

Case-Study's

1. Basic Git Workflow

- **Objective:** Learn the basics of Git, including initializing a repository, adding files, making commits, and pushing to a remote repository.
- **Scenario:** Create a simple project (e.g., a "Hello World" program in any language). Initialize a Git repository, create a few commits as you make changes to the project, and push the changes to GitHub.
- connect local git repository to remote github repository

```
C:\Users\Admin\Desktop\Git>git remote add origin git@github.com:Kavya-Pasula/Git.git
```

- Created a simple project file sample.py
- Opened command prompt and followed the steps below.

1. initializing a repository

```
C:\Users\Admin\Desktop\Git>git init  
Initialized empty Git repository in C:/Users/Admin/Desktop/Git/.git/
```

2. adding files

```
C:\Users\Admin\Desktop\Git>git add sample.py
```

3. commits

```
C:\Users\Admin\Desktop\Git>git commit sample.py  
[master (root-commit) eccfa5e] print("Hello World!")  
1 file changed, 1 insertion(+)  
create mode 100644 sample.py
```

4. pushing to a remote repository

```
C:\Users\Admin\Desktop\Git>git push --set-upstream origin GitUseCase  
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 255 bytes | 28.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)  
remote:  
remote: Create a pull request for 'GitUseCase' on GitHub by visiting:  
remote:      https://github.com/Kavya-Pasula/Git/pull/new/GitUseCase  
remote:  
To github.com:Kavya-Pasula/Git.git  
* [new branch]      GitUseCase -> GitUseCase  
branch 'GitUseCase' set up to track 'origin/GitUseCase'.  
  
C:\Users\Admin\Desktop\Git>
```

2. Branching and Merging

- **Objective:** Understand how to create branches, switch between branches, and merge branches.
- **Scenario:** Create a project with a main branch. Add a new feature on a separate branch and merge it back into the main branch. Resolve any merge conflicts that arise.

1. To create a New Branch

```
C:\Users\Admin\Desktop\Git>git checkout -b NewFeature
Switched to a new branch 'NewFeature'
```

2. Merge two branch

```
C:\Users\Admin\Desktop\Git>git merge NewFeature
Merge made by the 'ort' strategy.
 Addfeature.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Addfeature.txt

C:\Users\Admin\Desktop\Git>git push --set-upstream origin GitUseCase
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 363 bytes | 51.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Kavya-Pasula/Git.git
 ae2d464..df6a74d  GitUseCase -> GitUseCase
branch 'GitUseCase' set up to track 'origin/GitUseCase'.

C:\Users\Admin\Desktop\Git>
```

3. Collaborative Workflow

- **Objective:** Learn how to collaborate on projects using Git, including cloning repositories, creating pull requests, and code reviews.
- **Scenario:** Simulate a team environment where you clone a repository, create a branch to make some changes, push the changes, and open a pull request. Have someone else review and merge the pull request.


1. Navigate to the original repository you want to clone on GitHub. Click the "Fork" button. It will create a copy of the repository under your GitHub account.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Required fields are marked with an asterisk (*).

Owner *

 Kavya-Pasula

Repository name *

Git-Doc


Git is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

☒ Copy the **master** branch only

Contribute back to Kavya-Pasula/Git by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

