INTRODUCTION TO GIT AND GITHUB

By

Sakib Chowdury
ID: 2104010202241

CSE306: Software Engineering and Information System Design Lab

Instructor:

MD. Tamim Hossain
Lecturer
Department of Computer Science and Engineering
Premier University

Signature

Department of Computer Science and Engineering
Premier University
Chattogram-4000, Bangladesh
22 November,2023

* Abstract:

the lab report covers the foundations of version control with Git & collaborative software development with GitHub. The main goal is to introduce readers to fundamental principles and practical applications, addressing th need for effective code management in current programming.

The experiment addresses typical code management difficults by offering solutions using simple gits commands and GitHub repositories. the results show successful collaborative coding, with an emphasis on improved organization, version tracking, and overall project efficieney.

* Introduction:

Git is a fundamental version control system used in software development that allows for effective collaboration and code management. It is frequently used in conjunction with platforms like a GitHub, GitLab, and Bitbucket, and it is an essential element of the development workflow. As a key center for collaborative projects, GitHub, a prominent platform, has considerably contributed to the expansion of open-source development. GitHub, which is widely used by enterprises of all kinds, enables effective code hosting and collaboration, making it a cornerstone in current software development techniques.

* Materials:

Git, GitHub, Web Browser, Notepad, Text document etc.

- The Git application is used for version control and source code management is software development.
- The GitHub app is a tool that allows users to interact with GitHub features and services in a more user-friendly way.
- Text documents are a powerful and versatile tool that is well-suited for use on GitHub.

Activities:

1. Create a git repo.

- create a dictionary we want to set as our repository in a location:

```
$ mkdir myFirstRepo
cd myFirstRepo
```

- Initialize the directory as a repository:

```
$ git init
$ git config --global init.defaultBranch main
$ git branch -m main
```

- Use config to add name and email:

```
$ git config --global user.name "MyName"
$ git config --global user.email "MyEmail"
```

- check git name and email:

```
$ git config --global user.name
$ git config --global user.email
```

2. Create a txt script and run it in the terminal.

- Create txt script in my directory:
  myFirstFile.txt

- Add/remove the file to/from the repository:
  $ git status
  $ git add fileName_with_Extension
  $ git add.
  $ git rm --cached fileName

- Invoke git commit: and push existing repository:
  $ git commit -m main
  $ git remote add origin https://github.com/My_Name/repo_name.git
  $ git commit -m "messageHere"
  $ git push -U origin main

3. Change txt script, then add it, commit it, and do git status and git diff.

- Add file to staging
  $ git status
  $ git add myFirstFile.txt
  $ git status

- Commit files in staging
  $ git commit -m "saving Original File"
  $ git log
  $ git status
  $ git diff

- commit it again:

$ git commit -m "added more to text"

$ git log

$ git status

4. Update my local repository.

- update repo:

$ git pull

- fetch & merge repo:

$ git fetch

$ git merge.

Git Cheat Sheet:

1. git init
init (setting up). initialize an existing directory as a git repository.

2. git branch
branch will appear next to the currently active branch.

3. git add.
this will add the specidie file into the git repository.

4. git commit
$ git commit -m "message"
# this command records repo permanently

5. git status
this command will show the modified status of an existing repository.

6. git remote

$ git remote add origin "[URL]"

We can start pushing our code to the remote (central) repository of the project.

7. git push

$ git push origin [branch name]

By using the command 'git push' the local repository's file can be synced with the remote repository on GitHub.

8. git clone

$ git clone [URL]

This will import the files of a project from the remote repository to our local system.

9. git checkout

$ git checkout [NewBranchName]

This command allows us to switch to an existing branch within our repository.

10. git log

This command is handy when we want to examine the detailed log of every commit in our repository.

11. git stash

When we want to save our work without staging or committing the code to our Git repository and want to switch between branches.

12. git revert

$ git revert [commit id]

this command can be considered as an 'undo' command.

13. git diff

$git diff [version-x_commitid][version-y_commitid]

Diffing is a function that takes two input data sets and outputs the changes between them.

14. git merge

$ git merge [anotherFileName]

this command will combine multiple sequences of commits into one unified history.

15. git rebase

$ git rebase [base]

It is the process of moving and combining a sequence of commits to a new base commit.

16. git fetch

To integrate the commits into our master branch, we use the merge feature.

17. git reset

$ git reset -hard [some commit]

to return the entire working tree to the last committed state.

18. git pull

$ git pull origin master

It downloads the content from the specified remote repository and then immediately updated the local repo to match the content.

19. git tag

Tags are used to mark specific points in our project history.

20. git cherry-pick

$ git cherry-pick commit-hash

It is useful for developers to apply a specific commit from one branch to another branch

21. git bisect

It is used for binary search debugging

22. git reflog

It is used to show a log of all git actions that have been performed on a repository.

23. git rerere

Reuse recorded resolution of conflicted mergers.

24. git gc

optimize and clean up git's internal database.

25. git instaweb

start a web-based git interface

Discussion:

In our Git and GitHub lab, we aimed to grasp version control system fundamentals. Analyzing our experiments strengths, weaknesses, comparing results, identifing errors, and interpreting significance reveals insights for refining future studies.

We aimed to introduce participants to basic Git-commands, collaborative workflows using GitHub. Despite the strengths, there were notable weaknesses in our experimental design. the limited time allocated for the experiment restricted the depth to which certain concepts could be explored.

A technical error, a temporary GitHub server outage during the hands-on segment, hindered participatn's push and pull actions, emphasizing the need for contigency plans and alternative platforms in future experiments. A noticable issue was that sometimes files could not be pushed. After deleting the repo in the pc folder, creating a new repo, and commiting and pushing again, then the error no longer appeared.

Future research could explore the impact of Git and GitHub training on indivisuals with varied levels of programming experience and from diverse ~~displ~~ disciplinary background.

* Conclusion:

Finally, our Git & GitHub laboratory experiment gave useful insights into our design's strength and shortcomings, comparisons with comparable trails, identification of experimental mistakes, and interpretation of results. Our findings are significant because of the practical relevance of version control systems and the possibility for increased cooperation in coding projects. As we solve the hilighted flaws and investigate new research areas, we contribute to the ~~count~~ continuing discussion about successful ways for teaching and learning Git and GitHub.

* References:

GitHub Docs, nvie.com, version control with Git, Chacon, Scott, and Ben Straub, pro Git.