Name – Pratik Sawant, Sankalp Wani
Roll No – 22101A0029, 22101A0028
Branch – TE INFT A

Experiment 2: Automating Version Control and Code Collaboration of TE Mini Project with Git

Aim: To Perform Code Collaboration of TE Mini Project using GIT workflow

Problem:

TechCo's development team is collaborating on a new feature for their application. However, multiple developers are working on the same parts of the codebase, causing merge conflicts, inconsistent commits, and errors in version control. The team struggles to manage branches effectively and maintain a clean history. The current workflow relies heavily on manual processes, making it difficult to track and resolve issues efficiently.

Implementation:

- Set up a Git repository for a project (either create a new project or use an existing one).
- Collaborate with a team member by creating multiple branches for different features.
- Implement the following:
    - Create a feature branch and make changes to the code.
    - Push changes to the remote repository.
    - Use a Git cheat sheet to commit, merge branches, and resolve conflicts.
    - Create a pull request (PR) and ensure code review processes are followed.

Step 1: Initialize a Git Repository and Push It to GitHub

1.1 Initialize a Git Repository Locally

- Navigate to your project directory on your local machine (or create a new project folder).
- Open your terminal and run the following command to initialize a Git repository:
  `git init`

1.2 Add Files to the Repository

- Add a new file or make changes to existing files. For example, create a `README.md` file in the project folder:
  `echo "# My Project" > README.md`

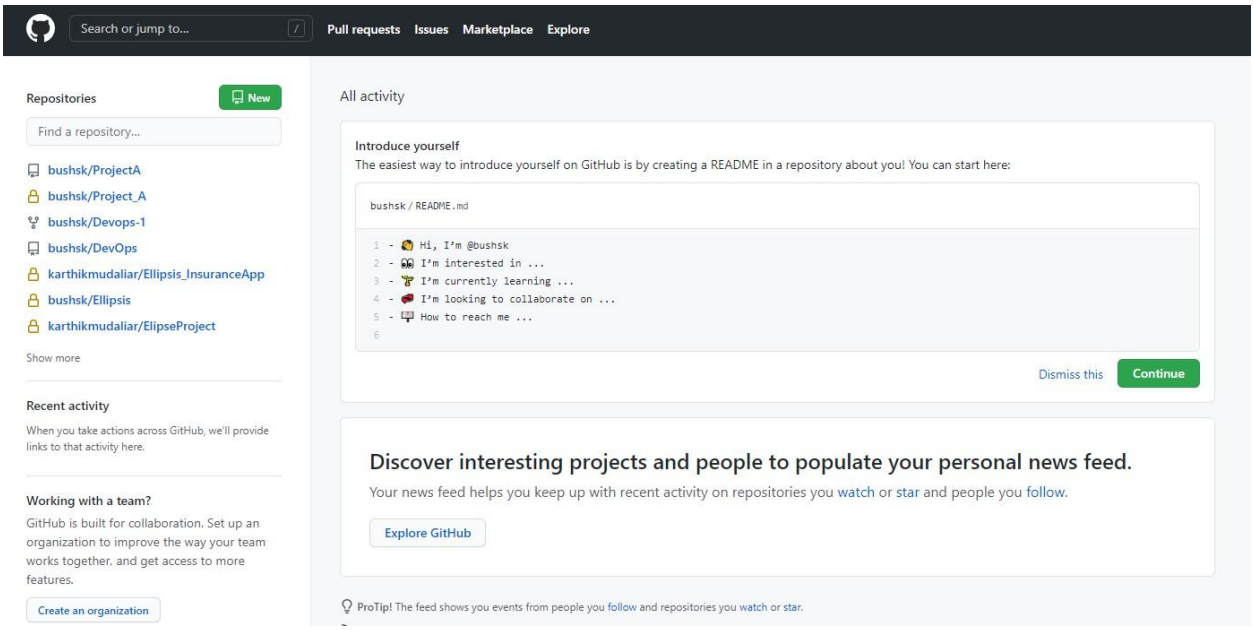
1.3 Stage and Commit Files

- Stage the files for commit using:
  `git add .`

- Commit the staged files with a message:
  git commit -m "Initial commit"

1.4 Create a GitHub Repository

- Go to GitHub and create a new repository. You can name it something like "my-project".
- Copy the URL of the GitHub repository, e.g., `https://github.com/your-username/my-project.git`.



