, Punjab

Submitted By:

Komal
2310992042
Sem-II (2023-27)

CHITKARA
UNIVERSITY

Source Code Management File

Subject Name: **Source Code Management (SCM)**

Subject Code: **22CS003**

Cluster: **Beta**

Submitted By:

Name: Komal
Roll No. 2310992042
Group: 23-A

Task 1.1 Submission (Week 4)

1. Setting up of Git Client
2. Setting up GitHub Account
3. Generate logs
4. Create and visualize branches
5. Git lifecycle description

INDEX

Sr. No.	Practical Name	Teacher Sign
1	Setting up of Git Client	
2	Setting up GitHub Account	
3	Generating logs .	
4	Create and visualize branches	
5	Git lifecycle description	
6		
7		
8		
9		
10		
11		
12		

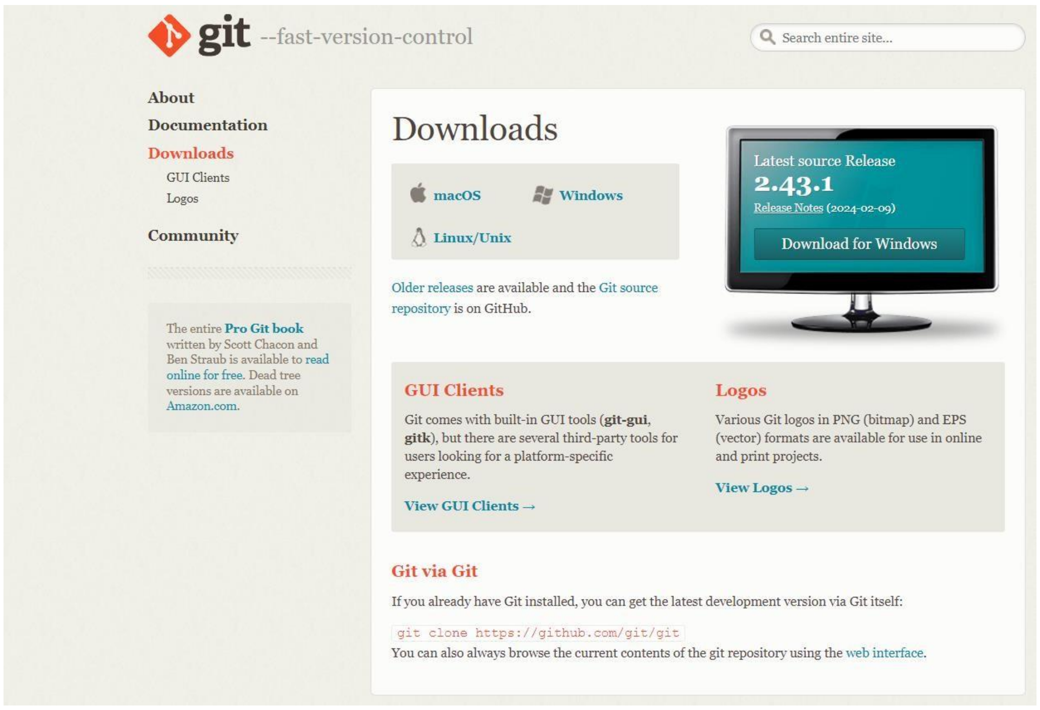
EXPERIMENT 1

Aim: Setting up of Git Client

Theory:
GIT: It's a Version Control System (VCS). It is a software or we can say a server by which we are able to track all the previous changes in the code. It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make, edit, recreate, copy or download any code on git hub using git.

Procedure: We can install Git on Windows, using the most official build which is available for download on the GIT's official website or by just typing (scm git) on any search engine. We can go on https://git-scm.com/download/win and can select the platform and bit-version to download. And after clicking on your desired bit-version or ios it will start downloading automatically.

Snapshots of download:



The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Older releases are available and the Git source repository is on GitHub.

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

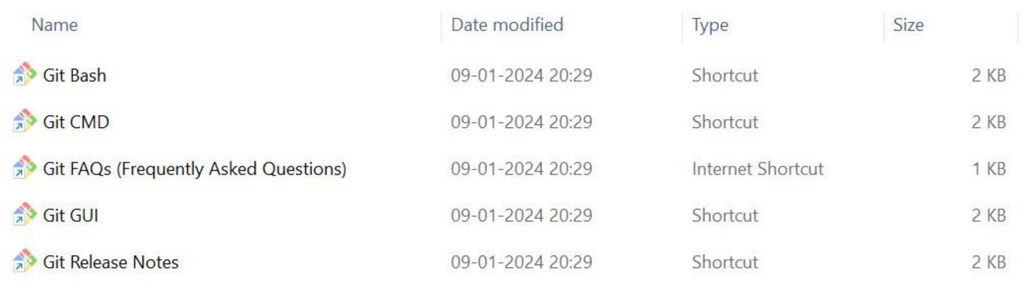
Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

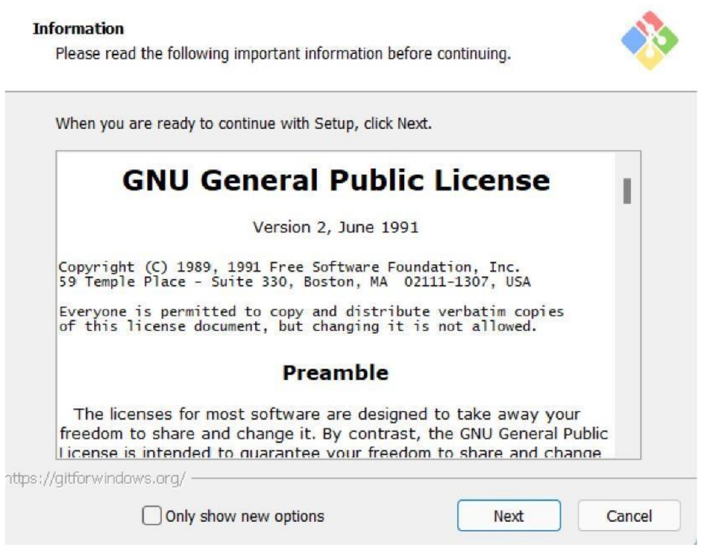
You can also always browse the current contents of the git repository using the web interface.

