Laboratory #10: Git practice

Unix (420-321-VA) - Fall 2021
Teacher: Tassia Camoes Araujo

Goals:
1. Review git commands
2. Start collaborating through git
3. Simulate conflicts & merges

Instructions

Part I: Git review

1. Review git essential commands presentend in Lecture #11, or many of the available tutorials online.
2. Create a git repository and add <u>at least</u> one colleague as collaborator
3. Each developer should clone the repository
4. Now, one developer at a time, will:
   a) Create or edit an existing file. Make sure to add at least 10 lines of content, so the exercise is effective
   b) Add the file to the local git repository (*$ git add filename*), commit (*$ git commit -m "log message"*) and push (*$ git push*)
   c) Check online that your changes were uploaded successfully
   d) Tell the rest of the team that your changes are up for them to retrieve (*$ git pull*).
   e) Call the next member in the team to do steps (a) to (d)

Part II: Conflicts playground

After all the team is able to write changes and retrieve other members' changes, we'll go to the conflict practice. A conflict will occur if you try to push to the remote repository, but your changes are based in an older version than what is currently online. If you are working by yourself, you can still perform this practice by cloning the repository in different locations in your filesystem. You will have 2 different local repositories, that will behave just as if they were two clones by different users, so conflicts between the local repositories will occur in the same way.

For instance, suppose the code online is on version 1.0. Everyone in the team has the same version in their local repository. Now two members of the team start making local changes at the same time, without communicating with the other. When the first one pushes to the repository it will work fine, since the next version (1.1) was created based on the 1.0 which is online. Now, when the second person tries to push (another 1.1 version), based on the old 1.0, there will be a conflict with the current version already online (also 1.1) – note that version numbers here are merely illustrative.

As a rule of thumb, git can only push on top of a version that is on the history of versions of the code. The code uploaded by another developer is totally foreign to your own local repository. Since it does not recognizes it as part of its history, the operation is not completed, a conflict occurs. You will be told to do a git pull to integrate the remote changes to your local code. If git is able to automatically merge, that is easy, if not, you'll need to manually edit and commit files.

Try to reproduce the situation described above. Make changes to the same file at the same time that another developer is also changing the file:

1. The first developer will commit and push changes to the remote repository.
2. The second developer will also commit and try to push, but the push will fail. Record the error message you get and add it to your report.
3. The second developer should try to integrate changes from the remote repository with the command "git pull". Since changes were made into the same file, the automatic merge fails with a message "CONFLICT (content): Merge conflict".
4. The second developer will edit the file and remove all lines that start with "<<<<<<<", "=======" or ">>>>>>>", leaving just the pertinent text content. It is the developer's choice of what to remove or keep. The result should be a clean file, ready to be uploaded to the remote server.
5. After cleaning the file, another commit is needed. Please report in your log message that you have fixed a merge. Run the command "git push" and this time is should work successfully.
6. The first developer should update the local repository with "git pull" and check the final result of the merged file.
7. Repeat the procedure once again, but <u>this time each developer should change a different file</u>.
8. The first developer should commit and push. Then the second developer should commit and push. <u>Does the push work?</u>
9. When you run the "git pull" command this time, <u>what happens?</u>

This process of committing, pushing, pulling and merging will happen many times in the development process. The more you practice, the better you'll be at avoiding and solving conflicts.

A good practice in collaborative development is to perform small and frequent commits and pushes. You should not work in different tasks in a row without sharing your code with your team, or risks are high that someone else will have made a change when you try to push.

Read this short article on that topic: https://www.worklytics.co/commit-early-push-often/

Part III: Final project brainstorming

For the final project of this course, students will choose a scenario to which a GNU/Linux system is a viable solution, such as purpose-specific desktop systems, online servers (e.g. web, cloud, streaming), and single-board computers (SBC) such as Raspberry Pi (e.g., media center, security camera, sensor control system). Scenarios must integrate the creation of multiple user accounts, process or service management, system security and automated tasks using a script language.

Based on the chosen scenario, students should identify at least 2 solutions (different GNU/Linux distributions or major software setup) to install and identify pros and cons of each, for the specific case at hands. Finally, one of the solutions should be chosen for final configuration to respond to the identified needs.

Use the rest of the time in the lab to research about final project ideas, talk to your colleagues, try to form a team with people interested by the same idea. Teams should be composed by 2 or 3 students.

Part IV: Deliverables

1. Include a header containing the course name, section, semester, and student name, and license for your report.
2. Include the record of your practice of conflicts and merges. You can add screenshots or simply the output texts that you get in your terminal.
3. Answer questions II-8 and II-9.
4. Include a record of your brainstorming done in part III.
5. Export your file as PDF and upload it to Omnivox.

Good luck!