1.5 Push to GitHub

- Set the remote repository URL:
  `git remote add origin https://github.com/your-username/my-project.git`

- Push the local repository to GitHub:
  `git push -u origin master`

Step 2: Create and Merge Multiple Feature Branches

2.1 Create a New Feature Branch

- Create a new branch for a feature. For example, create a `feature-login` branch:

  ```
  git checkout -b feature-login
  ```

2.2 Make Changes in the Feature Branch

- Add a new file or modify an existing file in the `feature-login` branch. For example, modify `README.md`:

```
echo "Login feature" >> README.md
```

2.3 Stage and Commit Changes

- Stage the changes and commit them:

```
git add README.md
git commit -m "Add login feature to README"
```

2.4 Push the Feature Branch to GitHub

- Push the feature branch to GitHub:

```
git push origin feature-login
```

2.5 Create Another Feature Branch

- Create another branch for a different feature, for example, `feature-registration`:

```
git checkout -b feature-registration
```

2.6 Make Changes in the `feature-registration` Branch

- Modify the same file (`README.md`), but with content related to registration:

```
echo "Registration feature" >> README.md
```

2.7 Stage and Commit Changes

- Stage and commit the changes:

```
git add README.md
git commit -m "Add registration feature to README"
```

2.8 Push the `feature-registration` Branch to GitHub

- Push the registration branch to GitHub:

```
git push origin feature-registration
```

Step 3: Simulate a Merge Conflict and Resolve It

3.1 Switch to `master` Branch

- Now switch back to the `master` branch:

```
git checkout master
```

3.2 Merge `feature-login` into `master`

- Merge the `feature-login` branch into the `master` branch:

```
git merge feature-login
```

- Since the changes are in different parts of the file, the merge will succeed without conflict.

3.3 Merge `feature-registration` into `master`

- Now, try to merge the `feature-registration` branch:

```
git merge feature-registration
```

- Since the `feature-registration` branch made changes to the same file (`README.md`) at the same location as `feature-login`, Git will raise a merge conflict.

3.4 Resolve the Merge Conflict

- Open the `README.md` file. You will see something like this:

```
<<<<<<< HEAD
Login feature
=======
Registration feature
>>>>>>> feature-registration
```

- Edit the file to resolve the conflict. For example, you might combine both features like this:
  `markdown`

```
Login feature
Registration feature
```

3.5 Stage the Resolved Conflict

- Once the conflict is resolved, stage the file:
  `git add README.md`

3.6 Commit the Merge

- Commit the merge with a message:
  `git commit -m "Resolve merge conflict between feature-login and feature-registration"`

Step 4: Perform a Pull Request Review and Handle the Integration Process

4.1 Create a Pull Request (PR) on GitHub

- Go to your repository on GitHub.
- You should see a button to create a Pull Request for the branches you've pushed (`feature-login` and `feature-registration`).
- Click on "New Pull Request" and select the `feature-login` branch and compare it with `master`.
- After reviewing the changes, click "Create Pull Request."
- Add a description of the changes and create the PR.

4.2 Review the Pull Request

- Review the changes in the pull request.
- You can see the changes made and leave comments on specific lines of code if necessary.
- You can ask a team member to review it as well or approve it yourself.

4.3 Merge the Pull Request

- After the PR review, click the "Merge pull request" button.
- Choose "Confirm merge" to integrate the changes from `feature-login` into `master`.

4.4 Repeat the Process for `feature-registration`

- Repeat the same process for the `feature-registration` branch.
- Create a new PR for `feature-registration`, review it, and merge it into `master`.

4.5 Clean Up the Branches

- After successfully merging the feature branches, delete them from GitHub (optional):

```
git push origin --delete feature-login
git push origin --delete feature-registration
```

Step 5: Connect GitHub Cloud to Jira

These instructions are for connecting **GitHub Cloud** or **GitHub Enterprise Cloud** to Jira. Show me how to connect GitHub Enterprise Server

When you connect a GitHub Cloud organization to Jira, your team can link their development activity to Jira issues. This lets you track branches, commits, and pull requests in the context of your Jira issues, on your Jira board, in the releases feature, and more.

