**Q) What is a git repository?**

A repository is a file structure where git stores all the project-based files. Git can either stores the files on the local or the remote repository.

**Q) Can you explain head in terms of git and also tell the number of heads that can be present in a repository?**

A head is nothing but a reference to the last commit object of a branch.

For every repository, there will always be a default head referred to as “master” or now “main” (as per GitHub) but there is no restriction to the count of heads available. In other words, it can have any number of heads.

Usages:  
  
- To go or checkout to 1 commit before the latest commit, we use git checkout HEAD~1  
  
- To uncommit the last 3 commits without losing the changes, we first run git reset HEAD~3. Then we can see the changes made in the last 3 commits and then update it manually and commit it finally.  
  
- In order to uncommit the last 3 commits and also remove the changes, we can run the command: git reset --hard HEAD~3. This command will completely remove all the changes.  
  
- To look into the changes made in the last 3 commits, we can run git diff HEAD~3  
  
- To make a new commit by reverting the last 3 commits, we can run the command: git revert --no-commit HEAD~3...HEAD

**Q) What is a conflict?**

Git usually handles feature merges automatically but sometimes while working in a team environment, there might be cases of conflicts such as:  
  
1. When two separate branches have changes to the same line in a file  
2. A file is deleted in one branch but has been modified in the other.

These conflicts have to be solved manually after discussion with the team as git will not be able to predict what and whose changes have to be given precedence.

**Q) What is the functionality of git ls-tree?**

This command returns a tree object representation of the current repository along with the mode and the name of each item and the SHA-1 value of the blob.

**Q) Define “Index”.**

Before making commits to the changes done, the developer is given provision to format and review the files and make innovations to them. All these are done in the common area which is known as ‘Index’ or ‘Staging Area’.

**Q) Why is it considered to be easy to work on Git?**

With the help of git, developers have gained many advantages in terms of performing the development process faster and in a more efficient manner. Some of the main features of git which has made it easier to work are:

Branching Capabilities:  
  
- Due to its sophisticated branching capabilities, developers can easily work on multiple branches for the different features of the project.  
- It also has an easier merge option along with an efficient work-flow feature diagram for tracking it.

Distributed manner of development:  
  
- Git is a distributed system and due to this nature, it became easier to trace and locate data if it's lost from the main server.  
- In this system, the developer gets a repository file that is present on the server. Along with this file, a copy of this is also stored in the developer’s system which is called a local repository.  
- Due to this, the scalability of the project gets drastically improved.

Pull requests feature:  
  
- This feature helps in easier interaction amongst the developers of a team to coordinate merge-operations.  
- It keeps a proper track of the changes done by developers to the code.

Effective release cycle:  
  
- Due to the presence of a wide variety of features, git helps to increase the speed of the release cycle and helps to improve the project workflow in an efficient manner

**Q) How will you create a git repository?**

Have git installed in your system.

Then in order to create a git repository, create a folder for the project and then run git init.

Doing this will create a .git file in the project folder which indicates that the repository has been created.

**Q) Tell me something about git stash?**

Git stash can be used in cases where we need to switch in between branches and at the same time not wanting to lose edits in the current branch. Running the git stash command basically pushes the current working directory state and index to the stack for future use and thereby providing a clean working directory for other tasks.

**Q) What is the command used to delete a branch?**

To delete a branch we can simply use the command git branch –d [head].

To delete a branch locally, we can simply run the command: git branch -d <local\_branch\_name>

To delete a branch remotely, run the command: git push origin --delete <remote\_branch\_name>

Deleting a branching scenario occurs for multiple reasons. One such reason is to get rid of the feature branches once it has been merged into the development branch.

**Q) What differentiates between the commands git remote and git clone?**

git remote command creates an entry in  git config that specifies a name for a particular URL. Whereas git clone creates a new git repository by copying an existing one located at the URL

**Q) What does git stash apply command do?**

git stash apply command is used for bringing the works back to the working directory from the stack where the changes were stashed using git stash command.

