**Git Assignments:**

**Assignment**

Module-2: GIT Assignment - 1

Based on what you have learnt in the class, do the following steps:

● Create a new folder

● Put the following files in the folder

1. Code.txt
2. Log.txt
3. Output.txt

● Stage the Code.txt and Output.txt files

● Commit them

● And finally push them to github

Please share the commands for the above points

Module-2: GIT Assignment - 2 Do the following tasks: ● Create a git working directory with feature1.txt and feature2.txt in the master branch ● Create 3 branches develop, feature1 and feature2 ● In develop branch create develop.txt, do not stage or commit it ● Stash this file, and checkout to feature1 branch ● Create new.txt file in feature1 branch, stage and commit this file ● Checkout to develop, unstash this file and commit Please submit all the git commands used to do the above steps

Module-2: GIT Assignment - 3 You have been asked to: ● Create a git working directory, with the following branches ○ Develop ○ F1 ○ f2 ● In the master branch, commit main.txt file ● Put develop.txt in develop branch, f1.txt and f2.txt in f1 and f2 respectively ● Push all these branches to github ● On local delete f2 branch ● Delete the same branch on github as well

Module-2: GIT Assignment - 4 You have been asked to: ● Put master.txt on master branch, stage and commit ● Create 3 branches: public1, public2 and private ● Put public1.txt on public 1 branch, stage and commit ● Merge public 1 on master branch ● Merge public 2 on master branch ● Edit master.txt on private branch, stage and commit ● Now update branch public 1 and public 2 with new master code in private ● Also update new master code on master ● Finally update all the code on the private branch

Module-2: GIT Assignment - 5 You have been asked to: ● Create a gitflow workflow architecture on git ● Create all the required branches ● Starting from the feature branch, push the branch to the master, following the architecture ● Push a urgent.txt on master using hotfix

CASE STUDY - GIT WORKFLOW You work as a Devops Architect in Zendriix Softwares. The company has been struggling to manage their product releases. The releases should happen on 25th of every month. Suggest a Git Workflow Architecture for this requirement. Simulate this workflow, by creating a pseudo code files and branches, and upload the same to your GitHub Account. As a part of solution, share the link to your GitHub repository

CASE STUDY- RESOLVING MERGE CONFLICTS You work for Zendrix Software & Co. You have been assigned the task of updating the Master branch of their Git repository with all the features from the feature branches. Following is the GitHub account, https://github.com/devops-intellipaat/merge-conflict.git Consider, • Feature1 branch to be a public branch • Feature2 branch to be a private branch The company relies on a monolithic architecture, and for now all the code resides in one file “main.c”. The respective features have been added in the feature branches for main.c. Meanwhile, a security patch was made to the master branch, and now feature1 and feature2 branches are behind from master by 1 commit. Following tasks have to be done: 1. Update Feature1 and Feature2 branch with the Security Patch 2. Apply changes of Feature1 and Feature2 branch on master 3. Finally push all the branches to GitHub For Solving this, please fork the repository to your Github account and then work. As a solution, please submit your GitHub’s repository link

**Assignment-2**

Do the following tasks:

● Create a git working directory with feature1.txt and feature2.txt in the master branch

● Create 3 branches develop, feature1 and feature2

● In develop branch create develop.txt, do not stage or commit it

● Stash this file, and checkout to feature1 branch

● Create new.txt file in feature1 branch, stage and commit this file

● Checkout to develop, unstash this file and commit Please submit all the git commands used to do the above steps

**Assignment-3**

You have been asked to:

● Create a git working directory, with the following branches ○ Develop ○ F1 ○ f2

● In the master branch, commit main.txt file

● Put develop.txt in develop branch, f1.txt and f2.txt in f1 and f2 respectively

● Push all these branches to github

● On local delete f2 branch

● Delete the same branch on github as well

**Assignment-4**

You have been asked to:

● Put master.txt on master branch, stage and commit

● Create 3 branches: public1, public2 and private

● Put public1.txt on public 1 branch, stage and commit

● Merge public 1 on master branch

● Merge public 2 on master branch

● Edit master.txt on private branch, stage and commit

● Now update branch public 1 and public 2 with new master code in private

● Also update new master code on master

● Finally update all the code on the private branch

**Assignment-5**

You have been asked to:

● Create a gitflow workflow architecture on git

● Create all the required branches

● Starting from the feature branch, push the branch to the master, following the architecture

● Push a urgent.txt on master using hotfix

1. [What is a clone in GitHub?](https://mindmajix.com/github-interview-questions#what-is-a-clone-in-github)
2. [How much space do we get on GitHub?](https://mindmajix.com/github-interview-questions#how-much-space-do-we-get-on-github)
3. [What do you know about GitHub and its repository?](https://mindmajix.com/github-interview-questions#what-do-you-know-about-github-and-its-repository)
4. [Can you tell us a few benefits of using GitHub over other platforms?](https://mindmajix.com/github-interview-questions#can-you-tell-us-a-few-benefits-of-using-github-over-other-platforms)
5. [Compare Git with SVN](https://mindmajix.com/github-interview-questions#compare-git-with-svn)
6. [Are you familiar with the Git Clone?](https://mindmajix.com/github-interview-questions#are-you-familiar-with-the-git-clone)
7. [What is the significance of Git version control](https://mindmajix.com/github-interview-questions#what-is-the-significance-of-git-version-control)
8. [What is the gist of Git?](https://mindmajix.com/github-interview-questions#what-is-the-gist-of-Git)
9. [How can we create a gist?](https://mindmajix.com/github-interview-questions#how-can-we-create-a-gist)
10. [What is gist programming?](https://mindmajix.com/github-interview-questions#what-is-gist-programming)
11. Which version of git do you use?
12. What is git merge and git rebase?
13. Do you know git squash?
14. What branching strategy do you work on for your project? Which strategy do you feel better and why?
15. What is a version control system (VCS)?
16. What is a git repository?
17. What does git clone do?
18. What does the command git config do?
19. Can you explain head in terms of git and also tell the number of heads that can be present in a repository?
20. What is a conflict?
21. What is the functionality of git ls-tree?
22. What does the git status command do?
23. Define “Index”.
24. What does the git add command do?
25. Why is it considered to be easy to work on Git?
26. How will you create a git repository?
27. Tell me something about git stash?
28. What is the command used to delete a branch?
29. What differentiates between the git remote and git clone?
30. What does git stash apply command do?
31. Differentiate between git pull and git fetch.
32. Can you give differences between “pull request” and “branch”?
33. Why do we not call git “pull request” as “push request”?
34. Can you tell the difference between Git and GitHub?
35. What do the git diff and git status commands do?
36. What has to be run to squash multiple commits (last N) into a single commit?
37. 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?
38. Can you tell me something about git reflog?
39. What consists of a commit object?
40. Explain the levels in git config and how can you configure values using them?
41. What is a detached HEAD and what causes this and how to avoid this?
42. What does the git annotate command do?
43. What is the difference between git stash apply vs git stash pop command?
44. What command helps us know the list of branches merged to master?
45. How will you resolve conflict in Git?
46. What is the best advisable step in cases of broken commit: Create an additional commit OR amend an existing commit?
47. How to revert a bad commit that is already pushed?
48. What is the functionality of the “git cherry-pick” command?
49. Explain steps involved in removing a file from the git index without removing it from the local file system?
50. What are the factors involved in considering which command to choose among git merge and git rebase?
51. How do you find a commit which broke something after a merge operation?
52. What are the functionalities of git reset --mixed and git merge --abort?
53. Can you tell the differences between git revert and git reset?