Before you begin

To install and set up the GitHub for Jira app, you need:

- Site administrator permission for your Jira site.
- Organization owner permission for your GitHub organization.

For some organizations, the task of connecting GitHub to Jira might involve multiple team members:

- A Jira site admin will install the GitHub for Jira app.
- A GitHub organization owner will connect a GitHub organization to your Jira site.

5.1 Install the GitHub for Jira app

1. In Jira, select **Apps**, then select **Explore more apps**.
2. Search for **GitHub for Jira**, then select it from the results.
3. Select **Get app**, then **Get it now**.

5.2 Connect a GitHub organization

1. After the app is installed, select **Get started**. If the app is already installed on your Jira site, you can find this section by selecting **Apps**, then **Manage your apps**, and then **GitHub for Jira**.
2. Select **Continue**.
3. Select **GitHub Cloud**, then **Next**.
4. Enter your GitHub username and password, then **Sign in**.

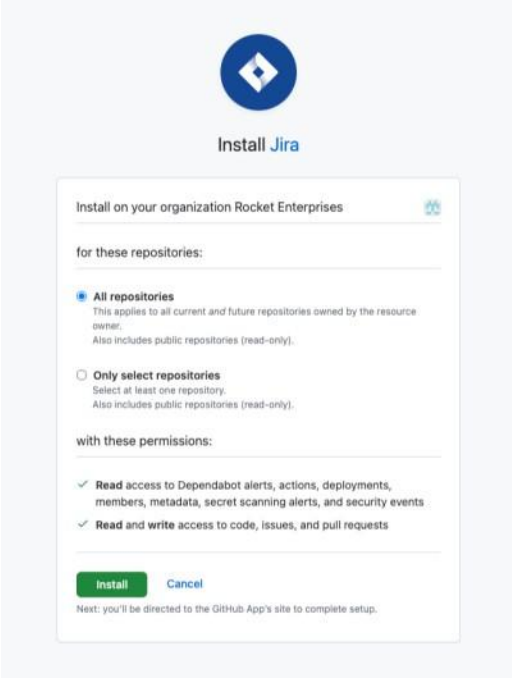
5. Find the organization you want to connect to Jira, then select **Connect**.

To check your permissions for a GitHub organization, open your GitHub organization settings and look for your permission level next to the organization name. Organization owners can review and accept permission requests from Jira in your organization settings. More about required permissions for GitHub for Jira

5.3 Add the Jira app to a new GitHub organization

If no organizations are available to connect to Jira, you'll need to install the Jira app on a new GitHub organization.

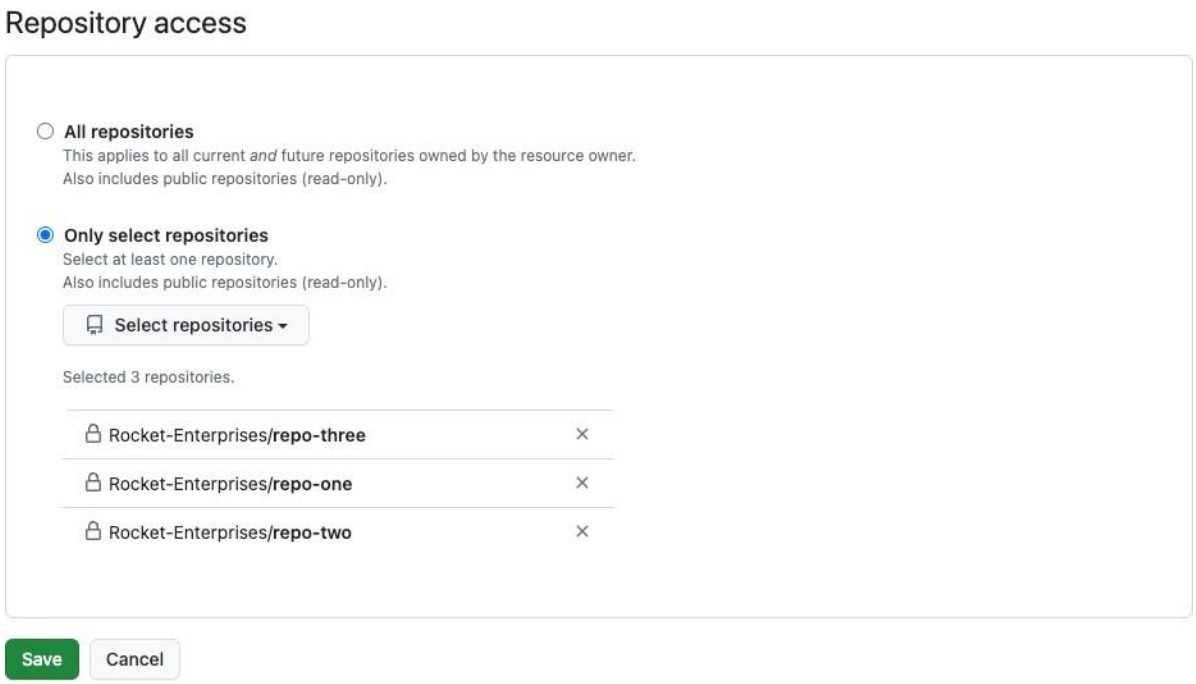
1. From step five in the instructions above, **Select an organization in GitHub.**
2. The GitHub site will open in a new tab and show a list of all the organizations available in your GitHub account. Select the organization you want to connect to Jira.
3. Choose the repositories you want to give Jira access to by selecting either **All repositories** or **Only select repositories**.
4. Select **Install.**



