Name: Yash Sanjay Suryawanshi

Roll No: 61 Class: TE-IT

LP-II: Web Application Development

Assignment 2A: Create version control account on GitHub and use Git commands to create repository and push your code to GitHub.

- **What is Git?**

Git is a version control system designed to help developers track changes in their projects, manage code versions, and collaborate effectively.

- **Key Features of Git:**
  1. **Version Control System**: Git keeps a detailed history of changes in code, enabling developers to revert to earlier versions, compare updates, and track contributions.
  2. **Local Repository Management**: It allows developers to store the entire project history locally, making it possible to work offline and sync changes with others later.
  3. **Branching and Merging**: Git supports the creation of branches for testing or adding new features. These branches can be merged back into the main project without disturbing the original code.
  4. **Distributed System:** Git is decentralized, meaning every developer has a complete copy of the repository, ensuring resilience and flexibility in case of system failures.
  5. **Lightweight and Fast:** Git is designed for speed, allowing quick commits, branching, and merging, even for large projects.

- **What is GitHub?**

GitHub is a cloud-based platform for hosting Git repositories, designed to help developers collaborate on projects, share code, and manage version control efficiently.

- **Key Features of GitHub:**
  1. **Repository Hosting**: GitHub provides a central location to store Git repositories online, making it easy to access, manage, and back up code from anywhere.
  2. **Version Control Integration**: GitHub seamlessly integrates with Git, allowing developers to push, pull, and synchronize changes, making code management smoother and more reliable.
  3. **Community and Open Source**: GitHub serves as a hub for open-source contributions, allowing developers to share projects publicly, contribute to others' work, and build a professional portfolio.
  4. **Collaboration Tools:** GitHub offers features like pull requests, code reviews, and issue tracking, making teamwork more efficient and organized.
  5. **GitHub Codespaces:** GitHub offers Codespaces, a cloud-based development environment that lets developers write, test, and debug code directly from their browser.

♦Git Architecture:

This diagram illustrates the basic architecture and workflow of **Git**, showcasing how changes move between different components and repositories.

