

## Experiment No 3

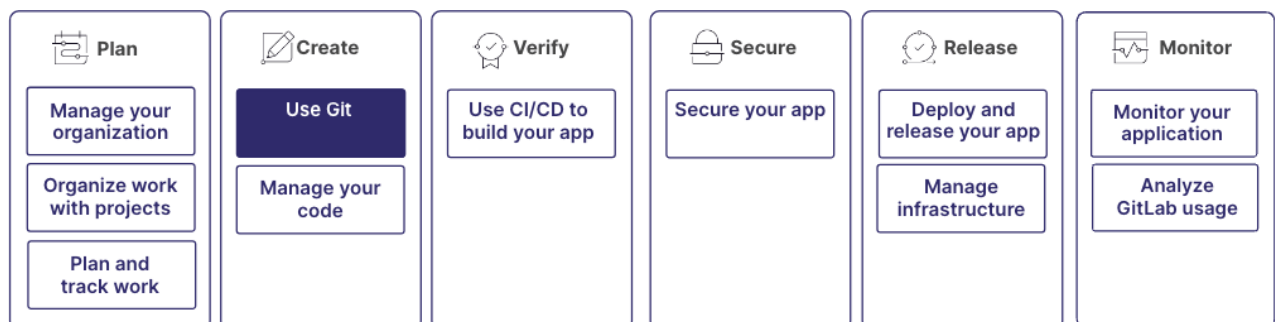
**Aim:** To perform various Git operations on local and remote repositories using GIT cheat-sheet.

**Theory:**

Git is a version control system you use to track changes to your code and collaborate with others. GitLab is a web-based Git repository manager that provides CI/CD and other features to help you manage your software development lifecycle.

You can use GitLab without knowing Git. However, it is advantageous to understand Git when you use GitLab for source control.

Learning Git is part of a larger workflow:



### Repositories

A Git repository is a directory that contains all the files, folders, and version history of your project. It serves as a central hub where Git manages and tracks changes to your code.

When you initialize a Git repository or clone an existing one, Git creates a hidden directory, `.git`, inside the project directory. The directory contains all the essential metadata and objects Git uses to manage your repository, including the complete history of all changes made to the files. Git tracks changes at the file level, so you can view the modifications made to individual files over time.

For more information, see [Repositories](#).

### Working directories

Your working directory is where you make changes to your code. When you clone a Git repository, you create a local copy of the repository in your working directory. You can edit files, add new ones, and test your code. To collaborate, you can:

- **Commit:** After you make changes in your working directory, commit those changes to your local repository.
- **Push:** Push your changes to a remote Git repository hosted on GitLab. This makes your changes available to other team members.
- **Pull:** Pull changes made by others from the remote repository, and ensure that your local repository is updated with the latest changes.

For more information, see [Common Git commands](#).

### Branches

In Git, you can use branches to work on different features, bug fixes, or experiments simultaneously without interfering with each other's work. Branching enables you to create an isolated environment where you can make and test changes without affecting the default branch. In GitLab, the default branch is usually called main.

### **Merge a branch**

After a feature is complete or a bug is fixed, you can merge your branch into the default branch. You can do this in a Merge request. Merging is a safe way to bring changes from one branch into another while preserving the history of the changes.

If there are conflicts between the branches, for example, if you modify the same lines of code in both branches, GitLab flags these as merge conflicts. These must be resolved manually by reviewing and editing the code.

### **Delete a branch**

After a successful merge, you can delete the branch if it is no longer needed. Deleting unnecessary branches helps keep your repository organized and manageable.

To ensure no work is lost, verify all changes are incorporated into the default branch before you delete the branch after the final merge.

For more information, see Branches.

### **Understand the Git workflow**

You can manage your code, collaborate with others, and keep your project organized with a Git workflow. A standard Git workflow includes the following steps:

1. **Clone a repository:** Create a local copy of the repository by cloning it to your machine. You can work on the project without affecting the original repository.
2. **Create a new branch:** Before you make any changes, it's recommended to create a new branch. This ensures that your changes are isolated and don't interfere with the work of others on the default branch.
3. **Make changes:** Make changes to files in your working directory. You can add new features, fix bugs, or make other modifications.
4. **Stage changes:** After you make changes to your files, stage the changes you want to commit. Staging tells Git which changes should be included in the next commit.
5. **Commit changes:** Commit your staged changes to your local repository. A commit saves a snapshot of your work and creates a history of the changes to your files.
6. **Push changes:** To share your changes with others, push them to the remote repository. This makes your changes available to other collaborators.
7. **Merge your branch:** After your changes are reviewed and approved, merge your branch into the default branch. For example, main. This step incorporates your changes into the project.

### **Forks**

Some organizations, particularly those working with open-source projects, may use different workflows. For example, Forks.

A fork is a personal copy of the repository that exists in your own namespace. Use this workflow when contributing to open-source projects or when your team uses a centralized repository.

## Install Git

To use Git commands and contribute to GitLab projects, you should download and install the Git client on your computer.