Fig-1.1 Opted for "64-bit Git for Windows Setup"



Name	Date modified	Type	Size
Git Bash	09-01-2024 20:29	Shortcut	2 KB
Git CMD	09-01-2024 20:29	Shortcut	2 KB
Git FAQs (Frequently Asked Questions)	09-01-2024 20:29	Internet Shortcut	1 KB
Git GUI	09-01-2024 20:29	Shortcut	2 KB
Git Release Notes	09-01-2024 20:29	Shortcut	2 KB

Fig-1.2 Git and its files in downloads



Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

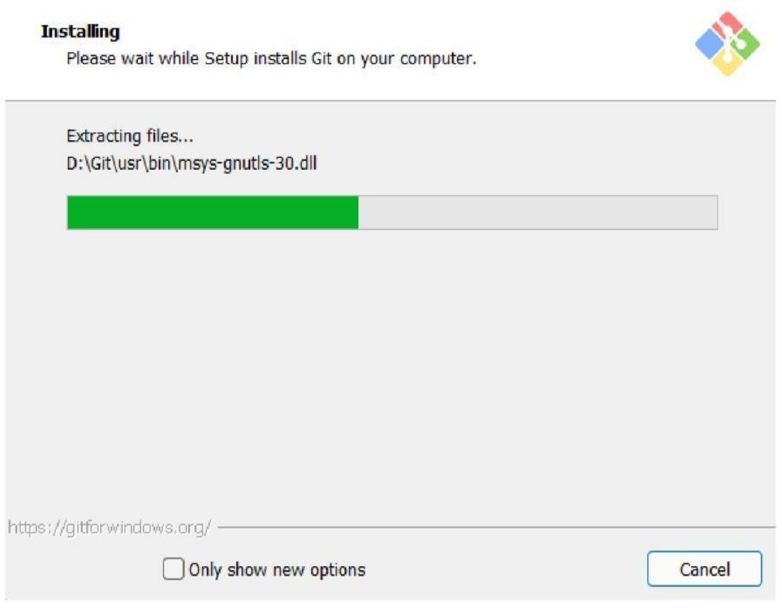
GNU General Public License

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

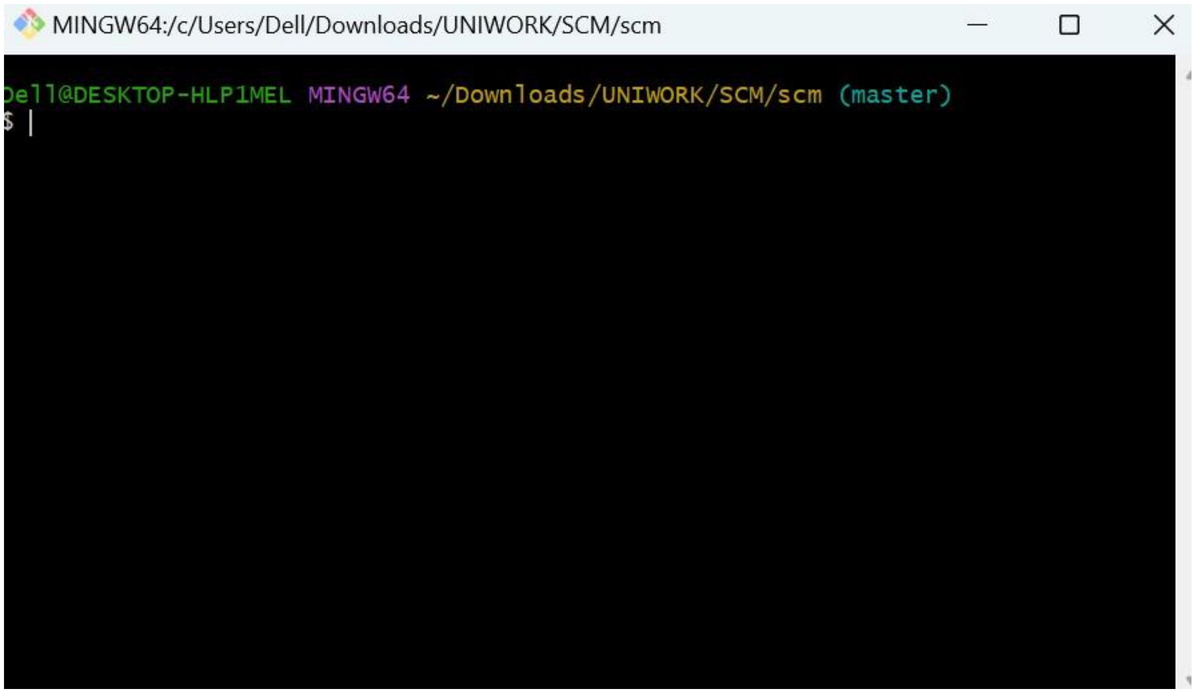
Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change



Installing
Please wait while Setup installs Git on your computer.

