PRACTICAL GUIDE TO GIT

---

1. Introduction

This guide provides a complete, step-by-step practical reference for learning and using Git. It is designed for students who will perform Git tasks in a controlled assessment environment. The guide covers all essential operations, including repository creation, staging, committing, branching, merging, conflict resolution, and remote repository operations.

The goal is to provide a clear, systematic foundation on which students can confidently perform any required Git tasks.

---

2. Understanding Git: Core Principles

2.1 Version Control System

Git is a distributed version control system used to manage and track changes in files. It allows users to maintain a history of modifications, collaborate with others, and restore previous versions of work.

2.2 Distributed Architecture

Unlike centralized systems, Git stores a complete copy of the repository (history included) on every user's machine. This ensures fast operations and enables work without an internet connection.

2.3 Fundamental Components

Git manages files across three distinct areas:

Working Directory

The actual files and folders you are working on.

Staging Area (Index)

A temporary area where selected changes are prepared before committing.

Local Repository

The stored history of commits, maintained inside the hidden `.git` folder.

The workflow is:

Working Directory → Staging Area → Local Repository
(edit files) → (git add) → (git commit)

---

3. Setting Up a Git Repository

3.1 Creating a Project Directory

```
mkdir project_name
cd project_name
```

3.2 Initializing a New Repository

```
git init
```

This creates the `.git` directory that contains all necessary repository data.

3.3 Viewing Repository Status

```
git status
```

Displays:

- current branch
- untracked files
- modified files

- staged files

Regular use of this command is essential for avoiding mistakes.

---

4. Working with Files

4.1 Creating a File

Use any preferred text editor to create files such as:

```
info.txt
```

4.2 Adding Files to Staging

```
git add info.txt
```

or to add all modified files:

```
git add .
```

4.3 Committing Staged Files

```
git commit -m "Describe the change made"
```

A commit records a snapshot of the staged changes.

4.4 Viewing Commit History

Full history:

```
git log
```

Compressed view:

```
git log --oneline
```

4.5 Viewing Specific Commit Details

```
git show <commit_id>
```

---

5. Branching

5.1 Purpose of Branches

Branches allow independent development without affecting the main project. They are used for:

- adding features
- testing ideas
- fixing bugs
- working in parallel with teammates

5.2 Listing Branches

```
git branch
```

5.3 Creating a Branch

```
git branch new_branch
```

5.4 Switching Branches

```
git checkout new_branch
```

5.5 Creating and Switching in One Command

```
git checkout -b update-info
```

---

6. Modifying Files Across Branches

When different branches contain different versions of a file, Git isolates these changes until they are merged.
This is the core concept required for understanding merge conflicts.

Example scenario:

- main branch contains original version of info.txt
- update-info branch contains updated version of line 2

Both versions coexist separately until merged.

---

7. Merging Branches

7.1 Switching to the Destination Branch

Before merging, switch to the branch that should receive changes:

```
git checkout main
```

7.2 Performing the Merge

```
git merge update-info
```

7.3 Types of Merge Outcomes

Fast-Forward Merge

Occurs when the target branch (main) has not made new commits.

Three-Way Merge

Occurs when both branches have diverged but do not conflict.

Merge Conflict

Occurs when both branches changed the same lines.

---

8. Merge Conflicts

8.1 Why Conflicts Occur

A conflict occurs when Git cannot determine which version of a line should be kept because:

- both branches modified the same line, or
- both branches modified overlapping areas of a file

8.2 Conflict Markers

Git marks conflicts using:

```
<<<<<<< HEAD
(Main branch content)
=======
(Merging branch content)
>>>>>>> branch_name
```

8.3 Example of Conflict in a File

```
Line 1: Example text
<<<<<<< HEAD
Line 2: Change from main branch
=======
Line 2: Change from update-info branch
>>>>>>> update-info
```

```
Line 3: Final text
```

---

9. Resolving Merge Conflicts

9.1 Conflict Resolution Steps

1. Open the file showing conflict markers.
2. Review both conflicting versions.
3. Choose one version or create a combined version as required.
4. Remove all conflict markers.
5. Save the file.

Stage the resolved file:

```
git add info.txt
```

6.

Commit the resolution:

```
git commit -m "Resolve merge conflict"
```

7.

9.2 Guidelines for Conflict Resolution

- Do not leave any conflict markers in the file.
- Ensure the final content is correct and complete.
- Maintain consistency with formatting and file structure.

---

10. Remote Repositories

Remote repositories allow you to store your project online and collaborate with others.

10.1 Adding a Remote Repository

```
git remote add origin <URL>
```

10.2 Verifying Remote Configuration

```
git remote -v
```

10.3 Pushing to the Remote Repository

Initial push:

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

Subsequent pushes:

```
git push
```

10.4 Pulling Updates from Remote

```
git pull
```