This helps the developers to resume their work where they had last left their work before switching to other branches

**Q) Differentiate between git pull and git fetch.**

| git pull | git fetch |
| --- | --- |
| This command pulls new changes from the currently working branch located in the remote central repository. | This command is also used for a similar purpose but it follows a two step process:  1. Pulls all commits and changes from desired branch and stores them in a new branch of the local repository.  current 2. For changes to be reflected in the current / target branch, git fetch should be followed by git merge command. |

**Q) Can you tell the difference between Git and GitHub?**

| Git | GitHub |
| --- | --- |
| This is a distributed version control system installed on local machines which allow developers to keep track of commit histories and supports collaborative work. | This is a cloud-based source code repository developed by using git. |
| This is maintained by “The Linux Foundation”. | This was acquired by “Microsoft” |
| SVN, Mercurial, etc are the competitors | GitLab, Atlassian BitBucket, etc are the competitors. |

GitHub provides a variety of services like forking, user management, etc along with providing a central repository for collaborative work.

**Q) What do the git diff and git status commands do?**

| git diff | git status |
| --- | --- |
| This shows the changes between commits, working trees, etc. | This shows the difference between the working directory and index that is essential in understanding git in depth. |

git diff works in a similar fashion to git status with the only difference of showing the differences between commits and also between the working directory and index

**Q) How would you recover a branch that has already pushed changes in the central repository but has been accidentally deleted from every team member’s local machines?**

We can recover this by checking out the latest commit of this branch in the reflog and then checking it out as a new branch

**Q) Can you tell something about git reflog?**

This command tracks every single change made in the repository references (that can be branches or tags) and also maintains the branches/tags log history that was either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, renamed, or any commits made on the branch are maintained by Git and listed by the ‘reflog’ command.

This recovery of the branch is only possible when the branch was either created locally or checked-out from a remote repository in your local repository for Git to store its reference history logs.

This command should be executed in the repository that had the lost branch

We can recover this by checking out the latest commit of this branch in the reflog and then checking it out as a new branch

**Q)  What is a detached HEAD and what causes this and how to avoid this?**

Detached HEAD indicates that the currently checked-out repository is not a local branch. This can be caused by the following scenarios:

When a branch is a read-only branch and we try to create a commit to that branch, then the commits can be termed as “free-floating” commits not connected to any branch. They would be in a detached state.

When we checkout a tag or a specific commit and then we try to perform a new commit, then again the commits would not be connected to any branch. When we now try to checkout a branch, these new commits would be automatically placed at the top.  
  
In order to ensure that detached state doesn't happen, =instead of checking out commit/tag, we can create a branch emanating from that commit and then we can switch to that newly created branch by using the command: git checkout -b <<new\_branch\_name>>. This ensures that a new branch is checkout out and not a commit/tag thereby ensuring that a detached state wouldn't happen

**Q) What is the difference between git stash apply vs git stash pop command?**

git stash pop command throws away the specified stash (topmost stash by default) after applying it.

git stash apply command leaves the stash in the stash list for future reuse. In case we wanted to remove it from the list, we can use the git stash drop command.

git stash pop = git stash apply + git stash drop

**Q) What command helps us know the list of branches merged to master?**

git branch --merged helps to get the list of the branches that have been merged into the current branch.

Note: git branch --no-merged lists the branches that have not been merged to the current branch

**Q) How will you resolve conflict in Git?**

Conflicts occur whenever there are multiple people working on the same file across multiple branches. In such cases, git won't be able to resolve it automatically as it is not capable of deciding what changes has to get the precedence.

Following are the steps are done in order to resolve git conflicts:  
1. Identify the files that have conflicts.  
2. Discuss with members who have worked on the file and ensure that the required changes are done in the file.  
3. Add these files to the staged section by using the git add command.  
4. Commit these changes using the git commit command.  
5. Finally, push the changes to the branch using the git.

