Commit Message Guidelines and Git Practice Report

Objective

The objective of this session is to build practical proficiency in Git and GitHub workflows, including repository creation, staging and committing changes, working with branches, performing merges, resolving merge conflicts, configuring GitHub, setting up SSH authentication, and pushing repositories securely using SSH.
This report also documents standard commit message practices using a structured convention such as `feat`, `fix`, and `refactor` to maintain high-quality project history.

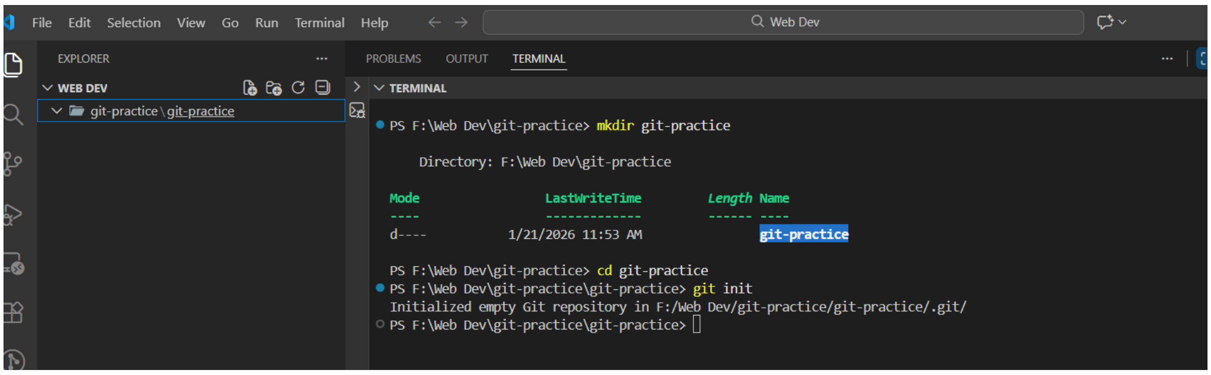
Exercise 1: Initialize a Repository

Goal: Create a new Git repository locally.

```
mkdir git-practice cd git-practice git init
```

Verify:
```
git status
```

Result: The repository was initialized successfully, and Git started tracking the folder with a `.git` directory.



```
PS F:\Web Dev\git-practice> mkdir git-practice

    Directory: F:\Web Dev\git-practice

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----           1/21/2026 11:53 AM                git-practice

PS F:\Web Dev\git-practice> cd git-practice
PS F:\Web Dev\git-practice\git-practice> git init
Initialized empty Git repository in F:/Web Dev/git-practice/git-practice/.git/
PS F:\Web Dev\git-practice\git-practice>
```

Exercise 2: Create a File, Add, and Commit

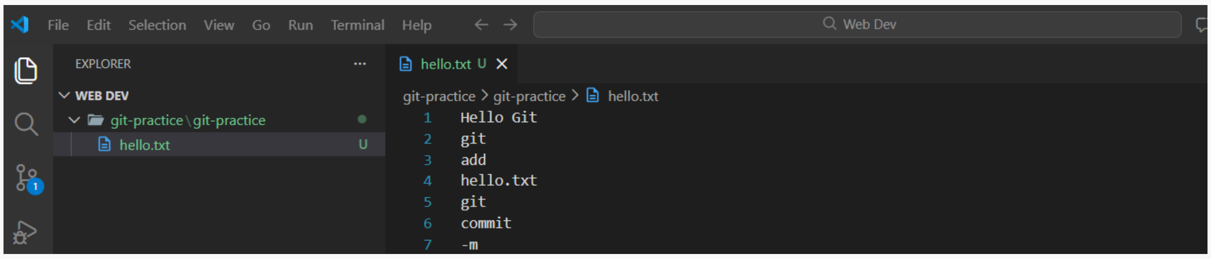
Goal: Track a file and commit it.

```
echo "Hello Git" > hello.txt git add hello.txt git commit -m "feat: add
hello.txt with greeting"
```

Verify:

```
git log --oneline
```

Result: The `hello.txt` file was added and committed successfully.

