Git

1. **Explain, how to resolve a conflict in Git?**

When the same resource has been pulled from the original branch by 2 or more diferente people, Git cannot merge automatically the branches. It could happens to one or more files/resources.

But there is a mechanism to solve the conflict: identify the conflict file, and you can choose if you want to keep the other changes or yours. To know where the conflict resides, you should open the conflict file and check for annotations marks as “<<<<<<<HEAD”, “==========” and “>>>>>>> <branch-name>”

So open the file with an editor (as vim) and update the file to keep the changes are OK. Then run “git add <conflict-updated-file>” to solve the conflict. Commit the updated file and the conflict should be resolved.

Note: there is graphical interface for solving the conflicts, instead to use a command line editor.

2. **How do you revert a commit that has already been pushed and made public?**

I assume that the commit has any type of error and you want to solve it.

You could revert the commit, using “git revert <commit-to-be-reverted>”. The branch will be in the same status before the commit.

But in a practical scenario, as it has been published, in my opionion, it is better to make a new commit solving the error. This is what I have done when I have this issue. I prefer this approach because the commit history is not changed for anyone has pulled the branch changes, and the error is being reflected in the branch history as another commit.

3. **Explain the difference between git pull and git fetch?**

Git pull is a combination of a git merge and a git fetch.

When you fetch a branch, the changes are stored in a new branch in your local repository, they are not automatically merged in the local branch. But when you pull a branche, the changes are included automatically in your local branch. So a merge is needed after the fetch for getting the same result as pull does.

4. **What is git stash?**

When you have not commited changes and you want to switch to a different branch, Git advices you about the changes are not commited, so you can “stash” them. Git stores those changes and you can come back over them in the future

5. **What is git fork? What is the difference between fork, branch and clone?**

A Git fork happens when a programmer decides to continue the project in a different way than the official one. So the repository is copied at a moment of the time.

I mean, you can fork any project, and the history of both projects (the original and the forked) not become equal anymore.

So you can introduce features that the “benevolent dictator for life” (or the board of the project) consideres are not useful for the project. So your branch starts to be different to the original one.

If a fork branch wants to be merged with the original branch, as the changes were different in the files/resources, confilcts will be raised, and it may be a hard work to solve those conflicts.

For instance, the original Apache Flink project had a fork that it was done by Alibaba (a chinese company). Months after the fork, Alibaba wanted to merge its changes in the Flink project, so both projects have to work alongide for solving the conflicts and introduce the features and functionality that Alibaba included in the software.

A branch is as a diferent version of the whole project. In git, you use different branchs to keep a stable state of the code in a branch, and make changes in a different branch. When the changes were tested in a branch, you can merge with the stable one, and go on.

Git clone is a copy of the project. But it is different to a fork, because you dont want to make changes to be not commited to the original branches. So you clone the project, you will have a copy of the state of the project in your local filestystem. So you can create a new branch for doing some changes and merge them in your local branches before to push the changes to the repository is online and distributed to the rest of components of the team.

6. **What is the difference between rebasing and merge in Git?**

They are very similar, as the result of both commands will be a unified branch. But in a different fashion, because the commits history will be different if you use “merge” or “rebase”.

When you have make changes in a branch and those changes are not included in the original branch, you have a fork branch (the features between the original branch and your branch are different). So, merging the branches will become a unified branch. But the commits were done in your fork are still include in the commit history.

When you do a rebase, the commit history from both branches are not kept. So you have a unified branch with only part of the historical commits.

7. **How do you squash the last N commits into a single commit?**

In this case, you have to choose with is the first commit to be stash. So you can use HEAD- and the number of commits to move on. In this case, as you want to stash N commits, you have to move HEAD-N in the history for getting the commit you want to start to stash.

So as you see in the previous question, to remove part of the commit history we use the “rebase” command.

So “git rebase –interactive HEAD-N” for getting remove the commits between the first one and the final commit you want to keep.