**Q) How to revert a bad commit which is already pushed?**

There can be cases where we want to revert from the pushed changes and go back to the previous version. To handle this, there are two possible approaches based on the situations:

Approach 1: Fix the bad changes of the files and create a new commit and push to the remote repository. This step is the simplest and most recommended approach to fix bad changes. You can use the command: git commit -m "<message>"

Approach 2: New commit can be created that reverts changes done in the bad commit. It can be done using git revert <name of bad commit>

**Q) What is the functionality of “git cherry-pick” command?**

This command is used to introduce certain commits from one branch onto another branch within the repository. The most common use case is when we want to forward- or back-port commits from the maintenance branch to the development branch.

**Q) Explain steps involved in removing a file from git index without removing from the local file system?**

Sometimes we end up having certain files that are not needed in the git index when we are not being careful while using the git add command. Using the command git rm will remove the file from both the index and the local working tree which is not always desirable.

Instead of using the git rm command we can use the git reset command for removing the file from the staged version and then adding that file to the .gitignore file to avoid repeating the same mistake again.

git reset <file\_name> # remove file from index

echo filename >> .gitingore # add file to .gitignore to avoid mistake repetition.

**Q) What are the factors involved in considering which command to choose among: git merge and git rebase?**

Both these commands ensure that changes from one branch are integrated into another branch but in very different ways. Git rebasing can be thought of as saying to use another branch as a new base for the work.

Whenever in doubt, it is always preferred to use the git merge command.  
  
Following are some factors that tell when to use merge and rebase commands:

In case our branch gets contributions from other developers outside the team as in open-source or public repositories, then rebase is not preferred.  
- This is because rebase destroys the branch and it results in broken and inconsistent repositories unless the git pull --rebase command is used.

Rebase is a very destructive operation. If not applied correctly, it results in loss of committed work which might result in breaking the consistency of other developer’s contribution to the repository.

If the model of having branches per feature is followed, rebasing is not a good idea there because it keeps track of related commits done by the developers. But in case the team follows having branches per developer of the team, then the branch has no additional useful information to be conveyed. In this model, rebasing has no harm and can be used.

If there is any chance where there might be a necessity to revert a commit to previous commits, then reverting a rebase would be almost impossible as the commit data would be destroyed. In such cases, the merge can be used.

**Q) How do you find a commit which broke something after a merge operation?**

This can be a time-consuming process if we are not sure what to look at exactly. Fortunately, git provides a great search facility that works on the principle of binary search as git-bisect command.

The initial set up is as follows:

git bisect start # initiates bisecting session

git bisect bad # marks current revision as bad

git bisect good revision # marks last known commit as good revision

Upon running the above commands, git checks out a revision that is labeled as halfway between “good” and “bad” versions. This step can be run again by marking the commit as “good” or “bad” and the process continues until the commit which has a bug is found.

**Q) What are the functionalities of git reset --mixed and git merge --abort?**

git reset --mixed command is used for undoing changes of the working directory and the git index.

git merge --abort command is used for stopping the merge process and returning back to the state before the merging occurred.

**Q) Can you tell the differences between git revert and git reset?**

| git revert | git reset |
| --- | --- |
| This command is used for creating a new commit that undoes the changes of the previous commit. | This command is used for undoing the local changes done in the git repository |
| Using this command adds a new history to the project without modifying the existing history | This command operates on the commit history, git index, and the working directory. |

**Q) What command removes untracked files from the working directory**

git clean

**Q) Which language is used in Git?**

Git uses 'C' language. Git is quick, and 'C' language makes this possible by decreasing the overhead of run times contained with high-level languages.

**Q) What is 'bare repository' in Git?**

A "bare" repository in Git includes the version control information and no working files (no tree), and it doesn?t include the special. git sub-directory. Instead, it consists of all the contents of the .git sub-directory directly in the main directory itself, whereas working list comprises of:

A .git subdirectory with all the Git associated revision history of your repo.