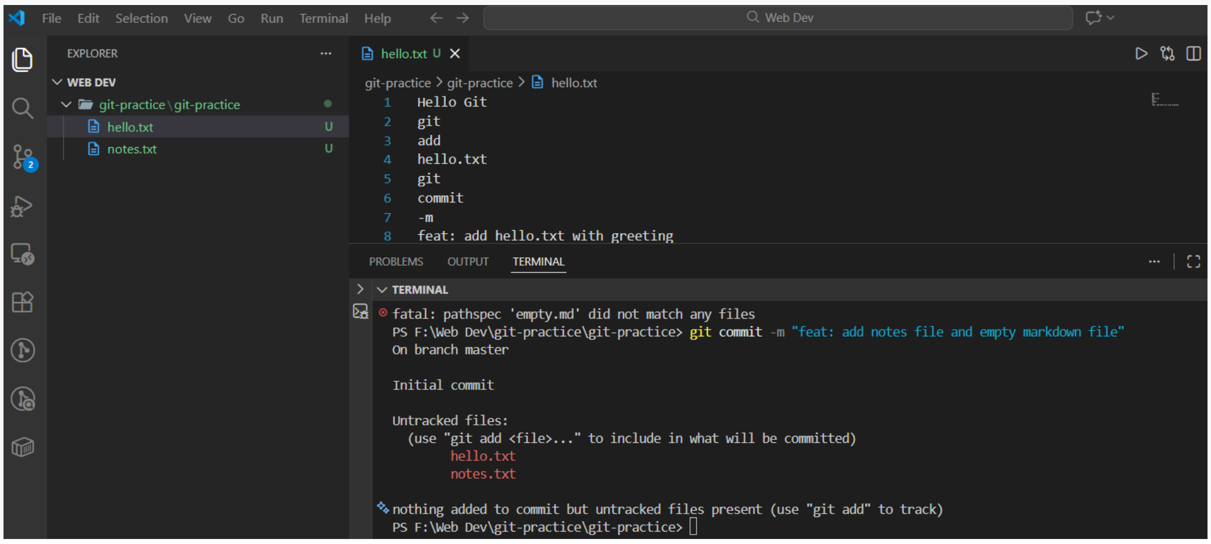

File Edit Selection View Go Run Terminal Help
Web Dev
EXPLORER
WEB DEV
git-practice\git-practice
hello.txt U
git-practice > git-practice > hello.txt

```
Hello Git
git
add
hello.txt
git
commit
-m
```

Exercise 3: Stage Multiple Files

Goal: Add multiple files together.

```
echo "line 1" > notes.txt echo "line 2" >> notes.txt touch empty.md git add
notes.txt empty.md git commit -m "feat: add notes and empty markdown file"
```


File Edit Selection View Go Run Terminal Help
Web Dev
EXPLORER
WEB DEV
git-practice\git-practice
hello.txt U
notes.txt U
git-practice > git-practice > hello.txt
PROBLEMS OUTPUT TERMINAL
TERMINAL

```
Hello Git
git
add
hello.txt
git
commit
-m
feat: add hello.txt with greeting
```

```
fatal: pathspec 'empty.md' did not match any files
PS F:\Web Dev\git-practice\git-practice> git commit -m "feat: add notes file and empty markdown file"
On branch master

Initial commit

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

nothing added to commit but untracked files present (use "git add" to track)
PS F:\Web Dev\git-practice\git-practice>
```

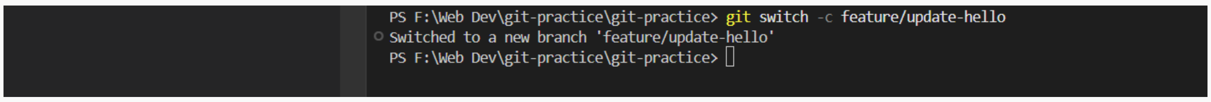
Exercise 4: Create a Branch and Switch to It