Extracting files...
D:\Git\usr\bin\msys-gnutls-30.dll



```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ |
```

Fig-1.5 Git Bash launched

EXPERIMENT 2

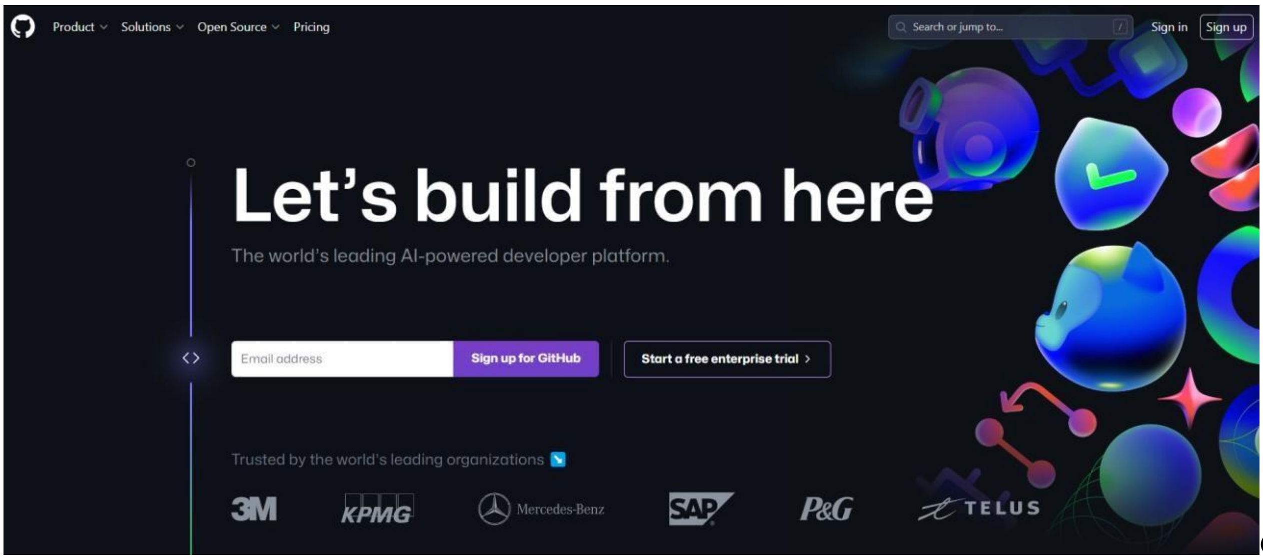
Aim: Setting up GitHub Account

Theory:

GitHub: GitHub is a website and cloud-based service (client) that helps an individual or developers to store and manage their code. We can also track as well as control changes to our or public code.

Advantages of GitHub: GitHub has a user-friendly interface and is easy to use.We can connect the git-hub and git but using some commands shown below in figure
001. Without GitHub we cannot use Git because it generally requires a host and if we are working for a project, we need to share it will our team members, which can only be done by making a repository. Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

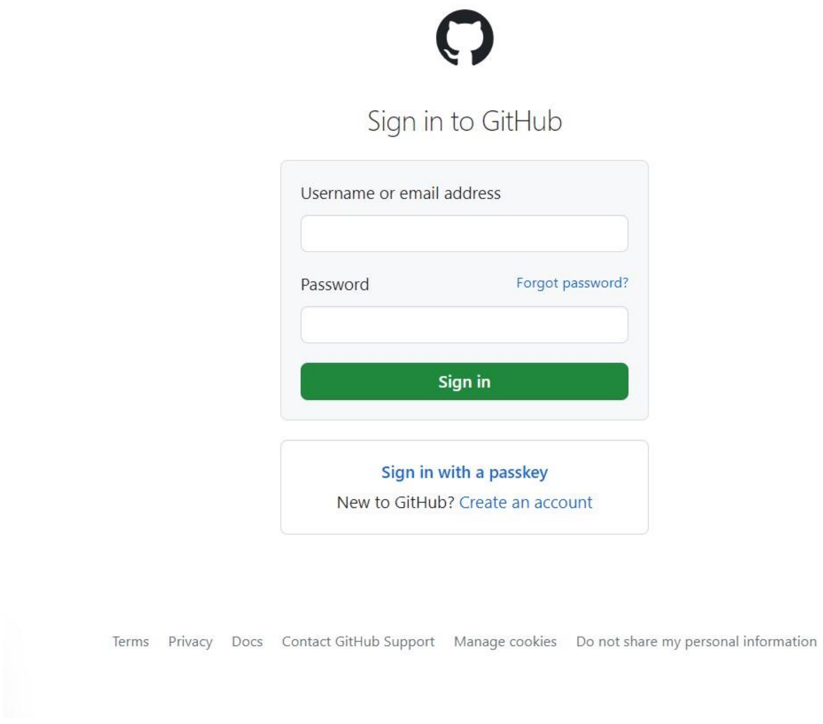
Procedure:
To make an account on GitHub, we search for GitHub on our browser or visit https://github.com/signup. Then, we will enter our mail ID and create a username and password for a GitHub account.



G

Fig-2.1 GitHub Sign Up

After visiting the link this type of interface will appear, if you already have an account, you can sign in and if not, you can create.