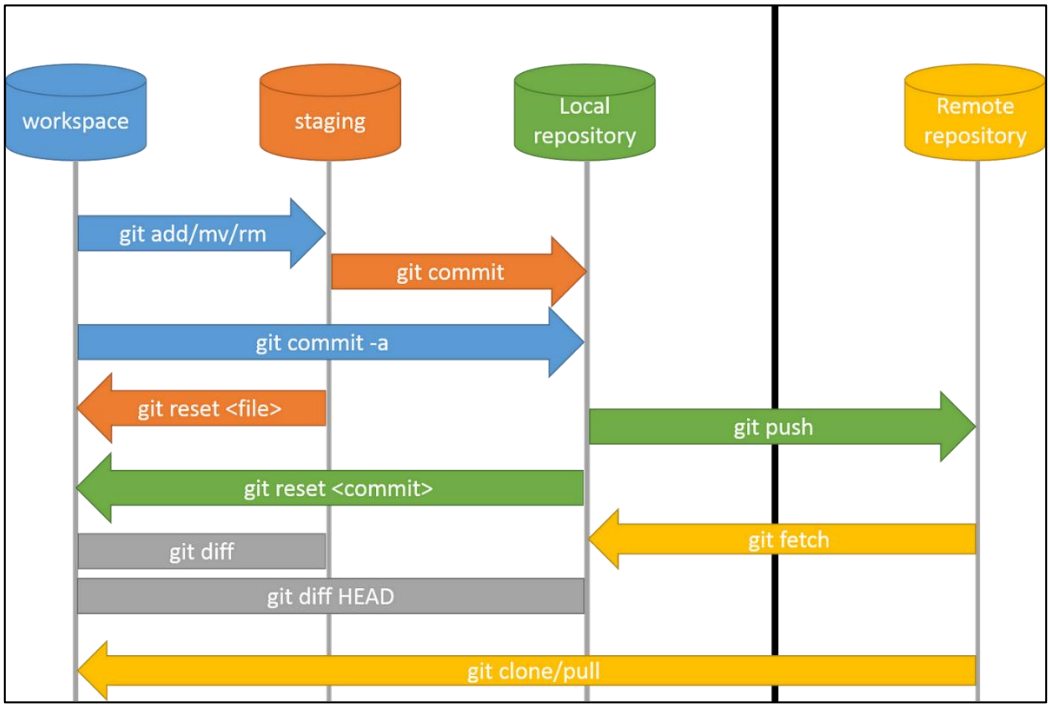

Figure 1: Git Architecture

1. **Workspace**:
   - This is your **working directory** where operations on files(add, edit, delete, etc.) happen.
   - To track changes, files need to be added to the **staging area**. It is done by using "**`git add .`**"

2. **Staging Area:**
   - The staging area is a temporary holding area for files before they are committed.
   - It allows for selective commits, enabling you to choose which files or changes to include in the next commit.
   - This provides fine-grained control over what gets included in the next version of your project.

3. **Local Repository**:
   - The local repository is a private storage space on your computer that keeps a complete history of your project.
   - Once changes are staged, they are permanently recorded in the local repository using **`git commit`**.
   - This acts as a backup and lets you revert to earlier versions if needed.

4. **Remote/Global Repository:**
   - It is a centralized repository hosted on a remote server (e.g., GitHub, GitLab, Bitbucket) that stores the complete history of your project.
   - Enables sharing your project with others, collaborating on code, and backing up your work.
   - Provides a central location for accessing and managing the project's history.
     - **`git push`**: Sends commits from your local repository to the remote repository.
     - **`git pull`**: Fetches changes from the remote repository and merges them into your local branch.

♦ Git Workflow: Step-by-Step Implementation:

1. **Create a Folder:** Create a new folder for your project and navigate to it.
2. **Add a File:** Add any new file (e.g., index.html, README.md) to the folder.
3. **Open Git Bash:** Right-click inside the folder and open it with Git Bash.
4. **Initialize Git:** Run **`git init`** to initialize the folder as a Git repository.

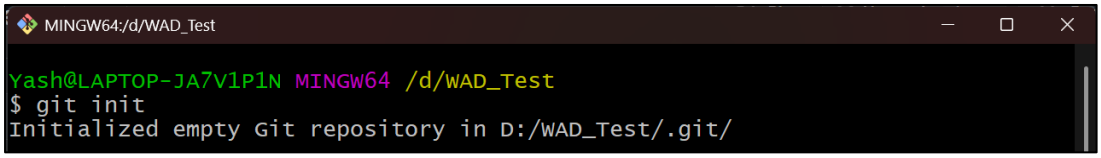

Figure 2: Initializing folder with git init command.

5. **Stage Changes:** Use "**`git add .`**" to add all files in the folder to the staging area.
6. **Commit Changes:** Run **`git commit -m "Any message"`** to save the staged changes to the local repository. Use **`git status`** to check status of file.

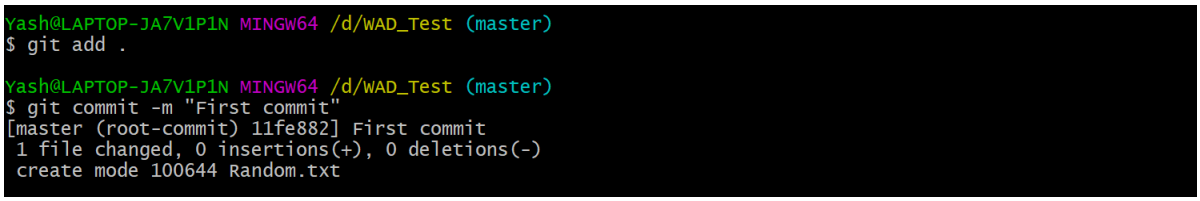

Figure 3: Add & Commit commands

7. **Create a GitHub Repository:** On GitHub, create a new repository but do not add a README. Copy the repository URL(HTTPS with *.git* at the end).
8. **Generate Personal Access Token:** Go to GitHub Settings > Developer Settings > Personal Access Tokens and generate a token with the required scopes (check all in this case). **Copy it** and store in a secure place.
9. **Add Remote Repository:** Adding a remote repository means linking your local Git repository to a repository hosted on a remote platform i.e. GitHub. Follow the structure of the URL as: https://<personal_access_token>@github.com/username/repository-name.git