The installation process varies depending on your operating system. For example, Windows, MacOS, or Linux. For information on how to install Git, see [Install Git](#).

## Git commands

To interact with Git from the command line, you can use Git commands:

- `git clone`: Clone a repository to your local machine.
- `git branch`: List, create, or delete branches in your local repository.
- `git checkout`: Switch between different branches in your local repository.
- `git add`: Stage changes for commit.
- `git commit`: Commit staged changes to your local repository.
- `git push`: Push local commits to the remote repository.
- `git pull`: Fetch changes from the remote repository and merge them into your local branch.

For more comprehensive information and detailed explanations, see [Command Git commands guide](#).

Output:

```
MINGW64/c/Users/15L/git-dvcs
15L@203-12 MINGW64 ~ (main)
$ cd git-dvcs

15L@203-12 MINGW64 ~/git-dvcs (master)
$ git config --global --list
color.ui=true
user.email=dhirajshevkhani23@gmail.com
user.name=Dhiraj Shevkani

15L@203-12 MINGW64 ~/git-dvcs (master)
$ git config --global user.name "janhavi"

15L@203-12 MINGW64 ~/git-dvcs (master)
$ git config --global user.email "janhavig20042gmail.com "

15L@203-12 MINGW64 ~/git-dvcs (master)
$ git config --global --list
color.ui=true
user.email=janhavig20042gmail.com
user.name=janhavi

15L@203-12 MINGW64 ~/git-dvcs (master)
$
```

```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ ls -a
./ ../ .git/

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ ls -al
total 4
drwxr-xr-x 1 15L 197121 0 Feb 12 10:47 ./
drwxr-xr-x 1 15L 197121 0 Feb 12 10:47 ../
drwxr-xr-x 1 15L 197121 0 Feb 12 10:47 .git/
```

```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git add .

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   img1.png

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git commit -m "First message"
[main (root-commit) c854a40] First message
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 img1.png

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$
```

```
15L@203-12 MINGW64 ~/git-dvcs (master)
$ git config --global --list
color.ui=true
user.email=janhavig2004@gmail.com
user.name=janhavi

15L@203-12 MINGW64 ~/git-dvcs (master)
$

15L@203-12 MINGW64 ~/git-dvcs (master)
$ cat ~/.gitconfig
[color]
    ui = true
[user]
    email = janhavig2004@gmail.com
    name = janhavi

15L@203-12 MINGW64 ~/git-dvcs (master)
$
```

```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html.txt

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

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git add .

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git commit -m "HI"
[main e39fb3e] HI
1 file changed, 1 insertion(+)
create mode 100644 index.html.txt

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
nothing to commit, working tree clean
```