**GIT INTERVIEW QUESTIONS**

1. Why we need git? What makes git unique from other tools like SVN?
2. Let's say i have maven repo cloned on to my local, did some changes and i have built the code now target folder will be generated. So now when i do git operations like git add, git commit or any other git operations target folder should not be considered, how would you achieve the same?
3. difference between git pull and git fetch?
4. How to clone specific branch in git?
5. Let’s say your organization has github and bitbucket to store code, you have cloned a repo onto your local and changed directory name. after some days one of your team members asks you to share clone link, how would you provide the same?
6. I have shell script to delete particular dependency ( repo is maven project). before running the script i need to clone repo to my local, here point to note i should only clone master branch and only last commit (last commit has all the code) how would you do this?
7. what is submodule and why we need submodule?
8. Let’s say you have changed 5 files a, b, c, d and e in a repo and you did git add., now all the files are in staging area, now i decided not to commit file d. how would delete it from staging area?
9. What is a conflict in Git?
10. What is dependency and plugin in maven? Give one example for each?
11. What are 3 build lifecycle in maven?
12. What is git reset ? Types of reset ?
13. How to delete local branch and remote branch in git?
14. Difference between git diff and git status?
15. What are hooks in git?
16. what is .git?
17. what is a feature branch? Why do you need branching?
18. what is the branching strategy you are using? Which one you feel good like?
19. which version of git you have used?
20. what is git merge and git rebase?
21. What is GitHub APIs?
22. What is Webhooks?
23. What is Cherry-pick? Why we use it?
24. command to list all branches in a repo?
25. Let’s say you are working on new feature in some branch, now your manager says stop working on that and change few other things on old code. Here after changing the old code, I need to place my new changes some place how would you handle this scenario?
26. what is chocklety?
27. How your pipeline works and How you configure that?

**Q: I have a shell script to delete a particular dependency in a repo (repo is a maven project). Before running this script, I need to clone the repo to my local, here point note I should only clone the master branch and only the last commit (as it contains full code) how would you do this?**

**I. e, Last commit and master branch?**

**Ans: git clone -----**

**Q: What is a submodule and why do we need a submodule?**

**Ans:**

**Q: Let’s say I have changed 5 files a, b, c, d, e in the repo and you did git add. , now all 5 files are in the staging area, now I decided not to commit file d. how would you delete from the staging area to local?**

**Git:**

**What is conflict in Git?**

**GIT**

1. What is git reset? Types of resets?

2. How to delete the local branch and remote branch in git?

3. Difference between git diff and git status?

4. What are hooks in git?

What is the branching strategy you used? How you handled in production?

Before going to production what are the branches you need to work with?

What is GitHub Apis? Why do we use it?

What are the repositories based on API (list out), and what is the structure you follow?

What are webhooks? Why it is used?

When you try to add git configurations, which file do you do modifications to?

**Basic Git Interview Questions**

1. What is Git?

[Git](https://www.simplilearn.com/tutorials/git-tutorial/what-is-git) is a version control system for tracking changes in computer files and is used to help coordinate work among several people on a project while tracking progress over time. In other words, it’s a tool that facilitates source code management in software development.

Git favors both programmers and non-technical users by keeping track of their project files. It enables multiple users to work together and handles large projects efficiently.

With the help of the versioning system, the developer can identify who has made what changes and then run tests and fix bugs if any and then do necessary feature implementation. In case of any unforeseen circumstances, the code can be reverted to any of the previously working versions thereby saving huge efforts.

**2. What do you understand by the term ‘Version Control System’?**

A [version control system](https://www.simplilearn.com/tutorials/git-tutorial/what-is-git) (VCS) is a system that records all changes made to a file or set of data, so a specific version may be called later if needed.

This helps ensure that all team members are working on the latest version of the file.



**3. What’s the difference between Git and GitHub?**

**Git**

Git is a version control system for tracking changes in computer files. The main point of Git is to manage projects, or a set of them when changes are made over time. It helps to track progress over time and coordinate work among several people on a project.

**GitHub**

[GitHub is a Git repository hosting service](https://www.simplilearn.com/tutorials/git-tutorial/git-vs-github) that provides a web-based graphical interface. GitHub helps every team member to work together on the project from anywhere, making collaboration easy.

**3. What is a Git repository?**

[Git repository](https://www.simplilearn.com/git-tutorial-article) refers to a place where all the Git files are stored. These files can either be stored on the local repository or on the remote repository.



**4. How can you initialize a repository in Git?**

If you want to initialize an empty repository to a directory in Git, you need to enter the **git init**command. After this command, a hidden **.git** folder will appear in the folder.

**4. What does the command git config do?**

The git config command is a convenient way to set configuration options for defining the behavior of the repository, user information and preferences, git installation-based configurations, and many such things.

