Session 2B Principles of FOSS and Version Control
Exercise 2

In this exercise, you will practice the basics of Git that you learned in Session 2B. Follow the instructions below. You can work individually or as a group, but either way use your support group to ask questions if you get stuck. Refer back to the lecture slides to help you. Remember :

- if you're using Windows, you'll need to launch the Git Bash program before starting
- if you're using Linux or Mac OS, you should just need to open a new terminal (assuming Git is installed)

Further Note : at the time of writing, Git is still routinely using *master* as the default branch name. If this has changed to *main* when you come to undertake this exercise, please consider all references to *master* to refer to *main*.

1. If this is your first time using Git, you should configure your username and email as per the instructions in the slides.

2. Create a new folder somewhere on your machine and switch to it using terminal commands.

3. Create a new Git repository in this new folder.

4. Write a simple piece of code in Spyder, and save the file in your new folder.

5. Check the status of your repository – you should see that Git is aware of your new file, but is not tracking changes.

6. Stage and commit the new file, with an appropriate commit message. Check the status again both post-stage and post-commit.

7. Check the log to see the details of your commit.

8. Create a new branch called *dev* and switch to it.

9. Add a new function / feature into your code, save it, stage the changes and commit with an appropriate commit message. Remember to check the status before and after staging and commits.

10. Look at the *diff* between the current version and the previous version.

11. Test the code works. Once you're happy it does, merge the changes you've made in the *dev* branch back into *master*. Take a look at the history of the *master* branch.

12. Switch back to the *dev* branch, and merge *master* back to *dev*.

13. Add a third function / feature to your .py file that contains an error.

14. Save, stage and commit the changes.

15. Revert the changes in the latest commit (which contains the error) to the previous version (which doesn't). Remember to use the --oneline option of the log command to more easily access the 7 digit commit identifiers.

16. You should now be back in the code that has two functions / features. Change one or more lines of existing code to work in a different way. Stage and commit the changes.

17. Switch back to the *master* branch. Change the same lines of code that you changed above, but in a different way to how you did before. Stage and commit the changes.

18. Try merging *dev* into *master*. You should find you have a conflict. Resolve the conflict however you choose, then stage and commit the fix.

19. Check the commit history of *master*.

20. Switch back to *dev* and check the commit history.

21. Merge *master* back into *dev*.

22. Create a new .csv file and put some minimal dummy data in it. Save it in your directory.

23. Check the status of your repository. You should see that Git is aware of the new file, but is not currently tracking it.

24. Create a .gitignore file that tells Git to ignore any .csv files in your project. Stage and commit the file to your repository.

25. Check the repository status again. You should see that Git is no longer paying any attention to your .csv file.

26. Create, stage and commit a readme.md file which contains information about your code project.

27. Merge *dev* back into *master*.