ASSIGNMENT

1. What is GIT and its significance in SDLC?

Git is a version control system for tracking changes in files and coordinating work on those files among multiple people. It is primarily used for source code management in software development. It is a distributed revision control system and is very useful to support software development workflows. The Git directory on every machine is a full repository which has full version tracking capabilities and independent of network access. You can maintain branches, perform merges, and continue with development even when you are not connected to the network.

2. What is the difference between GIT and SVN?

Git and SVN are both enterprise version control systems (VCS) that help with workflow and project management in coding, they do have their differences. The difference between Git and SVN version control systems is that Git is a distributed version control system, whereas SVN is a centralized version control system. Git uses multiple repositories including a centralized repository and server, as well as some local repositories. SVN does not have a centralized repository or server.

3. What are the advantages of using GIT?

Git has a staging area. This just means that if you made 100 new changes to your code, you can break these 100 changes into 10 or 20 or more commits each with their own comments and their own detailed explanation of what just happened! Not only can you stage your commits out to logically display what changes were made, but you can also do patch staging that ask you if you want. You would use patch staging if you and a co-worker are both working on the same file and you only want to commit a particular function that you’ve worked on. You do a Git patch using “git add -p”

4. What is “Staging Area” or “Index” in GIT?

One of the most essential concepts to Git is that of the staging area. Its use can fundamentally change how you work, for the better. With most other version control systems, there’s 2 places to store data: your working copy (the folders/files that you’re currently using) and the datastore (where the version control decides how to pack and store your changes). In Git there’s a third option: the staging area (or index).

5. What is GIT stash?

GIT stash temporarily shelves changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on. Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

6. What is the function of git clone?

GIT clone is a Git command line utility which is used to target an existing repository and create a clone or copy of the target repository. It Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository (visible using git branch --remotes), and creates and checks out an initial branch that is forked from the cloned repository’s currently active branch.

After the clone, a plain git fetch without arguments will update all the remote-tracking branches, and a git pull without arguments will in addition merge the remote master branch into the current master branch, if any (this is untrue when "--single-branch" is given; see below). This default configuration is achieved by creating references to the remote branch heads under refs/remotes/origin and by initializing remote.origin.url and remote.origin.fetch configuration variables.

7. How can you create a repository in Git?

We can create a new repository from scratch in Git by following below steps.

* Create a directory to contain the project.
* Go into the new directory.
* Type git init.
* Write some code.
* Type git add to add the files (see the typical use page).
* Type git commit.

The first file to create (and add and commit) is probably a ReadMe file, either as plain text or with Markdown, describing the project.

Markdown allows you to add a bit of text markup, like hyperlinks, bold/italics, or to indicate code with a monospace font. Markdown is easily converted to html for viewing in a web browser, and GitHub will do this for you automatically.

8. What is the purpose of branching in GIT?

Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes. This makes it harder for unstable code to get merged into the main code base, and it gives you the chance to clean up your future's history before merging it into the main branch.

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand-new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

The git branch command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the git checkout and git merge commands.

9. What is the difference between ‘git remote’ and ‘git clone’?

A remote in Git is a common repository that all team members use to exchange their changes. In most cases, such a remote repository is stored on a code hosting service like GitHub or on an internal server. In contrast to a local repository, a remote typically does not provide a file tree of the project's current state. Instead, it only consists of the. git versioning data.

Whereas, GIT clone is a Git command line utility which is used to target an existing repository and create a clone or copy of the target repository.

10. What is the function of ‘git diff’ in git?

Diffing is a function that takes two input data sets and outputs the changes between them. git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more. This document will discuss common invocations of git diff and diffing work flow patterns. The git diff command is often used along with git status and git log to analyse the current state of a Git repo.

11. Explain what the commit message is?

A git commit is a change (or “patch”) to code. A commit message is attached to that change — not the code itself. Accordingly, when you write a commit message you are writing it as if it's about to be applied, rather than about what you just did.

12. Why is it advisable to create an additional commit rather than amending an existing commit?

Git amend internally creates a new commit and replaces the old commit. If commits have already been pushed to central repository, it should not be used to modify the previous commits. It should be generally used for only amending the git comment.

13. What is Rebasing?

The rebase command integrates changes from one branch into another. It is an alternative to the better known "merge" command. Most visibly, rebase differs from merge by rewriting the commit history in order to produce a straight, linear succession of commits. Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow. From a content perspective, rebasing is changing the base of your branch from one commit to another making it appear as if you'd created your branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base. It's very important to understand that even though the branch looks the same, it's composed of entirely new commits. The primary reason for rebasing is to maintain a linear project history. For example, consider a situation where the master branch has progressed since you started working on a feature branch. You want to get the latest updates to the master branch in your feature branch, but you want to keep your branch's history clean, so it appears as if you've been working off the latest master branch. This gives the later benefit of a clean merge of your feature branch back into the master branch. Why do we want to maintain a "clean history"? The benefits of having a clean history become tangible when performing Git operations to investigate the introduction of a regression.