For example: To set up your name and email address before using git commands, we can run the below commands:

git config --global user.name “<>”

git config --global user.email “<>

**5. Name a few Git commands with their function.**

* Git config - Configure the username and email address
* Git add - Add one or more files to the staging area
* Git diff - View the changes made to the file
* Git init - Initialize an empty Git repository
* Git commit - Commit changes to head but not to the remote repository

**6. What are the advantages of using Git?**

* Faster release cycles
* Easy team collaboration
* Widespread acceptance
* Maintains the integrity of source code
* Pull requests

**7. What language is used in Git?**

Git is a fast and reliable version control system, and the language that makes this possible is ‘C.’

Using [C language](https://www.simplilearn.com/tutorials/git-tutorial/what-is-git) reduces the overhead of run times, which are common in high-level languages.

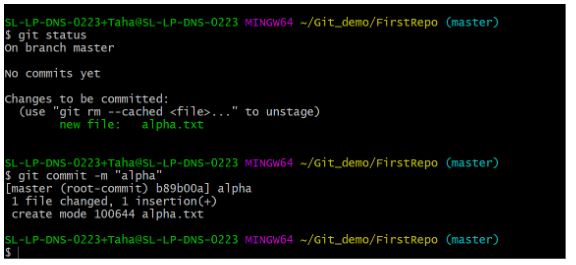
3. What does git clone do?

The command creates a copy (or clone) of an existing git repository. Generally, it is used to get a copy of the remote repository to the local repository.

**8. What is a commit message, and how is the commit command executed?**

The commit command is executed in a Git project to record the progress in the local repository. The commit command is executed only after the files to be committed have been added to the staging area using the git add command.

The command that makes it possible to write a commit message is ‘**git commit -m’**.



**9. Name some of the popular Git hosting repositories.**

* GitHub
* GitLab
* BitBucket
* Beanstalk
* FogBugz
* Surround SCM
* Buddy

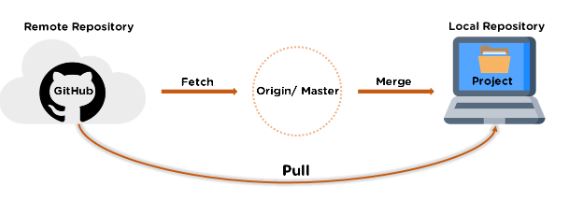
**10. Explain the git push command.**

The [Git push command](https://www.simplilearn.com/tutorials/git-tutorial/git-push-command) is used to push the content in a local repository to a remote repository. After a local repository has been modified, a push is executed to share the modifications with remote team members.



**11. Explain the git pull command.**

Git pull is used to fetch and merge changes from the remote repository to the local repository. Git pull is a combination of two commands: git fetch; followed by git merge.



**12. Difference between git fetch and git pull.**

Git fetch

* Git fetches only downloads new data from a remote repository.
* It does not integrate any of these new data into your working files.
* Can be done any time to update the remote-tracking branches

Command - git fetch origin

                    git fetch –-all

Git pull

* Git pull updates the current HEAD branch with the latest changes from the remote server.
* Downloads new data and integrate it with the current working files.
* Tries to merge remote changes with your local ones.

Command - git pull origin master

**13. What is a merge conflict in Git?**

A [merge conflict](https://www.simplilearn.com/tutorials/devops-tutorial/devops-interview-questions) is an event that takes place when Git is unable to resolve differences in code between the two commits automatically.

Git is able to automatically merge the changes only if the commits are on different lines or branches.



**14. How do you resolve a merge conflict?**

To resolve a merge conflict in Git, follow these steps:

* The most simple way to resolve the conflicted file is to open it and make the required changes
* After editing the file, we can use the git add a command to stage the new merged content
* The final step is to create a new commit with the help of the git commit command
* Git will create a new merge commit to finalize the merge

**15. What is the process to revert a commit that has already been pushed and made public?**

There are two processes through which you can revert a commit:

1. Remove or fix the bad file in a new commit and push it to the remote repository. Then commit it to the remote repository using:

**git commit –m “commit message”**

2. Create a new commit to undo all the changes that were made in the bad commit. Use the following command:

**git revert <commit id>**

**17. How is a bare repository different from the standard way of initializing a Git repository?**

Using a standard way

* You create a working directory with the git init command
* A .git subfolder is created with all the git-related change history

Using a bare way

* Does not contain any working or checked out a copy of source files
* Bare repositories store git revision history in the root folder of your repository instead of the .git subfolder

**16. What is Git stash?**

Let’s say you're a developer and you want to switch branches to work on something else. The issue is you don’t want to make commits in uncompleted work, so you just want to get back to this point later. The solution here is the Git stash.