Create fork

- clone a repository :- clone your fork repository

```
C:\Users\Admin\Desktop\Git>git clone git@github.com:Kavya-Pasula/Git-Doc.git
Cloning into 'Git-Doc'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

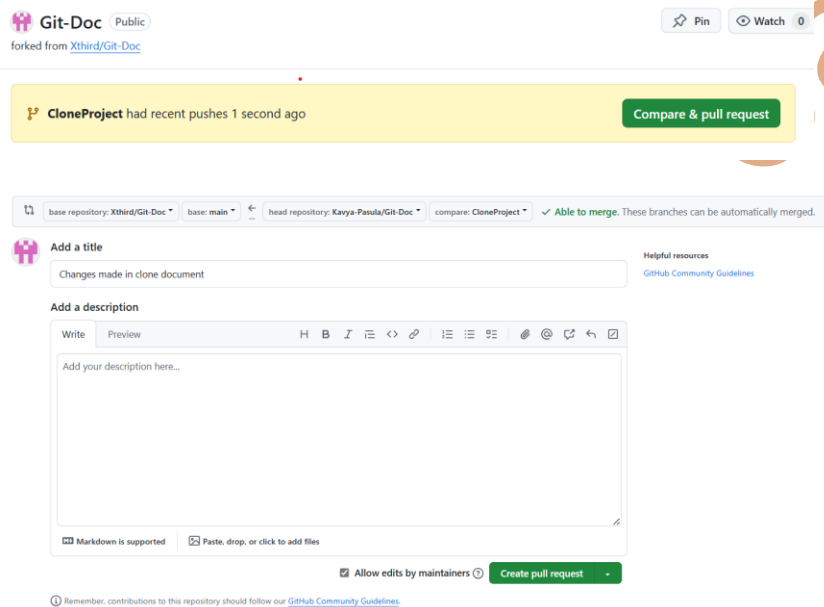
- create a branch to make some changes

```
C:\Users\Admin\Desktop\Git\Git-Doc>git status
On branch CloneProject
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:   Document.txt
```

- push the changes

```
C:\Users\Admin\Desktop\Git\Git-Doc>git push origin CloneProject
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 151.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'CloneProject' on GitHub by visiting:
remote:   https://github.com/Kavya-Pasula/Git-Doc/pull/new/CloneProject
remote:
To github.com:Kavya-Pasula/Git-Doc.git
* [new branch]      CloneProject -> CloneProject
```

5. open a pull request



4. Reverting Changes

- **Objective:** Learn how to undo changes in Git using commands like git revert, git reset, and git checkout.
- **Scenario:** Make a few changes to a project and commit them. Then, simulate a situation where a change needs to be undone or reverted to an earlier state.

1. git revert:- is a safe way to reversing changes without altering the project history.

```
PS C:\Users\Admin\Desktop\Git> git push
Revert "Changes made 2"
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 645 bytes | 161.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:Kavya-Pasula/Git.git
 c3dfc9b..ff26491 GitUseCase -> GitUseCase
PS C:\Users\Admin\Desktop\Git> git revert HEAD
[GitUseCase fcc404c] Revert "Changes made 2"
 1 file changed, 3 insertions(+), 1 deletion(-)
```

2. `git reset:-` command is used to undo changes in your Git repository. It can modify the staging area, working directory, or both, depending on the options used.

Note: `git reset --hard`, as they permanently delete changes.

```
PS C:\Users\Admin\Desktop\Git> git reset fcc404c
Unstaged changes after reset:
M      Changes.txt
M      sample.py
PS C:\Users\Admin\Desktop\Git> █
```

3. `git checkout:-` command is used to switch branches, restore files, or navigate to a specific commit in Git.

```
PS C:\Users\Admin\Desktop\Git> git checkout -b b1
Switched to a new branch 'b1'
PS C:\Users\Admin\Desktop\Git> █
```

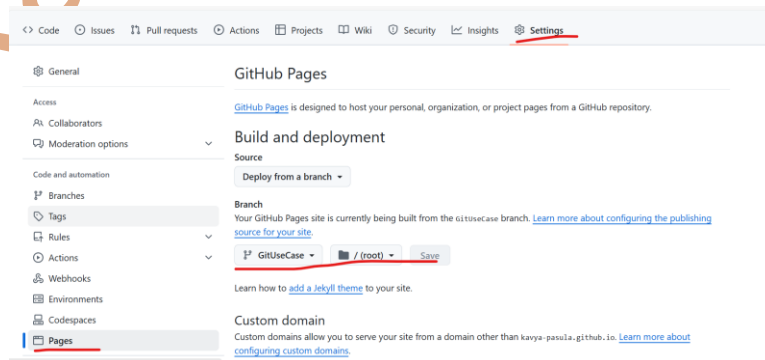
5. Using Git with GitHub Pages

- **Objective:** Practice deploying a simple website using GitHub Pages.
- **Scenario:** Create a static website (e.g., an HTML page) and host it on GitHub Pages. Learn how to push changes and see them reflected on the live website.


1. Create a static website (e.g., an HTML page) `index.html`

