SOFTWARE PROJECT MANAGEMENT

LAB 5

Lecturer: **Nguyễn Minh Tâm**

Email: **tam.nguyen272@hcmut.edu.vn**

Content: Version Control System in Software Project Management

Objectives:

- Get familiar with the Git version control system.

Exercise 1:

Main Task:

1. Create a new directory and use the **init** command to create a Git repository in that directory. Now, you should see a .git directory appear in your directory.
2. Create a **README** file. Using the **status** command, you should see that the README you created should appear as an untracked file.
3. Use the **add** command to add the new file to the staging area. Again, look at the output of the **status** command.
4. Now use the **commit** command to commit the contents of the staging area.
5. Create a src directory and add a couple of files to it.
6. Use the **add** command to stage the directory, not the individual files. Use the **status** command to see how both files have been staged. Commit them.
7. Make a change to one of the files. Use the **diff** command to view the details of the change.
8. Next, **add** the changed file, and notice how it moves to the staging area in the **status** output. Also, observe that the **diff** command you did before using add now gives no output. Why not? What do you have to do to see a **diff** of the things in the staging area?
9. Now - without committing - make another change to the file you changed in step 7. Look at the **status** output and the **diff** output. Notice how you can have both staged and unstaged changes, even when discussing a single file. Observe the difference when you use the **add** command to stage the latest round of changes.

Finally, **commit** them. You should now have started to get a feel for the staging area.

10. Use the **log** command to see all the commits you made so far.
11. Use the **show** command to look at an individual commit. How many characters of the commit identifier can you get away with typing at a minimum?
12. Make a couple more commits, at least one of which should add an extra file.

Extended Task:

1. Use the Git **rm** command to remove a file. Look at the **status** afterwards. Now commit the deletion.
2. Delete another file, but this time do not use Git to do it (e.g. If you are on Linux, use the normal (non-Git) **rm** command; on Windows use **del**).
3. Look at the **status**. Compare it to the status output you had after using the Git built-in **rm** command. Is anything different? After this, commit the deletion.
4. Use the Git **mv** command to move or rename a file, for example, rename **README** to **README.txt**. Look at the status. Commit the change.
5. Now do another rename, but this time use the operating system's command to do so. How does the status look? Will you get the right outcome if you were to **commit** at this point? Work out how to get the **status** to show that it will not lose the file, and then commit. Did Git at any point work out that you had done a rename?
6. Use **git help log** to find out how to get Git to display just the most recent 3 commits. Try it.
7. What does the **--stat** option do on the **diff** command? Find out if this also works with the **show** command. How about the **log** command?
8. How to see a diff that summarizes all that happened between two commit identifiers? (*Hint*: use the **diff** command)

Exercise 2:

Main Task:

1. Run the **status** command. Notice how it tells you what branch you are in.
2. Use the **branch** command to create a new branch.
3. Use the **checkout** command to switch to it.
4. Make a couple of commits in the branch - perhaps adding a new file and/or editing existing ones.
5. Use the **log** command to see the latest commits. The two changes you just made should be at the top of the list.
6. Use the **checkout** command to switch back to the master branch. Run **log** again. Notice your commits do not show up now. Check the files also - they should have their original contents.
7. Use the **checkout** command to switch back to your branch. Use **gitk** to take a look at the commit graph, notice it is linear.
8. **Checkout** the master branch. Use the **merge** command to merge your branch into it. Look for the information about it having been a fast-forward merge. Look at **git log**, and see that there is no merge commit. Take a look in **gitk** and see how the DAG is linear.
9. Switch back to your branch. Make a couple more commits.
10. Switch back to master. Make a commit there, which should edit a different file from the ones you touched in your branch - to be sure there is no conflict.
11. Now **merge** your branch again.
12. Look at **git log**. Notice that there is a merge commit. Also, look in **gitk**. Notice the DAG now shows how things forked, and then were joined up again by a merge commit.

Extended Task:

1. Once again, **checkout** your branch. Make a couple of commits.
2. Return to your master branch. Make a commit there that changes the same line, or lines, as commits in your branch did.

3. Now try to **merge** your branch. You should get a conflict.
4. Open the file(s) that is in conflict. Search for the conflict marker. Edit the file to remove the conflict markers and resolve the conflict.
5. Now try to **commit**. Notice that Git will not allow you to do this when you still have potentially unresolved conflicts. Look at the output of **status** too.
6. Use the **add** command to add the files that you have resolved conflicts into the staging area. Then use **commit** to commit the merge commit.
7. Take a look at **git log** and **gitk**, and make sure things are as you expected.
8. Perform the following sub-tasks if you wish to get a **bonus**…
    - Delete everything but your .git directory, then do a checkout command, to prove to yourself that this really will restore all of your current working copy.
    - Create a situation where one branch has changed a file, but the other branch has deleted it. What happens when you try to merge? How will you resolve it?
    - Look at the help page for merge, and find out how you specify a custom message for the merge commit if it is automatically generated.
    - Look at the help page for merge, and find out how to prevent Git from automatically committing the merge commit it generates, but instead give you a chance to inspect it and merge it yourself.

DEADLINE: **23:55 - 24/11/2024**

Note:

- Students must submit the report in PDF format. Your answer for each step in the task above should be explained in detail within the report and include visual evidence if needed.
- For this laboratory exercise, students must ensure that their submissions are directed to the appropriate lab class.