Sign in to GitHub

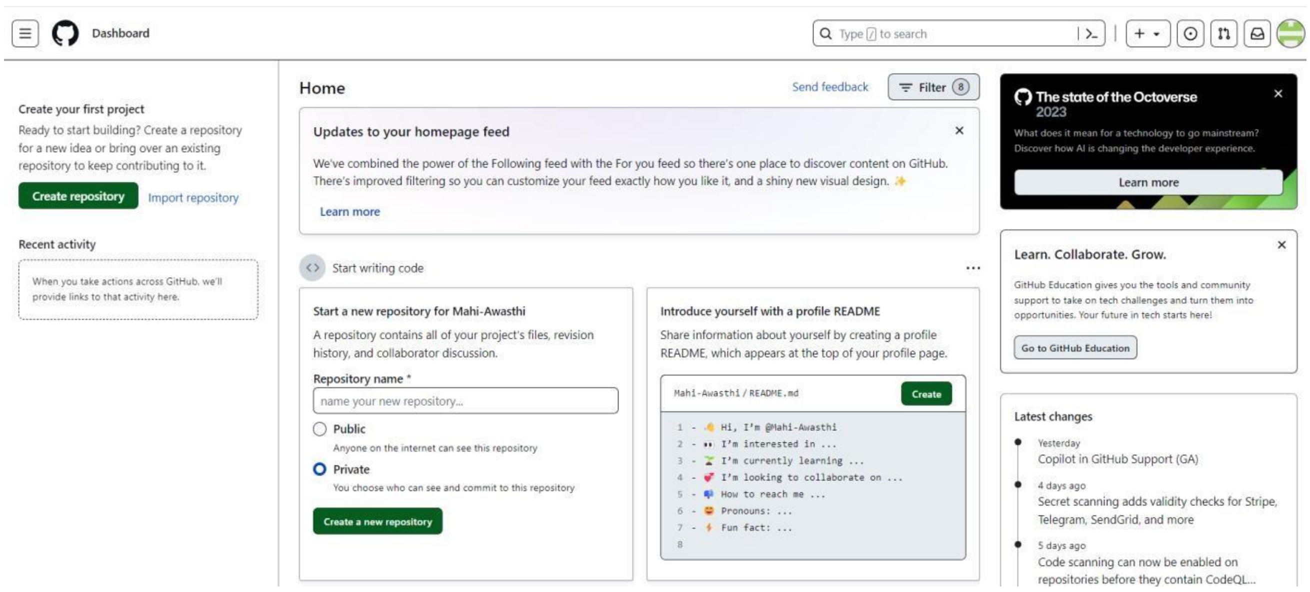
Username or email address

Password Forgot password?

Sign in with a passkey

New to GitHub? Create an account

Fig-2.2 GitHub Login



Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

Home

Updates to your homepage feed

We've combined the power of the Following feed with the For you feed so there's one place to discover content on GitHub. There's improved filtering so you can customize your feed exactly how you like it, and a shiny new visual design. ✨

Start a new repository for Mahi-Awasthi

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

name your new repository...

Public
Anyone on the internet can see this repository

[✓ Private]
You choose who can see and commit to this repository

Introduce yourself with a profile README

Share information about yourself by creating a profile README, which appears at the top of your profile page.

Mahi-Awasthi / README.md

```
- 👋 Hi, I'm @Mahi-Awasthi
- 👀 I'm interested in ...
- 🌱 I'm currently learning ...
- 💞️ I'm looking to collaborate on ...
- 📫 How to reach me ...
- 😄 Pronouns: ...
- ⚡ Fun fact: ...
```

The state of the Octoverse 2023

What does it mean for a technology to go mainstream? Discover how AI is changing the developer experience.

Learn. Collaborate. Grow.

GitHub Education gives you the tools and community support to take on tech challenges and turn them into opportunities. Your future in tech starts here!

Latest changes

- Yesterday
  Copilot in GitHub Support (GA)
- 4 days ago
  Secret scanning adds validity checks for Stripe, Telegram, SendGrid, and more
- 5 days ago
  Code scanning can now be enabled on repositories before they contain CodeQL...

Fig-2.3 GitHub Interface

EXPERIMENT 3

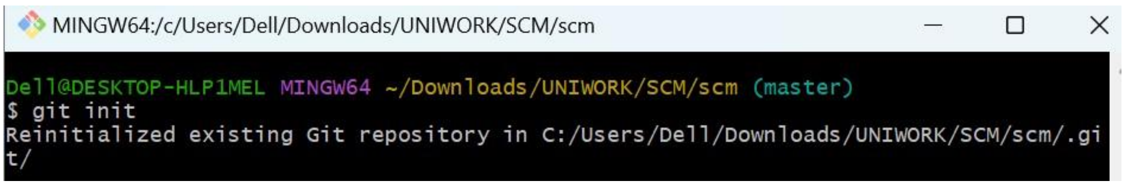
Aim: Program to generate log

Theory:

Logs: Logs are nothing but the history which we can see in git by using the code git log. It contains all the past commits, insertions and deletions in it which we can see any time. Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed.

Procedure:

First of all, create a local repository using Git. For this, you have to make a folder in your device, right click and select "Git Bash Here". This opens the Git terminal. To create a new local repository, use the command "git init" and it creates a folder ".git".



```
MINGW64:/c/Users/Dell/Downloads/UNIWORK/SCM/scm

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git init
Reinitialized existing Git repository in C:/Users/Dell/Downloads/UNIWORK/SCM/scm/.gi
t/
```

Fig-3.1 Git Init Command

When we use GIT for the first time, we have to give the user name and email so that if I am going to change in project, it will be visible to all.

For this, we use command:

"git config --global user.name Name" "git config --global user.email email"

For verifying the user's name and email, we use: "git config --global user.name"

"git config --global user.email"

Some Important Commands:-

a) ls → It gives the file names in the folder.
b) ls -lart → Gives the hidden files also.
c) git status → Displays the state of the working directory and the staged snapshot.
d) touch filename → This command creates a new file in the repository.
e) Clear → It clears the terminal.
f) rm -rf .git → It removes the repository.
g) git log → displays all of the commits in a repository's history

h) git diff → It compares my working tree to staging area.

Algorithm:

1. Check Git Version:

- Execute the command `**git --version**` in the terminal.
- Retrieve and display the installed Git version.

