CS308: Large Applications Practicum, 2020

Lab 2

Reference for this task: http://www.vogella.com/tutorials/Git/article.html

Sections for reference from the above article are mentioned in parenthesis in the tasks below.

You can skim though Sections 1, and 2. You may also have to do some of the configurations in Section 7 (esp 7.2, 7.6, 7.7, and 7.8). For most other configuration settings, keep the default options.

1 You are provided a small C project that computes the square root. You will now incorporate version control into this project with git. Perform the following sequence of operations in order.

1.1 **Creation.** Extract the provided archive into a new directory.
1.2 Go to the directory proj4/
1.3 Compile and run the program. Verify that it works correctly.
1.4 Create a new git repository inside this directory. Use `git init` (10.2)
1.5 Check the status. Use `git status`.
1.6 Add files to staging area. Use `git add` (10.5)
1.7 Add a comment before the main function.
1.8 Check status.
1.9 You keep repeating the last two steps until you are feel your code is good enough to commit.
1.10 If you are ready, add the changed files into the staging area. Use `git add`.
1.11 Check status. Make sure everything you want is staged.
1.12 Commit staged files into the repository. Give a proper commit message, such as "Initial working code for sqrt." Use `git commit`. (10.7)
1.13 View the history. Use git log. (10.8)
1.14 View the changes of the commit. Use `git show`. (10.9)
1.15 Add a .gitignore file to ignore executables (10.14)
1.16 Commit the .gitignore the file. (10.15)
1.17 Check status.
1.18 **Basic workflow.** Add a new printf statement in the main program stating "End of program. Exiting."
1.19 Check the status. `srqt.c` should appear modified. Make sure there are no bugs and the code compiles correctly.
1.20 Add the modified file to the staging area.
1.21 Commit the new changes. Give a proper commit message, such as "Added exit message."
1.22 Check status.
1.23 View the log.
1.24 Now you are satisfied with this code. Tag it as version 1. (15.1-15.4)
1.25 **Branching.** Now suppose you need to make a major change. Lets add checking if the input is a valid number. But you dont want to break version 1, so we will create a new branch. The default branch is the master branch. Lets call the new branch br_inputCheck.

1.26 List available branches. (13.2)
1.27 Create a new branch. Use `git branch`. (13.3)
1.28 Checkout branch br_inputCheck. (13.4)
1.29 Check the status. It should say you are in the branch br_inputCheck.
1.30 Make changes to this branch. Change the code so as to check if the input is a valid number. Typically the changes needed for this will not conflict the existing version 1 in the master branch. Make your changes accordingly. **We are deliberately avoiding conflicting changes here.**
1.31 Check status.
1.32 When you are ready with the code, add changed files into the staging area.
1.33 Commit the changes. You are still in branch br_inputCheck.
1.34 Check the status and the log. You are still in branch br_inputCheck.
1.35 Now lets merge the br_inputCheck into the master branch. You can see the differences between branches using `git diff`. (13.10)
1.36 Have a look at how git merges. (35)
1.37 Move back to the master branch. Use git checkout master.
1.38 Merge the branch br_inputCheck. (35.4)
1.39 If you have coded as directed above, the merging should not create any conflicts.
1.40 Check the status.
1.41 Check the log. Now the master branch has incorporated the changes from br_inputCheck.
1.42 Tag this as version 2.
1.43 Conflicts. Now suppose you need to make another major change. The current code returns NaN if the input is negative. You want to support negative numbers. The square root of -4 is 2i, where i is the root of -1. But you don't want to screw up version 2. So you create a new branch. Lets call the branch br_negativeSupport. (13.1)
1.44 List available branches. (13.2)
1.45 Create a new branch. Use `git branch`. (13.3)
1.46 Checkout branch br_negativeSupport. (13.4)
1.47 Check the status. It should say you are in the branch br_negativeSupport.
1.48 Make changes to this branch. Change the code so as to check if the input is negative, and accordingly print the correct output.
1.49 Check status.
1.50 When you are ready, add changed files into the staging area.
1.51 Commit the changes. You are still in branch br_negativeSupport.
1.52 Checkout the master branch.
1.53 Merge br_negative support. Based on how you have coded, Git may complain that there is a conflict.
1.54 Use `git mergetool` to view and handle the changes. You will need to change the conflicts manually, and remove the extra lines put by git.
1.55 Once you are satisfied, stage, commit and check the status.
1.56 If you are not getting a conflict, do the following. Change something in the master branch (say a printf statement) and commit. Now edit the same line in br_negativeSupport and commit. Come back to master and try to merge. You should now get a conflict. See the demo video on YouTube.

2 Conflict resolution. Here is another way to force a conflict.

2.1 There are currently two branches: the master branch, and br_negativeSupport.
2.2 You have already merged br_negativeSupport and master. During this, you may or may not have got a conflict.

2.3 Go to branch br_negativeSupport.
2.4 Change every `printf` statement to print the prefix “NEGSUPPORT: ”. Only edit the `printf` lines, do not add anything else.
2.5 Commit the version.
2.6 Create a new branch called br_printDebug.
2.7 In the branch br_printDebug, every `printf` statement will first print the prefix “DEBUG: ”. Only edit the `printf` lines, do not add anything else.
2.8 Once changes have been made, commit and tag the version.
2.9 Go back to master branch.
2.10 Merge br_printDebug with master.
2.11 Merge br_negativeSupport with master.
2.12 This time you should get a conflict. Open the code and note how conflicts are reported in git. You will need to manually remove the extra lines put in by git.

Once you have done all of the above, push your repository into github. Use the same repository that you have shared in the Google form. I must be able to see your entire commit history so make out the steps you have done. Name the repository as **CS308 Sept 2020 Git Lab 1**.
Make sure you give **proper commit messages** so that I can follow what you have done.

I will extend this assignment for more tasks in a day or two.