Git exercies

Kiran Vasudev

September 11, 2019

1 Exercise 1

1. **Create a new directory** and change into it.
2. Use the **init** command to create a Git repository in that directory.
3. Observe that there is now a .git directory.
4. Create a README file.
5. Look at the output of the **status** command; the README you created should appear as an untracked file.
6. Use the **add** command to add the new file to the staging area. Again, look at the output of the **status** command.
7. Now use the **commit** command to commit the contents of the staging area.
8. **Create a src** directory and add a couple of files to it.
9. Use the **add** command, add the directory itself, not the individual files. Use the status command. See how both files have been staged. Commit them.
10. Change contents of one of the files. Use the **diff** command to view the details of the change.
11. Next, **add** the changed file, and notice how it moves to the staging area in the status output. Also observe that the diff command you did before using add now gives no output.
12. Why not? What do you have to do to see a diff of the things in the staging area?
13. Now without committing make another change to the same file you changed in step 10.
14. Look at the status output, and the diff output. Notice how you can have both staged and unstaged changes, even when youre talking about a single file.
15. Observe the difference when you use the add command to stage the latest round of changes. Finally, commit them.
16. You should now have started to get a feel for the staging area.
17. Use the **log** command in order to see all of the commits you made so far.
18. Use the **show** command to look at an individual commit. How many characters of the commit identifier can you get away with typing at a minimum?
19. Make a couple more commits, at least one of which should add an extra file.

2 Exercise 2

1. Run the **status** command. Notice how it tells you what branch you are in.
2. Use the **branch** command to create a new branch named *introduction-script*
3. Create another branch called *introduce-your_name* from *introduction-script*
4. Create a file named INTRODUCTION and add the following details
   - Name
   - Bachelors degree
   - A small description of the person you are
   - Favorite movie/hobby/passtime
5. Commit and merge into *introduction-script* branch.
6. Create a tag $v1.0$
7. Visualize your changes using *gitg*

Next steps:

1. Checkout your branch. Make a couple of commits.
2. Return to your master branch. Make a commit there that changes the exact same line, or lines, as commits in your branch did.
3. Now try to merge your branch. You should get a conflict.
4. Open the file(s) that is in conflict. Search for the conflict marker. Edit the file to remove the conflict markers and resolve the conflict.
5. Now try to commit. Notice that Git will not allow you to do this when you still have potentially unresolved conflicts. Look at the output of status too.
6. Use the add command to add the files that you have resolved conflicts in to the staging area. Then use commit to commit the merge commit.
7. Take a look at git log and gitg, and make sure things are as you expected.

3 Exercise 3 [Team exercise]

1. First, one person in the group should create a public repository using their GitHub account.
2. This same person should then follow the instructions from GitHub to add a remote, and then push their repository(containing a README file with the team members' names).
3. All of the other members of the group should then fork the repository and clone their forked repository.
4. One of the group members(other than the creator of the public repository) should now make a local commit(on another branch), then push the branch to their forked repo.
5. Make a pull request to the public repository.
6. After the owner accepts the pull request, notice the changes happen to the repository.
7. Now the rest of the members need to update their local repository. For this, there are two ways:
   - Using git fetch
   - Using git pull

   Use git fetch to retrieve the updated repository first. Merge the changes to the master. Follow steps 4-7 but this time use git pull. What is different?

Next steps:

1. Now create a situation where two group members both edit the same line in the same file and commit it locally. Race to push.
2. When the runner-up does a pull, they should get a merge conflict.
3. Look as a group at the file in conflict, and resolve it.
4. Use the add command to stage the fix, and then use commit to make the merge commit.
5. Notice how this procedure is exactly the one you got used to when resolving conflicts in branches.