2. Configure Username and Email:

- Execute the commands: **git config --global user.name "Your Name"** **gitconfig—globaluser.email "your_email@example.com"** - Set the global username and email for Git.

3. Initialize a Git Repository:

- Navigate to the project directory in the terminal.
- Execute the command `**git init**`.
- Initialize a new Git repository in the current directory.

4. Create a New File:

- Create a new file with the help of **touch filename**.

5. Check Status of Git Repository:

- Execute the command `**git status**`.
- Check the current status of the Git repository, including any untracked or modified files.

6. Stage the File:

- Execute the command `**git add filename**`.
- Stage the specified file for the next commit.

7. Check Status Again:

- Execute the command `**git status**`.

- Verify the changes and the status of the repository after staging the file.

8. Commit the Changes:

- Execute the command `**git commit -m "file name"**`.
- Commit the staged changes with a descriptive message.

9. View Commit History:

- Execute the command `**git log**`.
- View the commit history in the terminal, showing commit hashes, authors, dates, and commit messages.

10. View Compact Commit History:

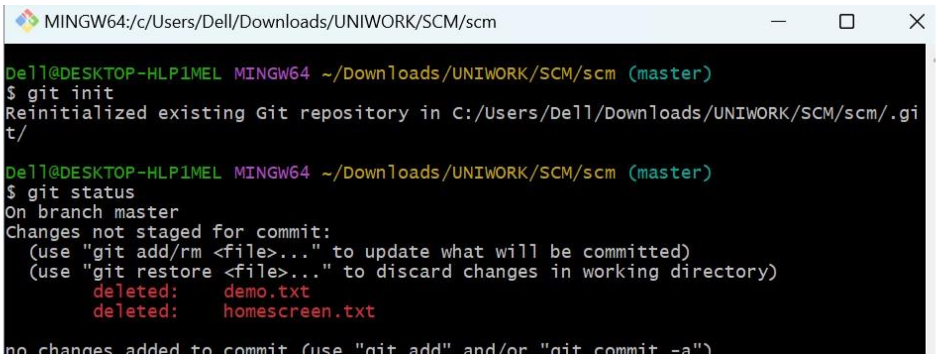
- Execute the command `**git log --oneline**`.
- View a compact version of the commit history, showing abbreviated commit hashes and commit messages.

11. View Commit History with Stats:

- Execute the command `**git log --stat**`.
- View the commit history with additional statistics, including the number of insertions and deletions per file.

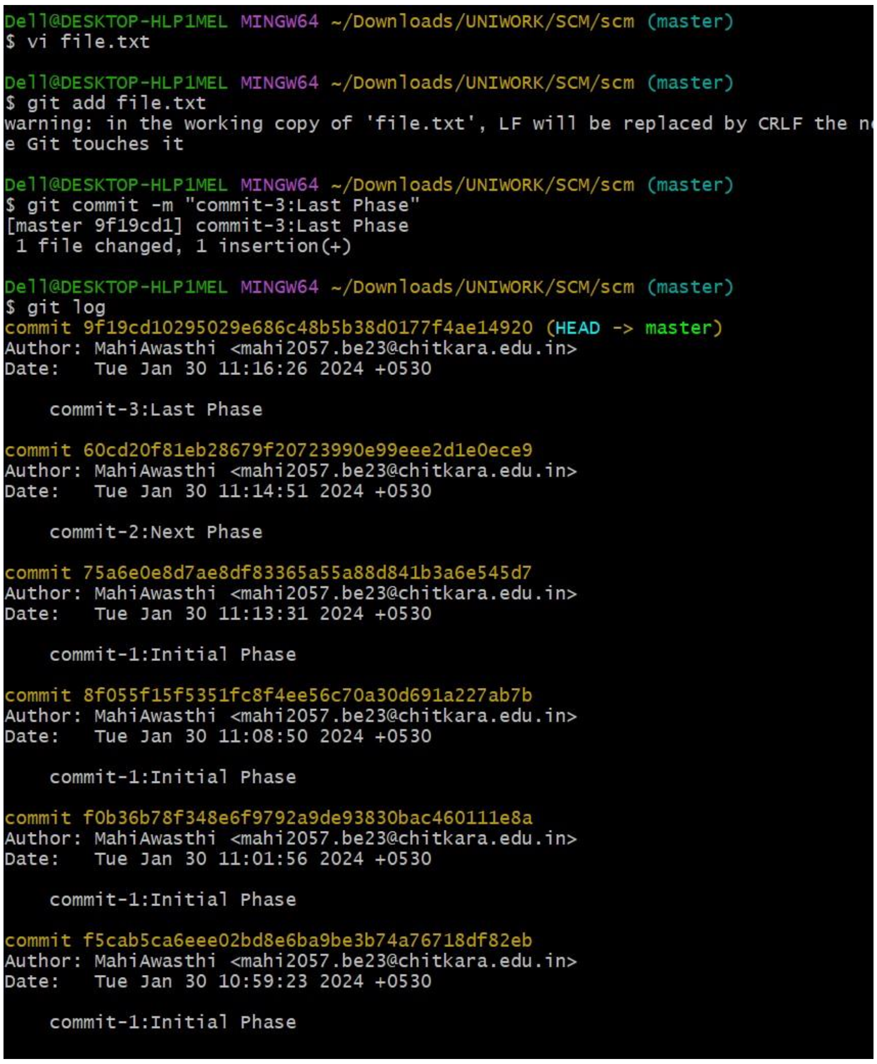
12. View Details of a Specific Commit:

- Execute the command `**git show commit_hash**`.
- View the detailed information about a specific commit, including changes made and the commit message.



```
MINGW64:/c/Users/Dell/Downloads/UNIWORK/SCM/scm

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git init
Reinitialized existing Git repository in C:/Users/Dell/Downloads/UNIWORK/SCM/scm/.gi
t/

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    demo.txt
        deleted:    homescreen.txt

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

Fig-3.2 Git Status



```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ vi file.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the n
e Git touches it

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git commit -m "commit-3:Last Phase"
[master 9f19cd1] commit-3:Last Phase
 1 file changed, 1 insertion(+)

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git log
commit 9f19cd10295029e686c48b5b38d0177f4ae14920 (HEAD -> master)
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 11:16:26 2024 +0530

    commit-3:Last Phase

commit 60cd20f81eb28679f20723990e99eee2d1e0ece9
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 11:14:51 2024 +0530

    commit-2:Next Phase

commit 75a6e0e8d7ae8df83365a55a88d841b3a6e545d7
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 11:13:31 2024 +0530

    commit-1:Initial Phase

commit 8f055f15f5351fc8f4ee56c70a30d691a227ab7b
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 11:08:50 2024 +0530

    commit-1:Initial Phase

commit f0b36b78f348e6f9792a9de93830bac460111e8a
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 11:01:56 2024 +0530

    commit-1:Initial Phase

commit f5cab5ca6eee02bd8e6ba9be3b74a76718df82eb
Author: MahiAwasthi <mahi2057.be23@chitkara.edu.in>
Date:   Tue Jan 30 10:59:23 2024 +0530

    commit-1:Initial Phase
```

Fig-3.3 Git Log

EXPERIMENT 4

Aim: Create and visualize branches

Theory:

Branching: A branch in Git is an independent line of work (a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.

Create branches: The main branch in git is called as master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches). Syntax:

For creating a new branch, git branch name by default is master branch.

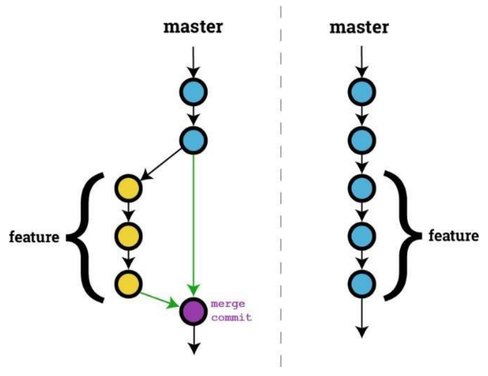
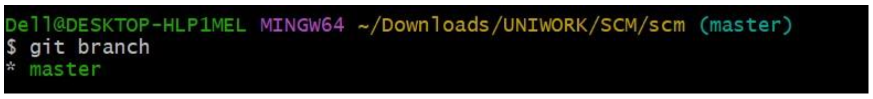


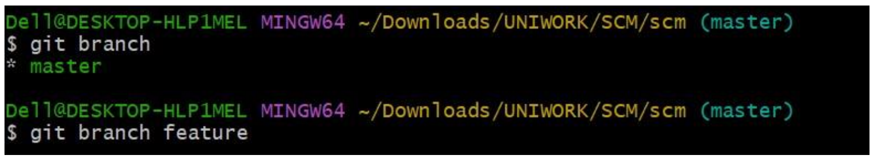
Fig-4.1

Snapshots –



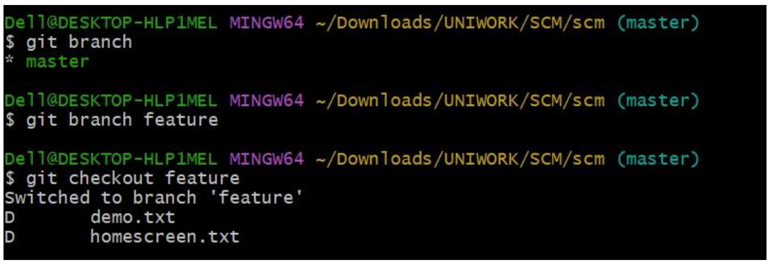
```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
* master
```

Fig-4.2 Default branch is master branch



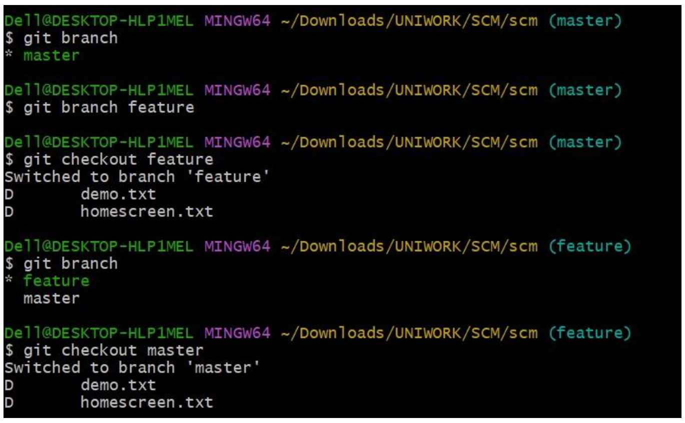
```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
* master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch feature
```

Fig-4.3 Adding a feature branch



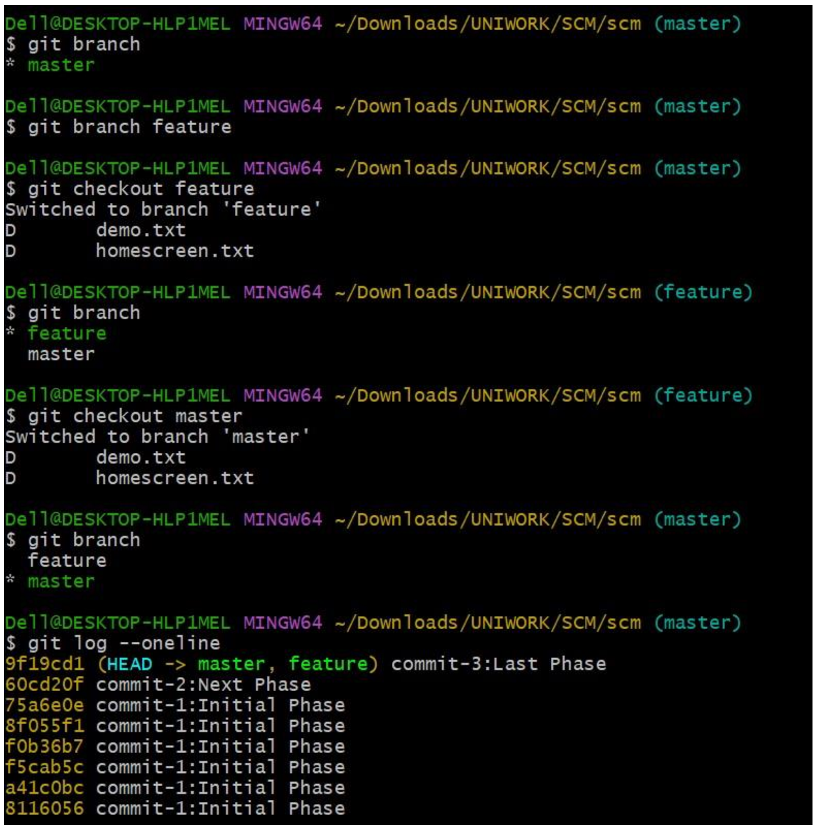
```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
* master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch feature