Goal: Create a feature branch.

```
git branch feature/update-hello git checkout feature/update-hello
```

or

```
git switch -c feature/update-hello
```

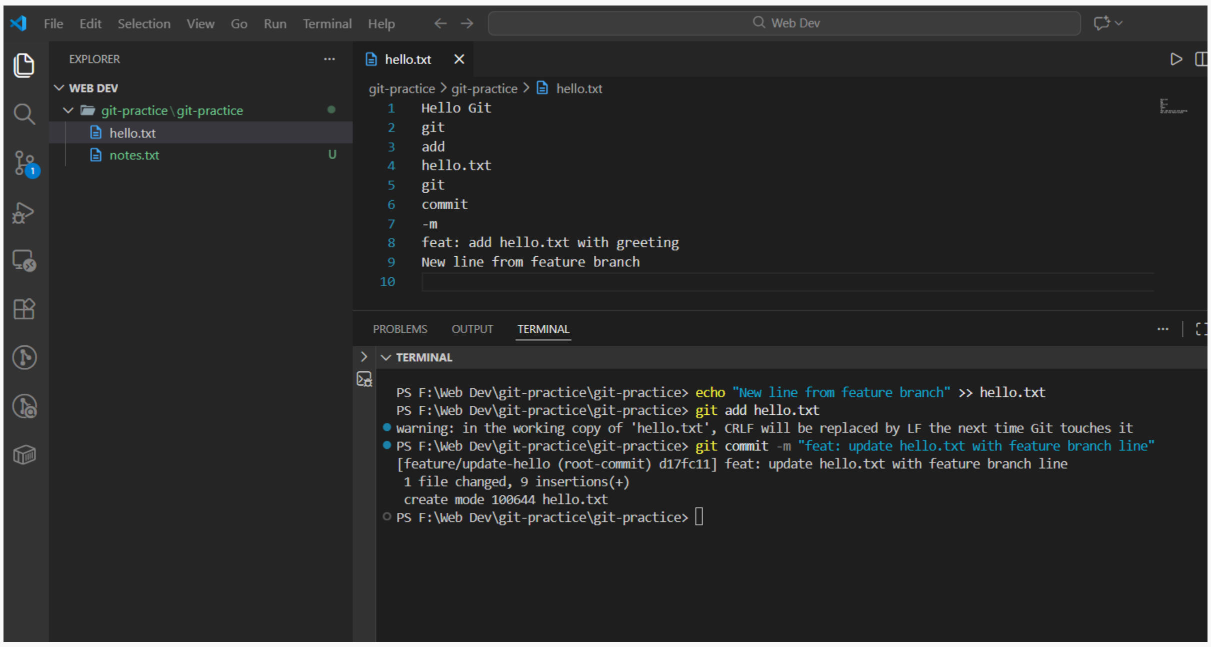


```
PS F:\Web Dev\git-practice\git-practice> git switch -c feature/update-hello
Switched to a new branch 'feature/update-hello'
PS F:\Web Dev\git-practice\git-practice>
```

Exercise 5: Make Changes on a Branch and Commit

Goal: Commit work safely on a branch.

```
echo "New line from feature branch" >> hello.txt git add hello.txt git
commit -m "feat: update hello.txt with feature branch line"
```



```
Hello Git
git
add
hello.txt
git
commit
-m
feat: add hello.txt with greeting
New line from feature branch
```

```
PS F:\Web Dev\git-practice\git-practice> echo "New line from feature branch" >> hello.txt
PS F:\Web Dev\git-practice\git-practice> git add hello.txt
warning: in the working copy of 'hello.txt', CRLF will be replaced by LF the next time Git touches it
PS F:\Web Dev\git-practice\git-practice> git commit -m "feat: update hello.txt with feature branch line"
[feature/update-hello (root-commit) d17fc11] feat: update hello.txt with feature branch line
 1 file changed, 9 insertions(+)
 create mode 100644 hello.txt
PS F:\Web Dev\git-practice\git-practice> 
```

Exercise 6: Merge Branch into Main

Goal: Merge completed feature into `main`.

```
git checkout main git merge feature/update-hello
```

Exercise 7: Create a Merge Conflict (Intentional)

Goal: Learn conflict creation.