5.4 Connect a new GitHub repository

- If you selected **All repositories** when you connected a GitHub organization, Jira will be able to access all the repositories in that organization, including any new repositories you create.
- If you selected **Only select repositories**, any new repositories you create won't be added automatically. Here's how to add them:

1. In Jira, select **Apps**, then select **Manage apps**.
2. In the sidebar, under **GitHub for Jira**, select **Configure.**
3. Find the organization that contains the repo you want to add and select the three-dot menu (**...**) to view available actions.
4. Select **Configure**.
5. A new tab will open with your GitHub organization settings. Under **Repository access**, select the dropdown **Select repositories**.
6. Select the repository you want to connect to, then **Save.**



If you use IP allowlists in your GitHub organization, you might have issues using GitHub for Jira, even if the correct IP addresses are in your IP allowlist. Here's the workaround: How to update your GitHub IP allowlist configuration

Still need help? Raise an issue with our team.

5.5 Link your development information to Jira issues

To link branches, commits, and pull requests to Jira, your team must include Jira issue keys in their development actions.

1. Find the issue key for the Jira issue you want to link to, for example "JRA-123". You can find the key in several places in Jira:
   • On the board, issue keys appear at the bottom of a card.
   • On the issue's details, issue keys appear in the navigation at the top of the page.
2. Check out a new branch in your repo, using the issue key in the branch name. For example, `git checkout -b JRA-123-<branch-name>.`
3. When committing changes to your branch, use the issue key in your commit message to link those commits to the development panel in your Jira issue.
   For example, `git commit -m "JRA-123 <summary of commit>".`
4. When you create a pull request, use the issue key in the pull request title.

- After you push your branch, you'll see development information in your Jira issue.

How to use the GitHub for Jira app

- Link GitHub development information to Jira issues
- Link GitHub workflows and deployments to Jira

Post lab exercise:

Instructions:

1. Initialize a Git repository and push it to GitHub.
2. Create and merge multiple feature branches.
3. Simulate a merge conflict and resolve it.
4. Perform a pull request review and handle the integration process.

1. Initial Setup:

Each developer should clone the repository and create a feature branch as instructed.

```
# Clone the repository
git clone https://github.com/TechCo/your-repo.git

# Navigate into the project directory
cd your-repo

# Create a feature branch
git checkout -b feature/john-does-feature
```

Developer 1 and **Developer 2** should create separate feature branches based on the above instructions. For example:

- Developer 1 creates the branch: `feature/john-feature`
- Developer 2 creates the branch: `feature/jane-feature`
- Paste the Output here:

2. Collaborative Changes:

Both developers will make changes to the same file (`app.js`), but in different parts of the code.

