INTRODUCTION TO GIT AND GITHUB

By

S M GOLAM SHOBHAN

ID: 2104010202298

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

13 November,2023

Abstract:

Introduction: Git is a distributed version control system that allows multiple developm~~ent~~ pers to collaborate on project efficiently. It tracks changes to code, facilitates branching for parrallel development and ensures version history integrity. GitHub, on the other hand, is a web-based plateform that hosts Git repositories (repo) providing a centralized hub for collaboration issue tracking and code review. Together Git and Github streamline the development process making it easier for teams to work together on software projects.

Materials:

1. PC
2. Internet
3. Server
4. Git hub Link
5. Account (Git hub)

Activity:

Create a git repo.

1. create a directory you want to set as your repository in a location:

```
$ mkdir myFirstRepo
cd myFirstRepo # you should be in ~documents/
                myFirstRepo
               # note: or ~documents/.../my
               FirstRepo if you specified
```

2. Initialize the directory as a repository:

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

③. Use config to add your name and email:

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

④. Create a txt script in your directory:

myFirstFile.txt

⑤ Inside the file, write code to print the text "Hello World!"

```
Print("Hello World!")
```

Activity-2 : change helloWorld.txt and run it on the terminal Then add it, commit it and do git status and git diff.

6) Add file to staging

```
$git status #get used to using this commar
```

as it helps to verify which state the files are in

```
$git add myFirstFile.txt
$ git status.
```

~~7~~. Commit files in staging!

```
$ git commit -m "saving original file"
$ git log #to see what our commit looks like
$ git status
```

8. Change, save, and exit out of the text file.

```
# change the text inside of the file
print ("Hello World! and Stanford!")
```

9. Set autometic command line coloring for Git for easy reviewing

```
$ git config ---global color.ui auto
```

10. Retrieve an entire repository form a hosted location via URL

```
$ git clone [Url]
```

Activity 3: Stage & Snapsort

11. Show modified files in working directory staged for your next commit

```
$ git status.
```

12. unstaging a file while retaining the changes in working diretory

```
$git reset [file]
```

13. Add file as it looks now to your next commit (stage)

```
$ git add [file]
```

14. diff of what is changed but not staged

```
$ git diff
```

<u>Activitys 4: Branch & Merge</u>

15. list your branches. a* will appear next to the currently active branch.

```
$ git branch
```

16. create a new branc at the current commit

```
$ git branch [branch-name]
```

17. switch the specifieed branch's history into the current .

```
$ git checkout.
```

18. merge the speeified brane's history into the current one

```
$git merge [branch]
```

19. Show all commits in the current branch's history

```
$git log
```

Activity 5: Temporary Commits

20. Save modified and staged changes

```
$git stash
```

21. list stack- order of stashed file changes

```
$ git stash list
```

22. write working from top of stash stack

```
$git stash pop
```

23. discard the changes from top of stast stack

```
$ git Stash drop
```

REWRITE HISTORY

24. apply any commit of current branch ahead of specifieed one

```
$git rebase [branch]
```

Discussion: users often face challengs with git and github. such as merge conflicts and the learning curve for commands. To addres merge conflicts, developors should carfully review and resolve differences before mergin branches. Overcoming the learning curve involve thorough understanding of git commands an workflows which can be acquired through tutorials and pratice. Accidental deletions or modifications can be mitigated by using git's version history to restore previous states. Collaborative conflicts are best manage by effective communication among team mem bers to synchronize code changes and avoied simultaneous pushes. Overall, Proacti learning and clear communication are key to resolving common Git and GitHub issue

Conclusion: Git is an invaluable tool that enhances collaboration., version ·control·and project management. Learning git ·empowers programmers to track changes systematically create branches for experimental features or bug fixed.and merge changes seamlessly. It facititates collaboration by providing a centrali ed repository ·allowing multiple ·developers to work on the same codebase without conflic Git's commit history and branching capa- bilities ·offer a safety net, enabling progr- ammers to revert to previous states or explore different approaches without compromising the project, Understanding git not only streamlines individual work- flow but also aligns with industry best praties, making it an essential skill for any programmer