A working tree, or find out copies of your project files.

**Q) What is GIT stash drop?**

When you are done with the stashed element or want to delete it from the directory, run the git 'stash drop' command. It will delete the last added stash item by default, and it can also remove a specific topic if you include as an argument.

**Q) Why do we require branching in GIT?**

With the help of branching, you can keep your branch, and you can also jump between the different branches. You can go to your past work while at the same time keeping your recent work intact.

**Q) What is the purpose of 'git config'?**

The 'Git config' is a great method to configure your choice for the Git installation. Using this command, you can describe the repository behavior, preferences, and user information.

**Q) What is the difference between git pull and git fetch?**

Git pull command pulls innovation or commits from a specific branch from your central repository and updates your object branch in your local repository.

Git fetch is also used for the same objective, but it works in a slightly different method. When you behave a git fetch, it pulls all new commits from the desired branch and saves it in a new branch in your local repository. If you need to reflect these changes in your target branch, git fetch should be followed with a git merge. Your target branch will only be restored after combining the target branch and fetched branch. To make it simple for you, remember the equation below:

Git pull = git fetch + git merge

**Q) Why GIT better than Subversion?**

GIT is an open source version control framework; it will enable you to run 'adaptations' of a task, which demonstrate the changes that were made to the code over time also it allows you keep the backtrack if vital and fix those changes. Multiple developers can check out, and transfer changes, and each change can then be attributed to a particular developer.

**Q) What does the committed item contain?**

Commit item contains the following parts; you should specify all the three present below:

A set of records, representing to the condition of a task at a given purpose of time

References to parent commit objects

An SHAI name, a 40 character string that uniquely distinguishes the commit object

**Q) Describing branching systems you have utilized?**

This question is a challenge to test your branching knowledge with Git along these lines, inform them regarding how you have utilized branching in your past activity and what reason does it serves, you can refer the below mention points:

Feature Branching:

A component branch model keeps the majority of the changes for a specific element within a branch. At the point when the item is throughout tested and approved by automated tests, the branch is then converged into master.

Task Branching

In this model, each assignment is actualized on its branch with the undertaking key included in the branch name. It is anything but difficult to see which code actualizes which task, search for the task key in the branch name.

Release Branching

Once the create branch has procured enough features for a discharge, you can clone that branch to frame a Release branch. Making this branch begins the following discharge cycle so that no new features can be included after this point, just bug fixes, documentation age, and other release oriented assignments ought to go in this branch. When it is prepared to deliver, the release gets converged into master and labeled with a form number. Likewise, it should be converged once again into creating a branch, which may have advanced since the release was started.

At last, disclose to them that branching methodologies fluctuate starting with one association then onto the next, so I realize essential branching activities like delete, merge, checking out a branch, etc.

**Q) What is Subgit? Why use it?**

'Subgit' is a tool that migrates SVN to Git. It is a stable and stress-free migration. Subgit is one of the solutions for a company-wide migration from SVN to Git that is:

It is much superior to git-svn

No need to change the infrastructure that is already placed.

It allows using all git and all sub-version features.

It provides stress ?free migration experience.

**Q) How do you squash last N commits into a single commit?**

Squashing multiple commits into a single commit will overwrite history, and should be done with caution. However, this is useful when working in feature branches. To squash the last N commits of the current branch, run the following command (with {N} replaced with the number of commits that you want to squash):

git rebase -i HEAD~{N}

Upon running this command, an editor will open with a list of these N commit messages, one per line. Each of these lines will begin with the word “pick”. Replacing “pick” with “squash” or “s” will tell Git to combine the commit with the commit before it. To combine all N commits into one, set every commit in the list to be squash except the first one. Upon exiting the editor, and if no conflict arises, git rebase will allow you to create a new commit message for the new combined commit.