MINGW64:/c/Users/15L/git-dvcs/git-demo-project1

```
15L@203-12 MINGW64 ~/git-dvcs (master)
$ cd git-demo-project1

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ ls
img1.png  index.html.txt  teststatus

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        teststatus

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

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$
```

```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html.txt

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

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git add .

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git commit -m "HI"
[main e39fb3e] HI
1 file changed, 1 insertion(+)
create mode 100644 index.html.txt

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
nothing to commit, working tree clean
```



```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git add index.html
fatal: pathspec 'index.html' did not match any files

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    teststatus

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

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git add teststatus

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   teststatus

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   teststatus

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git commit -am "express commit"
[main 23b1bb6] express commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 teststatus

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git status
On branch main
nothing to commit, working tree clean

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git log
commit 23b1bb63d20583794dccb1953b26fa19d1f664a2 (HEAD -> main)
Author: janhavi <janhavig2004@gmail.com>
Date:   Wed Feb 12 11:26:30 2025 +0530

    express commit

commit e39fb3e06df8562bdb6e6be11a8366074492fe0
Author: janhavi <janhavig2004@gmail.com>
Date:   Wed Feb 12 11:12:05 2025 +0530

    HI

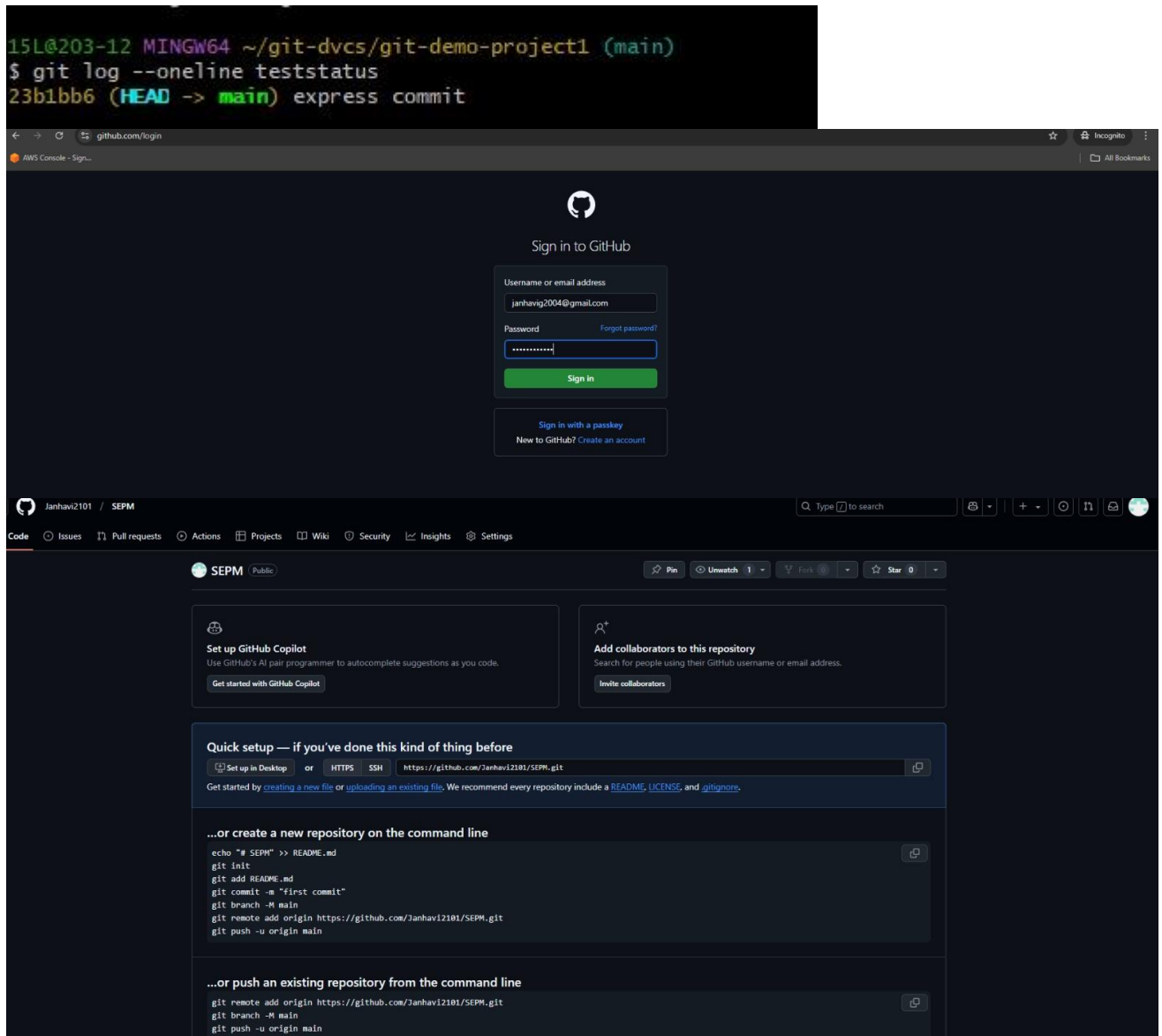
commit c854a404e269b4ffa65eda04823b4e4ae60e6536
Author: janhavi <janhavig2004@gmail.com>
Date:   Wed Feb 12 10:56:23 2025 +0530

    First message

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)

15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git log --oneline
23b1bb6 (HEAD -> main) express commit
e39fb3e HI
c854a40 First message
```

```
15L@203-12 MINGW64 ~/git-dvcs/git-demo-project1 (main)
$ git log --oneline teststatus
23b1bb6 (HEAD -> main) express commit
```



The image shows a terminal window and a web browser. The terminal window displays the output of a `git log --oneline teststatus` command, showing a commit hash `23b1bb6` for the `main` branch. The web browser shows the GitHub login page, where the user `janhavi2101` is logged in. Below the login page, the repository page for `janhavi2101 / SEPM` is visible. The repository page includes a search bar, a navigation menu, and a list of repository settings. The repository is public and has 1 unwatched repository. The repository page also includes a section for quick setup, which provides instructions for setting up the repository on the command line or pushing an existing repository from the command line.

Sign in to GitHub

Username or email address  
janhavi2101@gmail.com

Password  
[REDACTED]

Sign in

Sign in with a passkey  
New to GitHub? Create an account

janhavi2101 / SEPM

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

SEPM Public

Pin Unwatch 1 Fork Star 0

Set up GitHub Copilot  
Use GitHub's AI pair programmer to autocomplete suggestions as you code.  
Get started with GitHub Copilot

Add collaborators to this repository  
Search for people using their GitHub username or email address.  
Invite collaborators

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH <https://github.com/janhavi2101/SEPM.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# SEPM" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/janhavi2101/SEPM.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/janhavi2101/SEPM.git
git branch -M main
git push -u origin main
```

Janhavi2101 / SEPM

Q

+

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

main

SEPM /

Go to file

Add file

Janhavi2101

Create README.md

f4d1a29 · now

Name	Last commit message	Last commit date
README.md	Create README.md	now
img1.png	changed	1 minute ago
index.html.txt	HI	2 minutes ago
teststatus	Express commit	2 minutes ago

README.md

## SEM 6 Software Engineering and Project Management