1. Create and switch to branch A:

   ```
   git switch -c feature/conflict-a
   ```

   Edit the same line in `hello.txt`:

   ```
   echo "Conflict from branch A" > hello.txt git add hello.txt git commit -m
   "feat: branch A modifies hello.txt"
   ```
2. Switch back to main, create branch B:

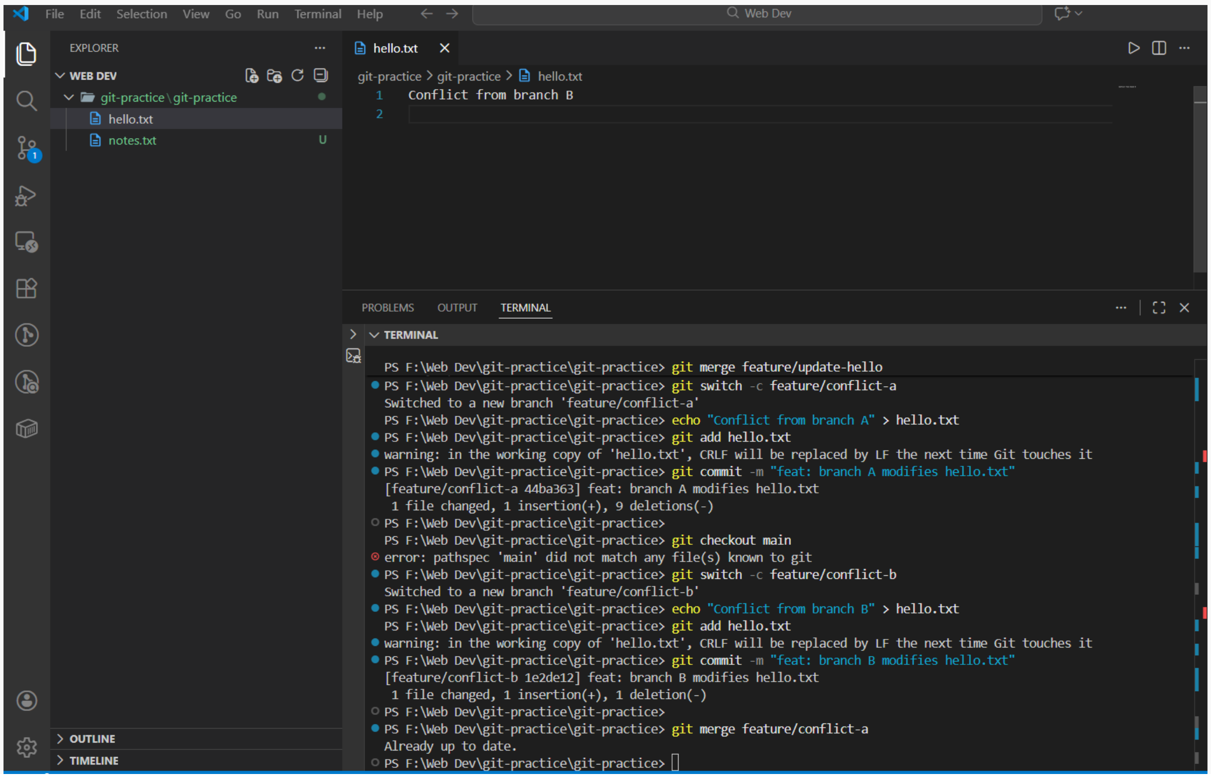
   ```
   git checkout main git switch -c feature/conflict-b
   ```

Modify the same file differently:

```
echo "Conflict from branch B" > hello.txt git add hello.txt git commit -m
"feat: branch B modifies hello.txt"
```

3. Merge branch A into branch B:

```
git merge feature/conflict-a
```



```
hello.txt
1  Conflict from branch B
2
```

```
PS F:\Web Dev\git-practice\git-practice> git merge feature/update-hello
PS F:\Web Dev\git-practice\git-practice> git switch -c feature/conflict-a
Switched to a new branch 'feature/conflict-a'
PS F:\Web Dev\git-practice\git-practice> echo "Conflict from branch A" > hello.txt
PS F:\Web Dev\git-practice\git-practice> git add hello.txt
warning: in the working copy of 'hello.txt', CRLF will be replaced by LF the next time Git touches it
PS F:\Web Dev\git-practice\git-practice> git commit -m "feat: branch A modifies hello.txt"
[feature/conflict-a 44ba363] feat: branch A modifies hello.txt
 1 file changed, 1 insertion(+), 9 deletions(-)
PS F:\Web Dev\git-practice\git-practice>
PS F:\Web Dev\git-practice\git-practice> git checkout main
error: pathspec 'main' did not match any file(s) known to git
PS F:\Web Dev\git-practice\git-practice> git switch -c feature/conflict-b
Switched to a new branch 'feature/conflict-b'
PS F:\Web Dev\git-practice\git-practice> echo "Conflict from branch B" > hello.txt
PS F:\Web Dev\git-practice\git-practice> git add hello.txt
warning: in the working copy of 'hello.txt', CRLF will be replaced by LF the next time Git touches it
PS F:\Web Dev\git-practice\git-practice> git commit -m "feat: branch B modifies hello.txt"
[feature/conflict-b 1e2de12] feat: branch B modifies hello.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
PS F:\Web Dev\git-practice\git-practice>
PS F:\Web Dev\git-practice\git-practice> git merge feature/conflict-a
Already up to date.
PS F:\Web Dev\git-practice\git-practice>
```