Git stash takes your modified tracked files and saves it on a stack of unfinished changes that you can reapply at any time. To go back to the work, you can use the stash pop.

**17. What does the git reset --mixed and git merge --abort commands do?**

git reset --mixed is used to undo changes made in the working directory and staging area.

git merge --abort helps stop the merge process and return back to the state before the merging began.

**18. How do you find a list of files that has been changed in a particular commit?**

The command to get a list of files that has been changed in a particular commit is:

git diff-tree –r {commit hash}

* -r flag allows the command to list individual files
* commit hash lists all the files that were changed or added in the commit.

Now let’s increase the level of difficulty with advanced Git interview questions and answers.

**19. Explain the different points when a merge can enter a conflicted stage.**

There are two stages when a merge can enter a conflicted stage.

1. Starting the merge process

* If there are changes in the working directory of the stage area in the current project, the merge will fail to start
* In this case, conflicts happen due to pending changes that need to be stabilized using different Git commands

2. During the merge process

* The failure during the merge process indicates that there’s a conflict between the local branch and the branch being merged
* In this case, Git resolves as much as possible, but some things have to be fixed manually in the conflicted files

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

**Fork**

The fork is the process when a copy of the repository is made. It's usually experimentation in the project without affecting the original project. They’re used to advise changes or take inspiration from someone else’s project.

**Branch**

Git branches refer to individual projects within a git repository. If there are several branches in a repository, then each branch can have entirely different files and folders.

**Clone**

Git clone refers to creating a clone or a copy of an existing git repository in a new directory. Cloning automatically creates a connection that points back to the original repository, which makes it very easy to interact with the central repository.

**21. What is the difference between git merge and git rebase?**

To incorporate new commits into your feature branch, you use **merge**

* Creates an extra merge commit every time you need to incorporate changes
* Pollutes your feature branch history

As an alternative to merging, you can rebase the feature branch into master.

* Incorporates all the new commits in the master branch
* Rewrites the project history by creating brand new commits for each commit in the original branch

**22. What is the command used to fix a broken commit?**

To fix a broken commit in Git, you may use the “git commit --amend” command, which helps you combine the staged changes with the previous commits instead of creating an entirely new commit.

**23. How is ‘git remote’ different from ‘git clone’?**

**git remote**

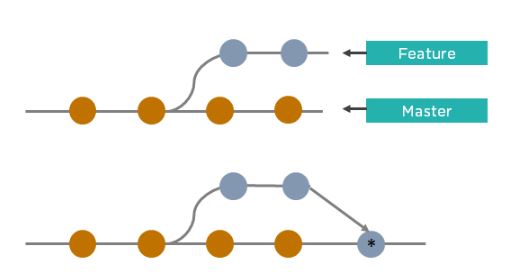
**git remote**enables you to create, view, and delete connections to other repositories. It's used to refer to a remote repository or a central repository.

**git clone**

**git clone**enables you to create a clone or copy of the target repository. It's used to target a different already existing repository.

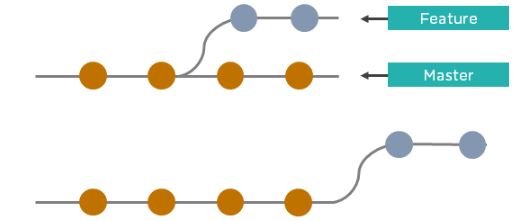
**23. What is the difference between git merge and git rebase?**

To incorporate new commits into your feature branch, you use merge



* Creates an extra merge commit every time you need to incorporate changes
* Pollutes your feature branch history

As an alternative to merging, you can rebase the feature branch into master.



* Incorporates all the new commits in the master branch
* Rewrites the project history by creating brand new commits for each commit in the original branch

**24. What is the command used to fix a broken commit?**

To fix a broken commit in Git, you may use the “git commit --amend” command, which helps you combine the staged changes with the previous commits instead of creating an entirely new commit.

