Lab 3: Version Control

Nathan Jarus

June 14, 2017

Introduction

This lab will get you familiar with the basics of git. You will need to sign in to https://git-classes.mst.edu and clone the repository for this lab. Your repository will be named something along the lines of `2017SS-A-lab03-nmjxv3`. Make sure to clone with the HTTPS URL (unless you've set up SSH keys).

Scenario

Your friend made a git repository for the filter program from Lab 2. They want to collaborate with you to add some features to it.

Problem 1: `.gitignorance` is bliss

Remember how you shouldn't commit generated files to a repository? Well, your friend managed to add `a.out` anyway. Oops.

1. Use `git rm` to delete `a.out` from the repo.
2. Check `git status` to see what this change looks like.
3. Make a `.gitignore` that ignores `a.out`.
4. Add your `.gitignore` to the repository and commit.

Note that your commit in step 4 should commit two changes:

- Remove `a.out`
- Add `.gitignore`

Problem 2: Peace in the repository

Your friend has added another feature: ignoring whitespace at the start of lines. Now, your program can filter lines that contain whitespace (tabs and spaces) before a `#`. They developed this feature on a branch named `whitespace`.

Because your friend is imaginary, you'll have to help them merge their feature into `master`:

1. Use `git checkout` and take a look at your friend's code. (You need to `checkout` a remote branch the first time you do anything with it.)
2. Use `git diff master` to see what they changed. Looks pretty OK, eh? Let's merge it in!
3. Check out the `master` branch.
4. Run `git merge whitespace` to merge your friend's branch into `master`.
5. Oh no! There's a merge conflict! Use `git status` to see which file the conflict is in, then edit that file and fix the problem.
6. Use `git add` to stage your resolution to the conflict and `git commit` to complete the merge.

Problem 3: Features grow on branches

Okay, so now you've got the repository cleaned up! Let's add a feature to this code. Instead of hardcoding the comment character as `#`, we want to have it take a command line argument that specifies the comment character.

So, for example, you could run `cat story.txt | filter %` to filter lines that start with a %.

1. Make a new branch to develop your feature on.
2. Modify `filter.cpp` to support your new feature, add it to the repository, and commit. (Too easy? Try using `getopt()`!)
3. Push your branch to the remote repository. Check Gitlab to make sure it's up there.
   The first time you `push` a new branch, you have to tell git to make a new branch on the remote. `git push` will tell you the right command to run.

Now, you'd like to merge your feature into `master` too!

1. Okay so merge it already.
2. Don't forget to push after you've merged!

Problem 4: Paradox-free time travel

Your friend realized that they accidentally deleted the header comments from `filter.cpp`!

1. Figure out which commit they deleted the comments in.
2. Use `git revert` to undo that mistake!
3. Make sure to push your revert to the remote repo.

Epilogue: Submitting

Your repo on gitlab is your submission, so whatever is up there is what we will grade. Make sure you've pushed both your feature branch and master. You can double-check on the Gitlab website to make sure your submission looks the way you want it to.

We expect to see the following files on the master branch:

- `README.md`
- `.gitignore`
- `filter.cpp`

Hints

- Use `git help <command>` to learn more about a command.
  For example, `git help commit` to learn about `git commit`
- Use GitLab's Network view (under the Repository tab) to see a pretty graph of your commit history. Youll also see your branch names and their corresponding commits there.