Exercises and answers, Git Basics

Questions

Q1: Why do we use version control systems?

Q2: Let's say you have created a new project and started writing code. Which git commands should you write in which order?

Q3: Why are commit messages important?

Q4: Why do we use branches?

Q5: Which git commands lets you see the changes you have done to the file compared to the last commit?

Q6: What is the main branch called?

Q7: What is the git command to use when you want to go back to a specific commit?

Q8: What is the git command to use when you want to merge branches together? What happens if there is an error?

Answers

A1: VCS are tools used to track changes to source code or other collection of files and/or folders/directories.

A2: First, we initialize git by using `git init`. Then we add the files we want to put in our repo. We can either do `git add .` or write filenames instead of `.`. Then, we commit those files with a message `git commit -m "Commit Message"`.

A3: There may come a time that you would want to check how you implemented your code. Commit messages will make it easy to go back to the versions you like. Also, it is better if someone else is maintaining your code. In addition, in a group setting, others would also understand why that commit is made.

A4: There may be possible reasons. One of them can be that multiple people can work on same code. Then, everyone creating their own branch would be the best for them and the team. So it allows multiple users to work in parallel on

the same code. In addition, sometimes you don't want to risk the tested code, so you don't edit anything into it and work on branches instead.

A5: *git diff* command. This will give you the changes done after the last commit. Also it can get arguments and show you the difference between other commits as well.

A6: Master branch.

A7: *git checkout*. We can use the hash value (or some part of it) to go back to the specific commit. We can also use the branch names as parameters such as `git checkout master`. Therefore, after using *git log* we can use the hash value to go back to that time and the code will be the old version.

A8: The command is *git merge*. You use it as follows: `git merge branch_name branch_name`. If there is a problem, git will tell you that there is a *merge conflict*. It will edit the code but will put signs on it so that you can edit and change them. If the branch changes some code from before or modifies it, git will understand and will not merge.