```
index.html U X
index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>My Website</title>
7  </head>
8  <body>
9      <h1>Welcome to My Website!</h1>
10     <p>This is a simple static website hosted on GitHub Pages.</p>
11 </body>
12 </html>
```

2. host it on GitHub Pages (<https://kavya-pasula.github.io/Git/>)



GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at <https://kavya-pasula.github.io/Git/>
Last deployed by  Kavya-Pasula 2 minutes ago

 Visit site

3. push changes

```
PS C:\Users\Admin\Desktop\Git> git add index.html
PS C:\Users\Admin\Desktop\Git> git commit -m "Edited index.html file"
[GitUseCase ef566fd] Edited index.html file
1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\Git> git push origin GitUseCase --force
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 306 bytes | 102.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:Kavya-Pasula/Git.git
f56b2d3..ef566fd GitUseCase -> GitUseCase
PS C:\Users\Admin\Desktop\Git>
```

6. Working with Git Tags

- **Objective:** Understand how to use tags in Git for marking specific points in history, such as releases.
- **Scenario:** Simulate a software release process by tagging specific commits as different versions (e.g., v1.0, v1.1). Practice pushing these tags to a remote repository.

Tags are commonly used to indicate software versions, making it easier to reference and check out specific releases in your project's history.

1. tagging specific commits as different versions (e.g., v1.0, v1.1).

```
PS C:\Users\Admin\Desktop\Git> echo "Working on Git Tags" >> Test.txt
PS C:\Users\Admin\Desktop\Git> git add Test.txt
PS C:\Users\Admin\Desktop\Git> git commit -m "Initial commit"
[GitUseCase 407840b] Initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\Admin\Desktop\Git> git tag v1.0
fatal: tag 'v1.0' already exists
PS C:\Users\Admin\Desktop\Git> echo "Working on Git Tags Feature update" >> Test.txt
PS C:\Users\Admin\Desktop\Git> git add Test.txt
PS C:\Users\Admin\Desktop\Git> git commit -am "Add feature"
[GitUseCase fd92be7] Add feature
1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\Admin\Desktop\Git> git tag v1.1
```

2. pushing these tags to a remote repository

```
PS C:\Users\Admin\Desktop\Git> git push origin --tags
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Kavya-Pasula/Git.git
* [new tag]          v1.0 -> v1.0
* [new tag]          v1.1 -> v1.1
PS C:\Users\Admin\Desktop\Git>
```

7. Stashing Changes

- **Objective:** Learn how to temporarily save changes using git stash and apply them later.
- **Scenario:** Simulate a situation where you need to switch branches but have uncommitted changes. Use git stash to save your work, switch branches, and then apply the changes back.

git stash allows you to save your work temporarily and switch branches without losing any changes.

1. git stash and apply

```
PS C:\Users\Admin\Desktop\Git> echo "Working on Git Stash" >> file.txt
PS C:\Users\Admin\Desktop\Git> git add file.txt
PS C:\Users\Admin\Desktop\Git> git stash
Saved working directory and index state WIP on GitUseCase: fd92be7 Add feature
PS C:\Users\Admin\Desktop\Git> git stash list
stash@{0}: WIP on GitUseCase: fd92be7 Add feature
PS C:\Users\Admin\Desktop\Git> git checkout main
branch 'main' set up to track 'origin/main'.
Switched to a new branch 'main'
PS C:\Users\Admin\Desktop\Git> git stash apply
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   file.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Git-Doc/
```

8. Exploring Git History

- **Objective:** Learn to explore the commit history using commands like git log and git diff.
- **Scenario:** Practice navigating the commit history to understand how changes evolved over time. Use git log to view commit details and git diff to see differences between commits.

1. git log:- command is used to display the commit history of a Git repository.