Exercise 8: Resolve Conflict and Commit

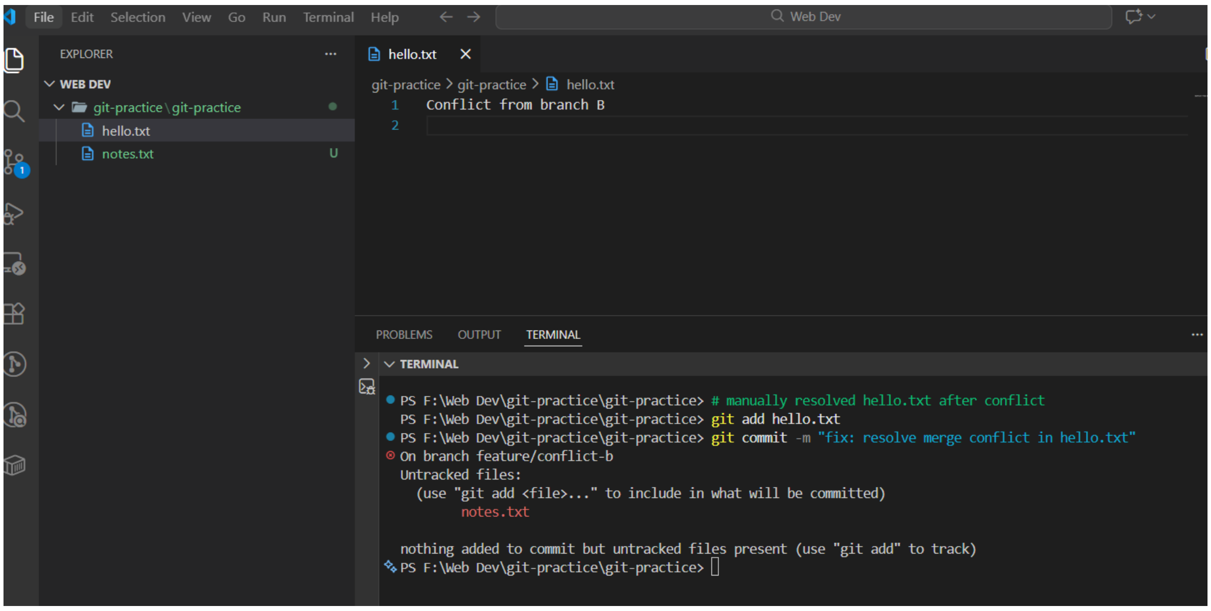
Goal: Fix conflict properly.

Open `hello.txt`, you will see conflict markers like:

```
<<<<<<< HEAD Conflict from branch B ======= Conflict from branch A >>>>>>>
feature/conflict-a
```

Now commit:

```
git add hello.txt git commit -m "fix: resolve merge conflict in hello.txt"
```


File Edit Selection View Go Run Terminal Help
Web Dev
EXPLORER
WEB DEV
git-practice \ git-practice
hello.txt
notes.txt
hello.txt
git-practice > git-practice > hello.txt
PROBLEMS OUTPUT TERMINAL
TERMINAL

```
Conflict from branch B
```

```
PS F:\Web Dev\git-practice\git-practice> # manually resolved hello.txt after conflict
PS F:\Web Dev\git-practice\git-practice> git add hello.txt
PS F:\Web Dev\git-practice\git-practice> git commit -m "fix: resolve merge conflict in hello.txt"
On branch feature/conflict-b
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        notes.txt

nothing added to commit but untracked files present (use "git add" to track)
PS F:\Web Dev\git-practice\git-practice>
```

Exercise 9: Connect to GitHub Remote

Goal: Add remote repository URL.

