CS 520 In class exercise 1 answers

By Patrick Walsh, Matthew Lips

Link to our repository: https://github.com/pw42020/ie1-basic-stats

---

Questions

Question 1

1. In the basic-stats folder in the main (previously called master) branch, how many commits does the README.md file have?

After checking out the history of the README.md file on GitHub, it is clear that there are 16 total commits in main/master for the README.md file.

2. For the README.md file, what is the set of the different authors of its commits?

The different authors for the commits, also found by looking in the same place above, are "brunyuriy", "rjust", and "DeveloperTommy"

3. For the README.md file, what is the hash of the commit that mentions the ant build tool?

From the same history link on GitHub, the "added more info about Ant" hash can be found to be `c2111cc0d37bfde779a317e533d3a5e68b8ed9e3`.

4. How many total commits have been made in the repository across all branches?

To add all of the commits that have been made in the repository across all branches, viewing the branches commit section on GitHub, we can see all commits on all branches other than main, and how many branches they are ahead of main as well.

Main has a total of 53 commits, and we can also add any amount of commits each branch is ahead of main as well. `v5.0.0` is ahead of main by 30 commits, `v4.0.0` is ahead of main by 24 commits, `v3.0.0` is ahead of main by 13 commits, `v2.0.0` is ahead of main by 5 commits, and `feature-branch` is ahead of main by 16 commits.

$$53 + 30 + 24 + 13 + 5 + 16 = 141$$

total commits.

5. List the files modified in the commit with the hash `01da475`.

By taking a look at the commit page on GitHub for the hash `01da475`, we find that

- src/Models/Model.java
- src/Models/Numbers.java
- src/Views/AddNumView.java
- src/Views/MeanView.java
- src/Views/MedianView.java
- src/Views/ModeView.java

- src/Views/NumbersView.java
- and src/Views/ResetView.java

were all updated in this commit.

6. What is the most recent commit in the main branch?

The most recent commit in the main branch is hash `da90e878188c6de8870581bdb447299821d7e87b` that is labelled "Updated README.md" on October 31st, 2017.

Question 2

> Detail the comprehensive procedure for adding a file named MinMaxCalculation.java to a remote Git repository. Your answer should cover the following key areas:
>
> - Commands executed on your local machine to initiate the process.
> - How to manage merges, particularly in the context of collaborative development.
> - Steps for pushing the changes to the remote repository.
> - Techniques for identifying the specific commit related to this operation.
>
> In your explanation, please integrate best practices such as appropriate naming conventions for branches and commits, as well as the use of .gitignore to exclude files that should not be versioned.

For adding `MinMaxCalculation.java` to a remote git repository, there were several steps I made to ensure that the file would be added and ensure that no files in remote would be overwritten.

Before writing code, I used the commands `git fetch` and `git merge` to merge my code from the remote repository.

This is also to manage merging from my partner into the master branch, in the case that the remote repository is now ahead in commits than my local branch.

After doing this, all of the files on my program are updated, so I wrote my code. The code is visible in `src/`.

After all of the code for `MinMaxCalculation.java` was completed, I then wrote `git fetch` and `git merge` again to ensure I would not conflict with anything my partner created on the `master branch`. Once this was complete, I (in the root directory) wrote `git add src/MinMaxCalculation.java` to add the file to the git commit, and then wrote `git commit -m "complete MinMaxCalculation.java with findMin() and findMax() implemented"` to create the commit. After this, I simply wrote `git push` to push all of my recent commits to the remote repository.

Some techniques you can use to identify this commit is through the name of the commit, which is the message in `git commit -m "<message>"`, but two git commits can have the same name. Instead, the best way to tell specific commits is through the hash received upon committing. Upon completing your commit message, the console will print out

```
[master <hash>] <message>
 X file changed, Y insertions(+), Z deletions(-)
```

The full hash is not received, but this end of the hash is unique enough where you will be able to identify your specific commit.

Question 3

> How many commits did you cherry-pick? Are the commit hashes of the cherry-picked commits identical in main and feature-branch? Briefly explain why.

I cherry picked 16 commits that I found were related to the README.md file. The commit hashes of the cherry-picked commits are not identical in main and feature-branch because in the act of cherry picking you are copying the changes of the commits which designate a new 'prime' commit to the branch, not necessarily using the exact same commit.

Question 4

> What happens if you merge a branch from which you previously cherry-picked single commits? How often do the cherry-picked commits appear in the history? Briefly explain why.

When you merge the branch the cherry-picked commits are a 'prime' of the original commit, so it is completely independent of the original commit and treated as its own commit in the history as such. So the cherry-picked commit is shown and then the cherry-picked 'prime' commit is shown in the history, so the commit appears twice.

Question 5

> What are the risks of rebasing? Mention at least two risks. Briefly describe a use case in which rebasing can be safely applied.

Two risks of rebasing are losing commits due to rebasing too often and possibly destroying code if you rebase incorrectly.

A use case in which rebasing can be safely applied is when you are working on your own local branch that other users do not use, specifically for rebasing commits together to make sure every commit is meaningful.

Question 6

> What are the risks of using reset when a commit has already been pushed?

Using reset can have cascading effects to the commit history in a project. The reset process orphans commits after the reset point. This causes commits to be disconnected from the project, so contributors could be branching from this disconnected portion that will have problems merging back to the main project. Resetting to a commit also has potential for work to be lost.

Question 7

> Does revert remove the reverted commit? Briefly explain how revert works.

No, git does not remove the reverted commit. Revert applies a change in the commit that is the opposite of the reverted content, so the project goes back to its original state, and the overwritten commit is still shown in the commit history. As an example, if commits were numbers then by reverting the following: 1 + 2 + 3, we

get 1 + 2 + 3 - 5 instead of simply 1. The original commit is 1, and in terms of content we revert back to the content, but we still see the full history of commits as 1 + 2 + 3 - 2.