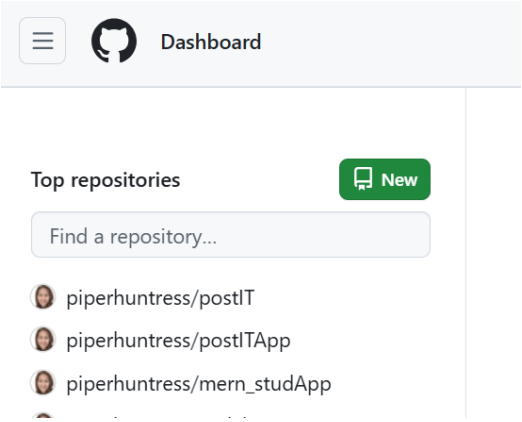
Activity 14 – Git and GitHub for Version Control

Activity 14 –Git Version Control and GitHub

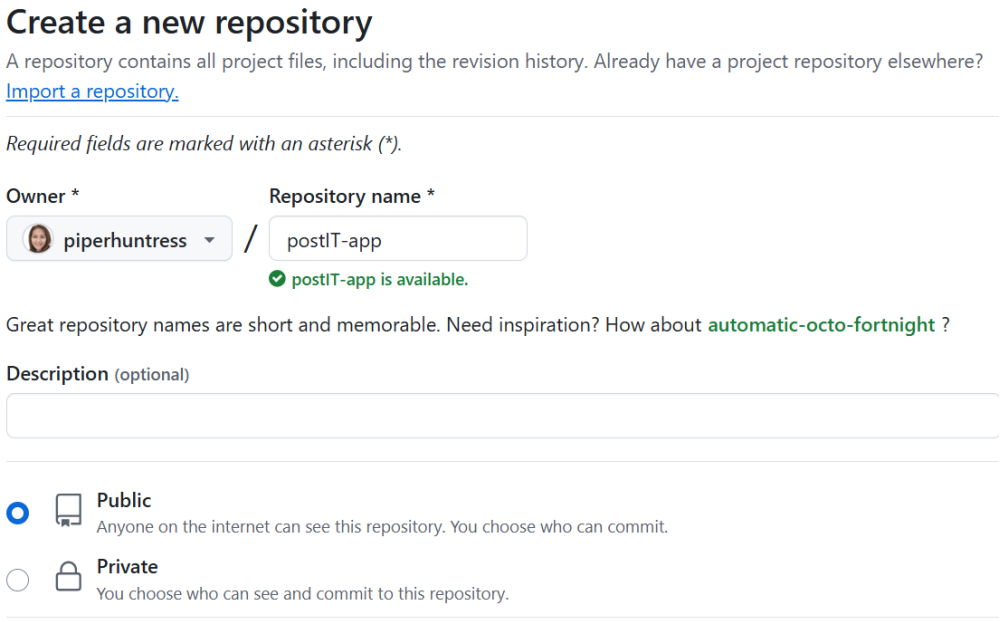
The objectives of this activity is to:

a) Use Git version control to manage and track changes to project files.Use Git version control to manage

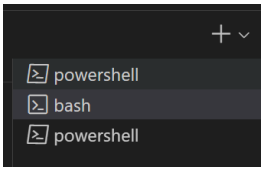
1. Register to https://github.com/ .
2. In the dashboard, click the new button to create a new repository.



3. Write the name of the repository, set it as public. Click the create repository button below.



4. In VS Code, open your project and open a new GIT op Bash terminal.



Generate new SSH Key

- Open Git Bash
- Generate SSH Key

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

This creates a new SSH key, using the provided email as a label.

```
> Generating public/private ALGORITHM key pair.
```

When you're prompted to "Enter a file in which to save the key", you can press Enter to accept the default file location. Please note that if you created SSH keys previously, ssh-keygen may ask you to rewrite another key, in which case we recommend creating a custom-named SSH key. To do so, type the default file location and replace id_ALGORITHM with your custom key name.

```
> Enter file in which to save the key (/c/Users/YOU/.ssh/id_ALGORITHM):[Press enter]
```

Adding a new SSH key ot your GitHub account

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

copy the public key .