Developer 1's changes:

```
// Modify a function in app.js
function greetUser() {
    console.log("Hello, user!");
    console.log("Welcome to TechCo's new app.");
}
```

Commit the changes:

```
git add app.js
git commit -m "Add greeting message for new users"
```

```
git push origin feature/john-feature
```

Developer 2's changes (on the same line):

```
// Modify the same function in app.js
function greetUser() {
    console.log("Hello, user!");
    console.log("We hope you enjoy using TechCo's app.");
}
```

Commit the changes:

```
git add app.js
git commit -m "Update greeting with a new message"
git push origin feature/jane-feature
```

- Paste the Output here:

3. Merge Conflict:

Once both developers have pushed their changes, it's time to merge their feature branches into `main`. This will result in a merge conflict since both developers modified the same lines of code in `app.js`.

Steps for Merging and Resolving the Conffict:

- Switch to the `main` branch:
  ```
  git checkout main
  git pull origin main  # Fetch the latest changes from the main branch
  ```

- Try merging one of the feature branches into `main` (e.g., Developer 1's branch):
  ```
  git merge feature/john-feature
  ```

- At this point, Git will notify that a conflict exists in `app.js`. The conflicted file will be marked as having conflicts. To resolve it:

1. Open `app.js` in a code editor. You will see something like this:

```
function greetUser() {
    console.log("Hello, user!");
<<<<<<< HEAD
    console.log("Welcome to TechCo's new app.");
=======
    console.log("We hope you enjoy using TechCo's app.");
>>>>>>> feature/jane-feature
}
```

2. The conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`) indicate where the conflict occurs. You need to decide which version to keep or how to merge the changes. Here's how you can resolve it:

```
function greetUser() {
    console.log("Hello, user!");
    console.log("Welcome to TechCo's new app.");
    console.log("We hope you enjoy using TechCo's app.");
}
```

3. After resolving the conflict, remove the conflict markers and save the file.

- Add the resolved file and commit the changes:
-
```
git add app.js
git commit -m "Resolve merge conflict in app.js"
git push origin main
```

Paste the Output here:

4. Rebasing Practice:

Now, Developer 1 and Developer 2 will fetch the latest changes from the `main` branch and rebase their feature branches onto `main` to ensure that their code is up-to-date.

Developer 1 Rebasing:

```
# Fetch the latest changes from the main branch
git fetch origin

# Rebase the feature branch onto main
git rebase origin/main

# Resolve any conflicts if necessary, following the same conflict
resolution process above
# After resolving conflicts, continue the rebase
git rebase --continue

# Push the rebased branch
git push origin feature/john-feature --force
```

Developer 2 Rebasing:

Developer 2 will repeat the same steps to rebase their branch.

```
git fetch origin
git rebase origin/main
git rebase --continue
git push origin feature/jane-feature --force
```

5. Reviewing Commit History:

Once the merge and rebasing are complete, both developers should inspect the commit history to ensure that their commits are clear and meaningful.

```
git log --oneline --graph
```

Ensure that each commit message follows a clear and consistent format, e.g., "Fix bug in the greeting function" or "Add new user welcome message."

6. Final Task: Creating Pull Requests (PRs):

- Developer 1 and Developer 2 should now open pull requests on GitHub to merge their feature branches into the `main` branch.

Developer 1's PR:

- Title: `Merge feature/john-feature into main`
- Description: "Added a new greeting message for first-time users."

Developer 2's PR:

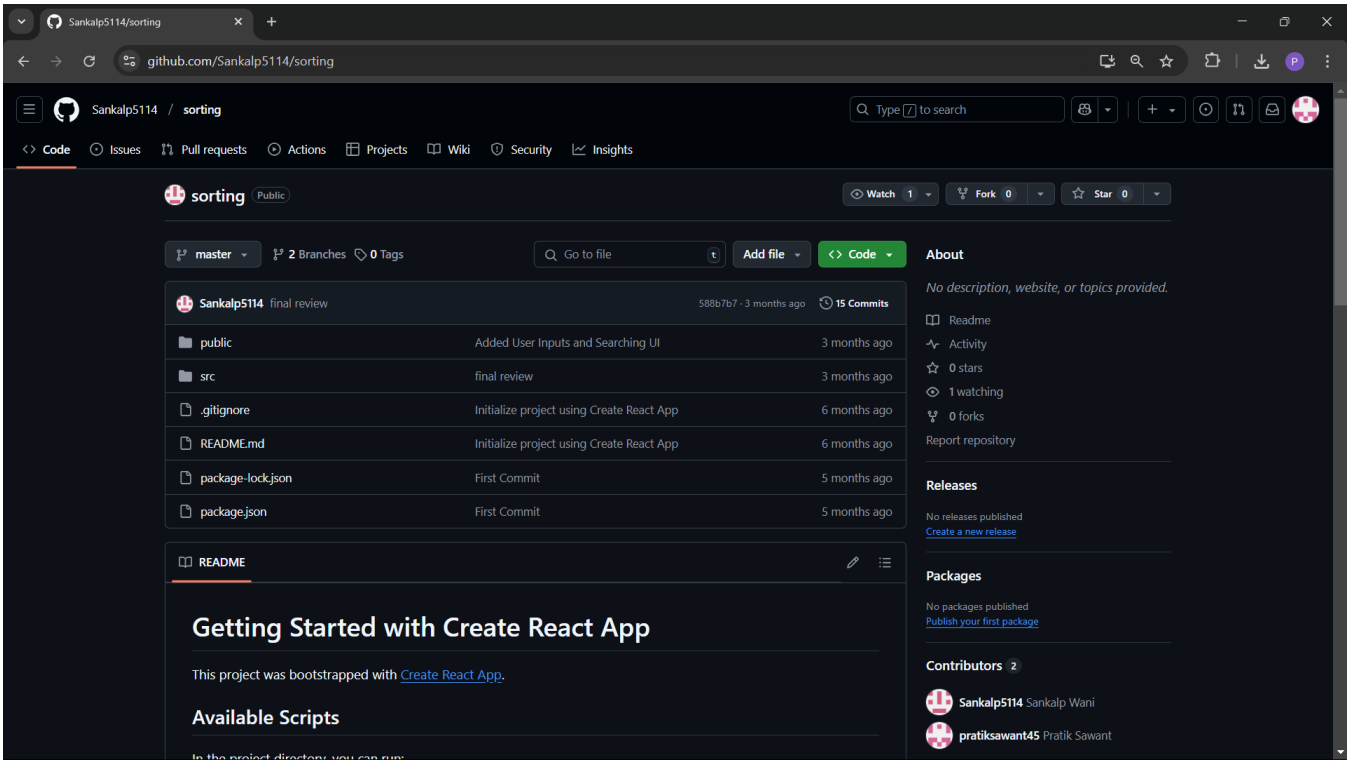
- Title: `Merge feature/jane-feature into main`
- Description: "Updated the greeting message to include additional text for users."

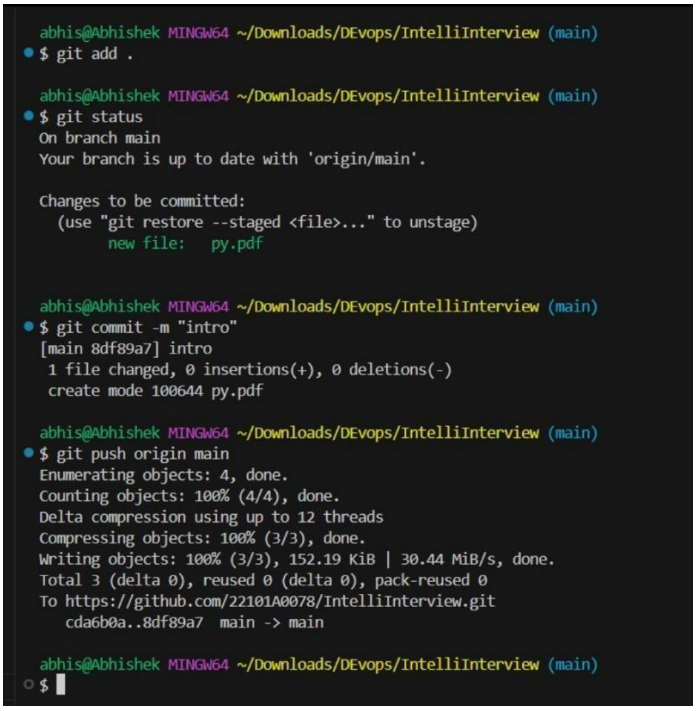
Once both pull requests are reviewed by the team, they can be merged into the `main` branch.