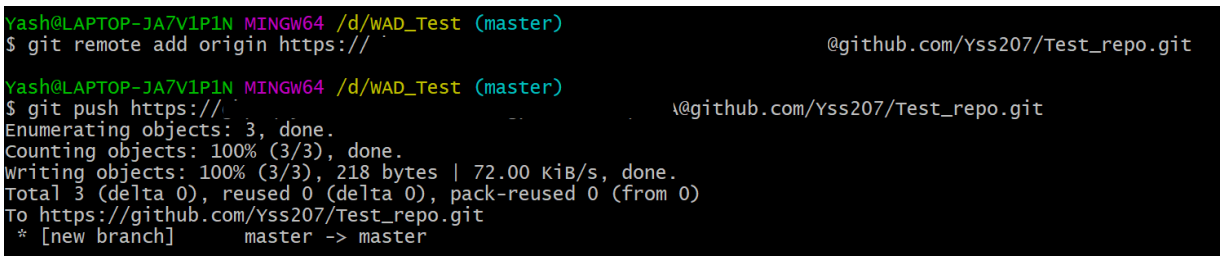

Figure 4: Setting Remote Repository & Performing Push(Hidden in the figure)

10. **Perform Initial Push using PAT:** Above screenshot also shows the initial push from the local repository to the remote repository, done using PAT. After successful execution of the command, file should be visible on the GitHub repository as well, along with its respective commit message.

11. **Upstream option:** Use **`git push –set-upstream <remote> <branch>`**. What it does:
    - Links the local `master` branch to the remote `master` branch in the `origin` remote repository.
    - After this, you can simply run `git push` or `git pull` without specifying the branch name.

♦ In case of any error of authentication, perform following commands:

1. **`git config –global user.name "GitHubUsername"`**
2. **`git config –global user.email "GitHubEmail"`**

These commands should resolve any issues faced while authentication. Ideal to perform them before using the Personal Access Token to set the remote repository.

♦ Git Clone & Git Pull:

1. **Git Clone:** *git clone* is used to create a copy of a remote repository on your local machine. It also sets up the remote origin automatically, allowing you to easily push and pull changes from the remote repository.
   Syntax: `git clone <repository-url>`

2. **Git Pull:** ***git pull*** fetches updates from the specified remote repository (e.g., origin) and the specified branch. It then merges the changes into your current local branch, keeping your local project up to date.
   Syntax: `git pull <remote> <branch>`

♦ Example of Git Clone & Pull:

Step 1: Create a Repository and Add a File on GitHub

1. Navigate to GitHub and create a new repository.
2. Add a file (e.g., example.txt) to the repository and save changes.

Step 2: Clone the Repository

1. Create any new folder and open Git Bash from that location.
2. Run the git clone command.
3. This will create a local copy of the repository in a new folder named after the repository(*in this case, Wad_tp folder will be created*).