- In the upper-right corner of any page on GitHub, click your profile photo, then click **Settings**.
- In the "Access" section of the sidebar, click **SSH and GPG keys**.
- Click **New SSH key** or **Add SSH key**.
- In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal laptop, you might call this key "Personal laptop".
- In the "Key" field, paste your public key.
- Click **Add SSH key**.

Testing your SSH connection

- Open Git Bash.
- Enter the following:

```
ssh -T git@github.com
# Attempts to ssh to GitHub
```

Verify that the fingerprint in the message you see matches GitHub's public key fingerprint. If it does, then type `yes`:

```
> Hi USERNAME! You've successfully authenticated, but GitHub does not
> provide shell access.
```

Verify that the resulting message contains your username

Configuring git for the first time use

5. Configure git for the first time use by setting your identity. Provide your github username and email address. Execute the command in git bash terminal.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

6. Execute the steps to initialize a new repository or an existing repository in the git bash terminal.

New Repository

```
echo "# utas-workshop-testing" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:<USERNAME>/<Repository-Name>.git
git push -u origin main
```

Existing Repository

```
git remote add origin git@github.com:<USERNAME>/<Repository-Name>.git
git branch -M main
git push -u origin main
```

7. Go to your client folder and cut the .**gitignore** file and copy to your root folder. The **.gitignore** file should be saved at the root of the project directory.

   /mern-project
   ├── .gitignore **<-- Place the .gitignore file here**
   ├── client/
   ├── server/

8. Open the .gitignore file and change the **/node_modules** to **node_modules/.**

node_modules/: This tells Git to ignore all node_modules directories, no matter where they appear in the repository (root, subfolders, etc.). Since we have two node_modules both client and server folders.

9. In .gitignore file, add the line to ignore all the **.env** files.

```
# Ignore all .env files
.env*
```

10. In the **git bash** terminal, execute the command to stage the files.

    ```
    $ git add .
    ```

11. Check status.

    ```
    $ git status
    ```

12. Get updates from the remote repository to synchronize the local repository.

    ```
    $ git fetch
    ```

13. Commit your changes. Message is the description of the changes made to the repository.

    ```
    $ git commit -m 'your message'
    ```

14. Push changes to the remote repository for example in the main branch.

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

Example for executing these commands:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Jasmine@Jaja-HP MINGW64 /d/react/FullStack/3-sem1Ay24-25-postITComplete/postitapp (main)
$ git add .

Jasmine@Jaja-HP MINGW64 /d/react/FullStack/3-sem1Ay24-25-postITComplete/postitapp (main)
$ git commit -m "update users component"
Auto packing the repository in background for optimum performance.
See "git help gc" for manual housekeeping.
Enumerating objects: 72, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Checking connectivity: 33730, done.
 4 files changed, 6 insertions(+), 2 deletions(-)
 create mode 100644 client/src/Components/Location.zip
 rename server/uploads/{1732462765330-pic-removebg-preview.jpg => 1732633180463-pic-removebg-prev
iew.jpg} (100%)
```

```
Jasmine@Jaja-HP MINGW64 /d/react/FullStack/3-sem1Ay24-25-postITComplete/postitapp (main)
$ git push -u origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.63 KiB | 1.63 MiB/s, done.
Total 10 (delta 7), reused 3 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 7 local objects.
To https://github.com/piperhuntress/postIT-app.git
   5216d71..d8e81f9  main -> main
branch 'main' set up to track 'origin/main'.

Jasmine@Jaja-HP MINGW64 /d/react/FullStack/3-sem1Ay24-25-postITComplete/postitapp (main)
$ 
```

15. After pushing to the remote repository, go to your GitHub account and check the pushed files.