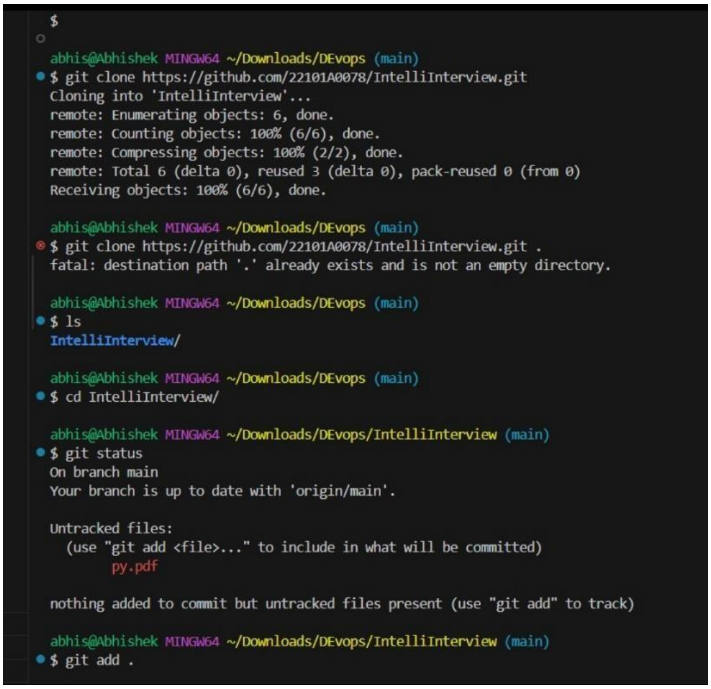
Conclusion: By following these steps, the developers will:

- Learn how to resolve merge conflicts effectively.
- Practice using rebase to keep their branches up to date with `main`.
- Understand how to maintain clean commit histories and write meaningful commit messages.
- Gain hands-on experience with Git workflows that are crucial for collaborative development.

OUTPUT-


Sankalp5114/sorting
github.com/Sankalp5114/sorting
Sankalp5114 / sorting
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
sorting
Public
Watch 1
Fork 0
Star 0
master
2 Branches
0 Tags
Go to file
Add file
Code
Sankalp5114 final review
588b7b7 · 3 months ago
15 Commits
public
Added User Inputs and Searching UI
3 months ago
src
final review
3 months ago
.gitignore
Initialize project using Create React App
6 months ago
README.md
Initialize project using Create React App
6 months ago
package-lock.json
First Commit
5 months ago
package.json
First Commit
5 months ago
README
Getting Started with Create React App
This project was bootstrapped with Create React App.
Available Scripts
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Report repository
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 2
Sankalp5114 Sankalp Wani
pratiksawant45 Pratik Sawant


abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git add .
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git status
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: py.pdf
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git commit -m "intro"
[main 8df89a7] intro
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 py.pdf
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 152.19 KiB | 30.44 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/22101A0078/IntelliInterview.git
cda6b0a..8df89a7 main -> main
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$


$
abhis@Abhishek MINGW64 ~/Downloads/DEvops (main)
$ git clone https://github.com/22101A0078/IntelliInterview.git
Cloning into 'IntelliInterview'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
abhis@Abhishek MINGW64 ~/Downloads/DEvops (main)
$ git clone https://github.com/22101A0078/IntelliInterview.git .
fatal: destination path '.' already exists and is not an empty directory.
abhis@Abhishek MINGW64 ~/Downloads/DEvops (main)
$ ls
IntelliInterview/
abhis@Abhishek MINGW64 ~/Downloads/DEvops (main)
$ cd IntelliInterview/
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
Untracked files:
(use "git add <file>..." to include in what will be committed)
py.pdf
nothing added to commit but untracked files present (use "git add" to track)
abhis@Abhishek MINGW64 ~/Downloads/DEvops/IntelliInterview (main)
$ git add .