DEPARTMENT OF EECS

Indian Institute of Technology Bhilai

CS200 — SOFTWARE TOOLS AND TECHNOLOGIES Lab II

Scope: Git Branch

Difficulty Level: Intermediate

Assignment 2

September 30, 2020

- Instructions
  - All answers will be in separate files in a single folder, named: `<group-id>_<group-name>`
  - Name files as `q<question-no>` without any extension. e.g., q2
  - Use LaTeX to show your answers that need `git` graphs
  - Make a `tarball` for the folder that contains your answers
  - Compress the `tarball` using `gzip` before uploading on Piazza

1. The "Three States" in `git` were shown in class. [Warm-up]
   (a) Find out the mechanism by which `git` allows to have multiple working directories.
   (b) List four `git` commands that lead to a directed edge from the staging area (Index) to working directory. Show a single `git` flow example using all the four commands.
   (c) Demonstrate how you can `commit` a single file in parts.
   (d) Now show with an example `git` flow what would you do to turn the changes committed in two different commits into a single commit. Dump the `git` graphs to show the same.

2. (a) Write a `shell` script to recreate the following: [The Merger]
   Initialize `git`. Make some dummy commits on `master` branch and then `checkout` one branch each for each of your group members with branch-name as `<roll-no>`. Make dummy commits in each of the branches including `master` in some random order.
   (b) Now write a `shell` script to find the branch that has the `latest` commit and then merge all other branches to that branch using a loop. Dump the `git` graphs after each merge. Don't show `blobs` and `trees` (Hint: `git-graph` natively supports this). Show the incremental graphs with appropriate comments in LaTeX.
   [Hint: Assume there are no conflicts while merging]

3. Merging and Rebasing were shown to be two ways to combine `git` branches. [Merge Vs Rebase]
   Develop two minimal `git` flows: one to showcase a situation explaining when merge is better than rebase and vice-versa for the other one. Give brief explanations for both and also show the final `git` graphs. Submit `shell` scripts which will allow us to retrace the examples developed.

4. Develop a realistic mini-project collaboratively using `git` branching. [Welcome to reality!]
   (a) First, write a code (anyone from the group) for reading a binary tree from an input file provided by the user. Assume the input file contains the *in-order* traversal of the binary tree.

Question 4 continues on the next page. . .

(b) Commit your code in the “master” branch and create three new branches named “func1”, “func2” and “func3”, each to be edited by a particular group member.

(c) Group members 1, 2 and 3 respectively develop codes to print *pre-order*, *post-order* and *zigzag* traversals (top to bottom) of the input binary tree in their corresponding branches. Commit each branch individually.

(d) Use `git` “merge” to merge all three branches into the “master” branch.

(e) Group Member 1: Develop an I/O code in the “master” branch which reads the user-provided tree, repeatedly asks the user which traversal he/she wants to print and runs the corresponding code already developed.

(f) Group Member 2: Create a new branch “bugFix” and update the code for *zigzag* traversal in such a way that it prints the reverse *zigzag* now (bottom to top). Commit and merge this branch into the “master” branch.

(g) Group Member 3: Create another branch “NewFunc” and write a new code to convert the given binary tree into a binary search tree. Commit and merge this branch into the “master” branch.

(h) Group Member 1: Updates the I/O code such that it also gives users the option to display the aforementioned new functionality developed by Group member 3 (along with the existing functionalities).

(i) Show the `git` graph. (ii) Perform all the merging using `git` “rebase” and show the corresponding `git` graph. (iii) Is it possible for Group member 1 to keep both the *zigzag* orders (top to bottom and bottom to top) developed by Group member 3 as options in his/her final I/O code? If yes, how? If no, why?

P.S. Use any language for writing the codes.