**Q) How do you find a list of files that has changed in a particular commit?**

git diff-tree -r {hash}

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

The output will also include some extra information, which can be easily suppressed by including a couple of flags:

git diff-tree --no-commit-id --name-only -r {hash}

Here --no-commit-id will supress the commit hashes from appearing in the output, and --name-only will only print the file names, instead of their paths.

How do you setup a script to run every time a repository receives new commits through push?

To configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

Update hook works in a similar manner to pre-receive hook, and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.

Finally, post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the “.git” directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

**Q) How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?**

This can be done with a simple script bound to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository. For example, the following script:

#!/bin/sh

files=$(git diff --cached --name-only --diff-filter=ACM | grep '.go$')

if [ -z files ]; then

exit 0

fi

unfmtd=$(gofmt -l $files)

if [ -z unfmtd ]; then

exit 0

fi

echo “Some .go files are not fmt’d”

exit 1

… checks to see if any .go file that is about to be commited needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

**Q) How can you copy a commit made in one branch to another (e.g. a hot fix commit from released branch to current development branch)?**

You need to use the cherry-pick command. It provides the possibility to play back an existing commit to your current location/branch. So you need to switch to the target branch (e.g. git checkout development) and call git cherry-pick {hash of that commit}.

In spite of applying the same changes, it will be a new commit with a new hash because the changes are applied to a different destination.

**Q) What is a commit message?**

 A commit message explains what change you made to your project. It is greatly important to learn how to make a good commit message no matter if it is a personal or professional project.

**Q) How can you fix a broken commit?**

In order to fix any broken commit, use the command “git commit --amend”. When you run this command, you can fix the broken commit message in the editor.

**Q) How is git instaweb used?**

‘git instaweb’ is used to automatically direct a web browser and run a webserver with an interface into your local repository.

**Q) Explain the advantages of forking workflow**

There is a fundamental difference between the forking workflow and other popular git workflows. Rather than using a single server-side to act as the “central” codebase, it gives every developer their own server-side repository. The Forking Workflow is commonly seen in public open-source projects.

A crucial advantage of the Forking Workflow is that contributions can be integrated without even needing everybody to push to a single central repository that leads to clean project history. Developers can push to their own server-side repositories, but only the project maintainer can push to the official repository.

If developers are ready to publish a local commit, then they push the commit to their own public repository and not the official one. After this, they go for a pull request with the main repository that lets the project maintainer know an update is ready to be integrated.

**Q) What does ‘hooks’ comprise of in Git?**

This directory consists of shell scripts that are activated if you run the corresponding Git commands.  For example, git will try to execute the post-commit script after you have run a commit

**Q) What is Git fork? What is the difference between fork, branch, and clone?**

A fork is a copy of a repository. Normally you fork a repository so that you are able to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else’s project or to use someone else’s project as a starting point for your own idea.

git cloning means pointing to an existing repository and make a copy of that repository in a new directory, at some other location. The original repository can be located on the local file system or on remote machine accessible supported protocols. The git clone command is used to create a copy of an existing Git repository.

In very simple words, git branches are individual projects within a git repository. Different branches within a repository can have completely different files and folders, or it could have everything the same except for some lines of code in a file.

**Q) What is the difference between rebasing and merge in Git?**

In Git, the rebase command is used to integrate changes from one branch into another. It is an alternative to the “merge” command. The difference between rebasing and merge is that rebase rewrites the commit history in order to produce a straight, linear succession of commits.

Merging is Git’s way of putting a forked history back together again. The git merge command helps you take the independent lines of development created by git branch and integrate them into a single branch.

**Q) Explain the difference between reverting and resetting.**

Git reset is a powerful command that is used to undo local changes to the state of a Git repository. Git reset operates on “The Three Trees of Git” which are, Commit History ( HEAD ), the Staging Index, and the Working Directory.

Revert command in Git creates a new commit that undoes the changes from the previous commit. This command adds a new history to the project. It does not modify the existing history.