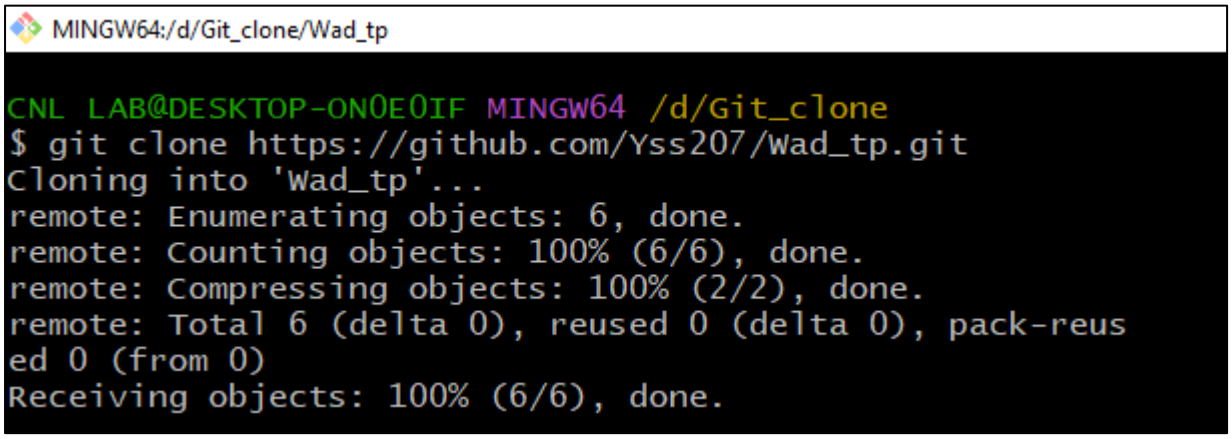

Figure 5: git clone

Step 3: Edit the file on GitHub

1. Open the repository on GitHub and edit the file you added (e.g., example.txt).
2. Save the changes directly on GitHub.

Step 4: Navigate to the Cloned Repository on Your Local Machine

1. Use the cd command to navigate to the folder where the repository was cloned:

```
CNL LAB@DESKTOP-ON0E0IF MINGW64 /d/Git_clone
$ cd Wad_tp/

CNL LAB@DESKTOP-ON0E0IF MINGW64 /d/Git_clone/Wad_tp (mai
n)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reus
ed 0 (from 0)
Unpacking objects: 100% (3/3), 984 bytes | 9.00 KiB/s, d
one.
From https://github.com/Yss207/Wad_tp
   945d7a1..f93a43b  main       -> origin/main
Updating 945d7a1..f93a43b
Fast-forward
 Fun_wad.txt | 2 ++
 1 file changed, 2 insertions(+)
```

Figure 6: git pull

Step 5: Pull the Changes from GitHub

1. Run the git pull command to retrieve the latest changes made on GitHub(*as shown above)*.
2. This will fetch and merge the updated file (e.g., Fun_wad.txt) into your local repository.

♦Git Branch

A branch in Git is a fundamental concept that allows developers to diverge from the main line of development, enabling them to work on new features, bug fixes, or experiments without affecting the stable version of the project.

♦Common commands for branching:

1. List all Branches:
   - **`git branch`**
2. Branch Creation:
   - Done using **`git branch <branch-name>`**
3. Branch Renaming:
   - **`git checkout -m <new-name-of-branch>`**
4. Branch Switching:
   - **`git checkout <branch-name>`**
5. Merging branches:
   - First switch to branch which will be on the receiving end of the merge operation.
   - **`git merge <branch-name-which-is-to-be-merged-with-current>`**
6. Deleting branch:
   - First navigate to another branch.
   - **`git branch -d <branch-to-be-deleted>`**

```
Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git branch
* master

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git branch second-branch

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git branch
* master
  second-branch

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git checkout second-branch
Switched to branch 'second-branch'
```

```
Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (second-branch)
$ git branch
  master
* second-branch

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (second-branch)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git merge second-branch
Updating 1e5b401..4f29402
Fast-forward
 Third.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Third.txt

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git push origin master
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Yss207/Test_repo.git
   1e5b401..4f29402  master -> master
```

```
Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git branch
* master
  second-branch

Yash@LAPTOP-JA7V1P1N MINGW64 /d/WAD_Test (master)
$ git branch -d second-branch
Deleted branch second-branch (was 4f29402).
```

Above figures illustrate the branching process, starting with creating another branch, switching to it, merging changes of the second-branch with the master branch, and finally deleting the second branch.