**Git Interview Questions**

**1.what is the difference between centralized VC and distributed VC?**

**\*Centralized Version Control**

**a.** In CVS, a client need to get local copy of source from server.

**b.** CVS system does not provide offline access.

**c.** CVS is slower as every command need to communicate with server.

**d.** If CVS Server is down, developers cannot work.

**e.** Relatively it is easier to learn and use

**EX:** SVN(subversion),perforce

\***Distributed Version Control**

**a.** In DVS, each client can have a local branch as well and have a complete history on it**.**

**b.** DVD systems are workable offline as a client copies the entire repository on their local machine.

**c.** DVS is faster response times as majority of operations are performed locally.

**d.** If DVS server is down, developer can work using their local copies.

**e.** DVS systems are difficult for beginners. Multiple commands needs to be remembered.

**EX:** Git,Mercurial.

**2. What is git and what is git-hub?**

**\*Git:**

Git is distributed version control system which is used to track the changes made in source code during any software development.

1. **Distributed Version Control:** Each developer has a complete local copy of the project's entire history allowing them to work independently and merge changes later.
2. **Branching and Merging:** Git makes branching and merging operations efficient, enabling developers to work on separate features or bug fixes in parallel and then integrate their changes seamlessly.
3. **Speed and Performance:** Git is known for its speed and efficiency enabling fast operations even with large codebases.
4. **Flexibility:** Git is versatile and can be used for projects of any size and type, not limited to software development.

**\*Git Hub:**

GitHub is a online web-based Git repository hosting service, which offers all of the distributed revision control and source code management functionality of Git as well as adding its own features.

1. **Repository Hosting:** GitHub hosts Git repositories, making it easy for developers to share and collaborate on code.
2. **Collaboration Tools:** GitHub provides tools for issue tracking, code review, and project management, facilitating collaboration among team members.
3. **Pull Requests:** Developers can submit changes to a repository through pull requests, allowing others to review the changes before merging them into the main codebase.

**3.What is the difference between Git and Git Hub?**

|  |  |
| --- | --- |
| **Git** | **Git Hub** |
| Git is Software. | Git hub is a online service |
| Git is command line tool. | Git hub is graphical user interface |
| Git is installed locally on the System. | Git hub is hosted on the web |
| Git is maintained by Linux. | Git hub is maintained by Microsoft. |
| Git is focused on version control and code sharing | GitHub is focused on centralized source code hosting. |
| Git is open source and licensed | GitHub includes a free-tier and pay-for-use tier. |
| Git was released in 2005 | GitHub was launched in 2008. |

**4.what is the difference between fetch and pull?**

|  |  |
| --- | --- |
| **Fetch** | **Pull** |
| A git command that pulls down the code from the remote repository to tracking branches in the local repository. | A git command that downloads the latest changes from the remote repository and automatically merge those changes in the local repository. |
| Retrieves the changes made in the remote repository without merging them. | Downloads the changes in the remote repository and merge those changes and store them on the local repository. |
| Git fetch is simple command. | Git pull is similar to git fetch followed by git merge. |
| It allows the developers to know the commit pushed by the other developer before integrating with the local repository. | Allows the developers to bring changes to the local code repository and to update it. |

**5.What is the difference between Revert and Reset?**

**\*Revert:**

**a.** git revert creates a new commit with the changes that are rolled back.

**b.** git revert requires the id of the commit you want to remove keeping it into your history.

**c.** The syntax for git revert is generally: git revert [commit].

**d.** Use Git revert when you want to undo changes in a way that is safe for shared branches and preserves the commit history.

**Ex: git revert abcdef**

**\*Reset:**

**a.** git reset erases your git history instead of making a new commit.

**b.** git reset requires the commit you want to keep, and will consequentially remove anything after that from history.

**c**. Git reset is a more powerful and potentially risky command. It is used to reset the current branch to a specific commit, and it has different modes (soft, mixed, and hard).

**d.(1)**-Soft- Leaves your changes in the working directory and staging area.

**(2)-**Hard- Discards changes in both the working directory and staging area, effectively resetting everything to the specified commit.

**e.** The syntax for git reset is generally: git reset [mode] [commit].

Ex. **git reset abcdef.**

**6.What is the difference between Branch and Tag?**

**\*Branch**

**a.** Purpose:

A branch in Git represents an independent line of development. You can create branches to work on new features, bug fixes, or any other changes without affecting the main or other branches.

**b.** Dynamic:

Branches are dynamic and can be created, deleted, and merged. You can switch between branches to work on different features or isolate different streams of development.

**c.** Commits:

Each commit in Git belongs to a specific branch. When you create a new commit, it is added to the currently checked-out branch.

**d.** Usage:

Developers commonly create branches for new features or bug fixes, allowing them to work on these changes independently before merging them back into the main branch.

**Example Commands:**

**1.**Create a branch -git branch [branch name]

**2.**Switch to the new branch-git check out [branch name]

**\*Tag:**

**a.** Purpose:

A tag is a static reference to a specific commit in Git. It is used to mark a particular point in history, typically to indicate a version or a release of the software.

**b.** Static:

Tags are static and immutable. Once you create a tag, it remains fixed to the specific commit it references.

**c.** Commits:

Tags point to a specific commit, and they are often used to mark important milestones in development, such as a release point.

1. Usage:

Tags are commonly used for versioning. For example, you might create a tag for version 1.0 of your software to mark the state of the code at the time of release.

**Example command**:

Create a tag -git tag

git tag tag-name

git show tag-name