```
PS C:\Users\Admin\Desktop\Git> git log --oneline
ae4b894 (HEAD -> GitUseCase) Git cmd log and diff
91c1ee3 Git history cmd
fd92be7 (tag: v1.1, origin/GitUseCase) Add feature
407840b Initial commit
20a4b86 (tag: v1.0) Add feature
c2b0bb2 Initial commit
ef566fd Edited index.html file
f56b2d3 added index file
6c5e2bf added index file
fcc404c (b1) Revert "Changes made 2"
ff26491 Changes made 1
ef13c9c Changes made in the file
c3dfc9b Revert "Changes made inthe file"
97f2bd8 Changes made inthe file
df6a74d Merge branch 'NewFeature' into GitUseCase
ae2d464 Master file
bbf4aa9 (origin/NewFeature, NewFeature) Add new feature need to merge with main file
eccfa5e (master) print("Hello World!")
PS C:\Users\Admin\Desktop\Git> 
```

2. **git diff**:- command is used to show the differences between various states of a Git repository.

`git diff hash1 hash2`

```
PS C:\Users\Admin\Desktop\Git> git diff ae4b894 91c1ee3
diff --git a/history.txt b/history.txt
index 74c610b..7c8403c 100644
Binary files a/history.txt and b/history.txt differ
PS C:\Users\Admin\Desktop\Git>
```

9. Git Hooks

- **Objective:** Learn how to automate tasks using Git hooks.
 - **Scenario:** Set up a pre-commit hook to check for code formatting or linting before allowing a commit. This can help enforce coding standards.
1. **Git hooks**:- Navigate to `C:\Users\Admin\Desktop\Git\.git\hooks` We can see `pre-commit.sample` file is a template for creating a pre-commit hook in Git. Rename `pre-commit.sample` to `pre-commit`.
 2. Create empty text file and add to git repository and try do commit that file into git repository

```
PS C:\Users\Admin\Desktop\Git> echo " " >> efile.txt
PS C:\Users\Admin\Desktop\Git> git add efile.txt
PS C:\Users\Admin\Desktop\Git> git commit -m "Adding empty file in Git"
[GitUseCase 3fcd0a3] Adding empty file in Git
1 file changed, 0 insertions(+), 0 deletions(-)
```

Note: linting :- Code formatting and linting are both practices aimed at improving code quality.

10. Git Rebase

- **Objective:** Understand how to use git rebase to integrate changes from one branch into another.
 - **Scenario:** Create a feature branch from the main branch, make some commits, then rebase the feature branch onto the main branch to bring it up to date.
1. **Git rebase** is a command that allows you to integrate changes from one branch into another by rewriting the commit history.

```
PS C:\Users\Admin\Desktop\Git> echo "Update in main branch" >> update.txt
PS C:\Users\Admin\Desktop\Git> git add update.txt
PS C:\Users\Admin\Desktop\Git> git commit -m "update in main"
[main bc87dde] update in main
1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\Admin\Desktop\Git> git checkout GitUseCase
M
main.txt
Switched to branch 'GitUseCase'
Your branch and 'origin/GitUseCase' have diverged,
and have 28 and 18 different commits each, respectively.
(use "git pull" if you want to integrate the remote branch with yours)
PS C:\Users\Admin\Desktop\Git> echo "Added feature in child branch" >> data.txt
PS C:\Users\Admin\Desktop\Git> git add data.txt
PS C:\Users\Admin\Desktop\Git> git commit -m "Added feature in child"
```

```
PS C:\Users\Admin\Desktop\Git> git rebase main
```