Check remotes:

```
git remote -v
```

Add remote (SSH example):

```
git remote add origin git@github.com:<your-username>/git-practice.git
```

Verify:

```
git remote -v
```

Exercise 10: Push to GitHub (SSH)

Goal: Upload local commits to GitHub.

```
git push -u origin main
```

3. GitHub Setup

3.1 GitHub Profile Setup

Steps completed:

1. Created a GitHub account.
2. Added a profile photo for professional identity.
3. Added a short bio describing academic and technical interests.

This setup improves credibility and makes the profile suitable for collaboration and portfolio visibility.

3.2 SSH Key Setup

SSH authentication was configured to allow secure GitHub access without repeatedly entering credentials.

Step 1: Generate SSH Key

```
ssh-keygen -t ed25519 -C "aabid@gmail.com"
```

Step 2: Start SSH Agent and Add Key

```
eval "$(ssh-agent -s)" ssh-add ~/.ssh/id_ed25519
```

Step 3: Add Public Key to GitHub

```

cat ~/.ssh/id_ed25519.pub
```

The public key output was copied and added to GitHub under:
Settings → SSH and GPG Keys → New SSH key

Step 4: Test SSH Connection

```
ssh -T git@github.com
```

Result: SSH authentication succeeded successfully

3.3 Push Repository Using SSH

The repository remote was confirmed to be SSH-based:

```
git remote set-url origin git@github.com:Aabid/gitprac.git git push -u
origin main
```

This ensures secure pushing and pulling using SSH.

4. Commit Message Guidelines

4.1 Importance of Commit Messages

Commit messages are critical for software engineering because they provide:

- Traceability of changes over time
- A clear explanation of what was changed and why
- Easier debugging and rollback
- Better collaboration in teams
- Cleaner project history for code reviews

Poor commit messages reduce maintainability and create confusion in long-term development.

4.2 Best Practices for Commit Messages

Guideline 1: Keep commits small and focused

Each commit should represent one logical change.

Good:

- Add password validation logic
- Fix input parsing bug

Bad:

- "updated everything"
- "final changes"

Guideline 2: Use imperative tone

Write commit messages as commands:

✓ `add validation for password input`
✗ `added validation for password input`

Guideline 3: Be specific and descriptive

A good commit message should clearly describe what changed.

Example:
✓ `fix: prevent crash when input is empty`
Instead of:
✗ `fix bug`

Guideline 4: Follow a structured convention

A standard convention improves readability and automation. The format used is:

```
type: short description
```

Optionally:

```
type(scope): short description
```

4.3 Commit Types and Examples

`feat:` *(Feature Addition)*

Used when introducing new functionality.

Examples:

- `feat: add CLI password strength checker`
- `feat: implement password scoring rules`
- `feat(cli): accept user input from terminal`
`fix:` *(Bug Fix)*

Used when correcting incorrect behavior or errors.

Examples:

- `fix: handle empty password input safely`
- `fix: correct regex for special characters`

- `fix: resolve merge conflict in hello.txt`
  `refactor:` *(Code Refactoring)*

  Used when improving code structure without changing behavior.

  Examples:

- `refactor: extract scoring logic into separate function`
- `refactor: rename variables for clarity`
- `refactor: reduce duplicated validation logic`

4.4 Examples of Bad Commit Messages (Avoid)

These commit messages are low quality and should not be used:

- `update`
- `done`
- `final`
- `changes`
- `fix stuff`

They provide no technical meaning and make the commit history unusable.

5. Conclusion

This session successfully covered essential Git workflows including repository initialization, staging and committing changes, branching, merging, intentional conflict creation, conflict resolution, remote configuration, and pushing changes to GitHub through SSH authentication.
Additionally, professional commit message practices were documented using structured conventions like `feat`, `fix`, and `refactor`, improving both collaboration and code maintainability.