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git checkout feature
Switched to branch 'feature'
D       demo.txt
D       homescreen.txt
```

Fig-4.4 Switching to feature branch



```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
* master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch feature

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git checkout feature
Switched to branch 'feature'
D       demo.txt
D       homescreen.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (feature)
$ git branch
* feature
  master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (feature)
$ git checkout master
Switched to branch 'master'
D       demo.txt
D       homescreen.txt
```

Fig-4.5 Switching to master branch



```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
* master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch feature

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git checkout feature
Switched to branch 'feature'
D       demo.txt
D       homescreen.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (feature)
$ git branch
* feature
  master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (feature)
$ git checkout master
Switched to branch 'master'
D       demo.txt
D       homescreen.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git branch
  feature
* master

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git log --oneline
9f19cd1 (HEAD -> master, feature) commit-3:Last Phase
60cd20f commit-2:Next Phase
75a6e0e commit-1:Initial Phase
8f055f1 commit-1:Initial Phase
f0b36b7 commit-1:Initial Phase
f5cab5c commit-1:Initial Phase
a41c0bc commit-1:Initial Phase
8116056 commit-1:Initial Phase
```

Fig-4.6 Checking commits

EXPERIMENT 5

Aim: Git lifecycle description

Theory:

Stages in GIT Life Cycle: Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

- Working directory
- Staging area
- Git directory

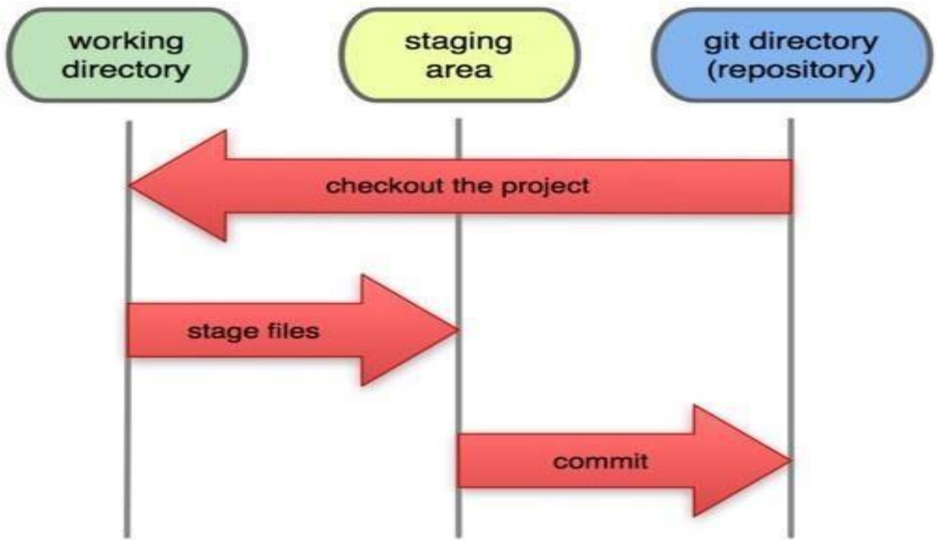


Fig-5.1

Working Directory:
Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

Staging Area:
Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.
Git Directory:
Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what the

commit is about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

Remote Repository: It means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.

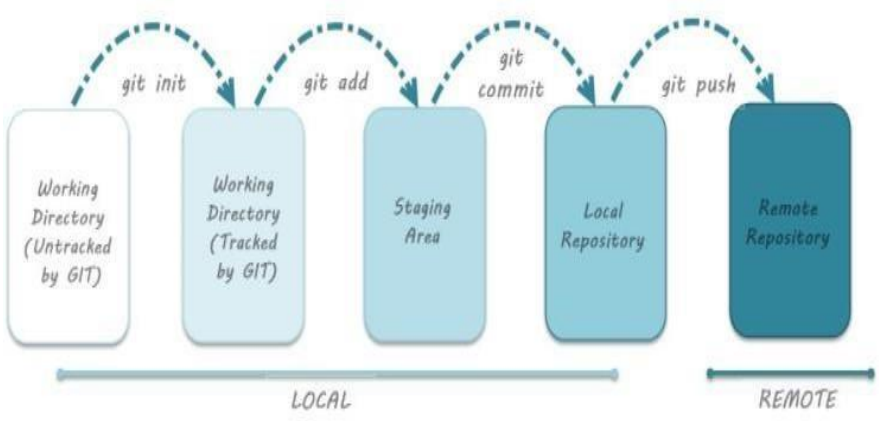


Fig-5.2

Algorithm:

1. Initialize the Git repository:
   - **git init**

2. Create a new file:
   - **touch filename**

3. Check the status of the repository:
   - **git status**

4. Stage all files with the .txt extension:
   - **git add *.txt**

5. Check the status again to see staged changes:
   - **git status**

6. Add further new files.

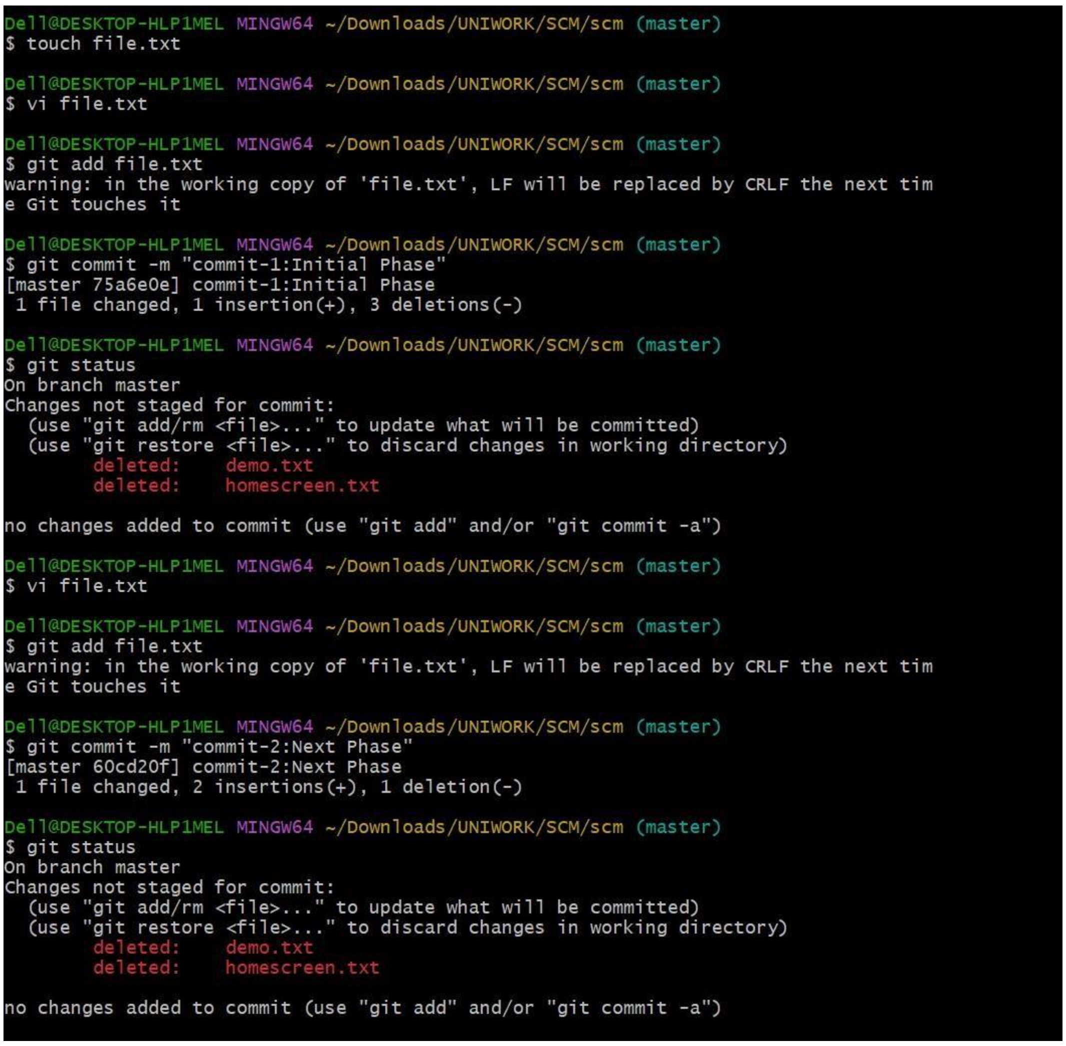
7. Check the status to see untracked or modified files:
   - **git status**

8. Stage all changes (including new files):
   - **git add -A**

9. Check the status to verify staged changes:
   - **git status**

10. Remove a file from the staging area (but keep it in the working directory): - **git rm --cached <file name>**

11. Commit the staged changes with a comment: - **git commit -m "comment"**



```
Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ touch file.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ vi file.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next tim
e Git touches it

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git commit -m "commit-1:Initial Phase"
[master 75a6e0e] commit-1:Initial Phase
 1 file changed, 1 insertion(+), 3 deletions(-)

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    demo.txt
        deleted:    homescreen.txt

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

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ vi file.txt

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next tim
e Git touches it

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git commit -m "commit-2:Next Phase"
[master 60cd20f] commit-2:Next Phase
 1 file changed, 2 insertions(+), 1 deletion(-)

Dell@DESKTOP-HLP1MEL MINGW64 ~/Downloads/UNIWORK/SCM/scm (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    demo.txt
        deleted:    homescreen.txt

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

Fig-5.3