**25. How is ‘git remote’ different from ‘git clone’?**

git remote

git remote enables you to create, view, and delete connections to other repositories. It's used to refer to a remote repository or a central repository.

git clone

git clone enables you to create a clone or copy of the target repository.

It's use to target a different already existing repository.

**26.  What is git stash drop?**

The Git stash drop command is used to remove a particular stash. If there’s a stash you're no longer using or you want to remove a specific item of stash from the list, you can use the stash commands.

Let’s say you want to delete an item named stash@{abc}; you can use the command git stash drop stash@{abc}.

**27. If you recover a deleted branch, what work is restored?**

The files that were stashed and saved in the stash index can be recovered. The files that were untracked will be lost. That’s why it's a good idea to stage and commit your work or stash them.

**28. What’s the difference between reverting and resetting?**

**Reverting**

The revert command in Git is used to create a new commit that undoes the changes made in the previous commit. When you use this command, a new history is added to the project; the existing history is not modified.

**Resetting**

Git reset is a command that is used to undo the local changes that have been made to a Git repository. Git reset operates on the following: commit history, the staging index, and the working directory.

**29. How can you discover if a branch has already been merged or not?**

There are two commands to determine these two different things.

1. git branch --merged - Gives the list of branches that have been merged into the current branch.
2. git branch --no-merged - Gives the list of branches that have not been merged.

**30. What is “git cherry-pick”?**

The command git cherry-pick enables you to pick up commits from a branch within a repository and apply it to another branch. This command is useful to undo changes when any commit is accidentally made to the wrong branch. Then, you can switch to the correct branch and use this command to cherry-pick the commit.

**5. 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

6. 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 a

7. 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.

8. What does git status command do?

git status command is used for showing the difference between the working directory and the index which is helpful for understanding git in-depth and also keep track of the tracked and non-tracked changes.

9. 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’.

In the above image, the “staged” status indicates the staging area and provides an opportunity for the people to evaluate changes before committing them.

10. What does git add command do?

1. This command adds files and changes to the index of the existing directory.
2. You can add all changes at once using git add . command.
3. You can add files one by one specifically using git add command.
4. You can add contents of a particular folder by using git add // command.

**11. 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.

**13. 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.

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 To delete a branch remotely, run the command: git push origin --delete 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.

**15. 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.

**16. 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.

**17. Differentiate between git pull and git fetch.**

**git pull**

This command pulls new changes from the currently working branch located in the remote central repository.

**git fetch**

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.

git pull = git fetch + git merge

**18. Can you give differences between “pull request” and “branch”?**

**pull request:**

This process is done when there is a need to put a developer’s change into another person’s code branch.

**Branch:**

A branch is nothing but a separate version of the code.

**19. Why do we not call git “pull request” as “push request”?**

“Push request” is termed so because it is done when the target repository requests us to push our changes to it.

“Pull request” is named as such due to the fact that the repo requests the target repository to grab (or pull) the changes from it.

**21. What do the git diff and git status commands do?**

**git diff**

This shows the changes between commits, working trees, etc.

**git status**

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.

**22. What has to be run to squash multiple commits (last N) into a single commit?**

Squashing multiple commits to a single one overwrites the history which is why it is recommended to be done using full caution. This step can be done by running the command: git rebase -i HEAD~{{N}} where {{N}} represents the number of commits needed to be squashed.

**23. 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.

**24. 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.

**25. What consists of a commit object? A commit object consists of the following components:**

A set of files that represents the state of a project at a given point in time.

Reference to parent commit objects.

A 40-character string termed as SHA-1 name uniquely identifies the commit object.

**26. Explain the levels in git config and how can you configure values using them?**

=> In order to make git work, it uses a set of configurations that are pre-defined by default by means of configuration files (or config files). We can change the default behavior of git by just modifying these files which are basically text files. In order to do this, it is important to understand how git identifies these files. It does so by following the below steps:

- Firstly, git searches for the config values in the system-wide gitconfig file stored in <>/etc/gitconfig file that has settings defined and applied to every user of the system and all their repos.

 - In case you want git to search from this particular file and read/write on it, we can pass the option --system to git config command.

- Next, git searches for the ~/.gitconfig file or ~/.config/git/config that has the scope specific to the user.  - Git can be made to read/ write from this file specifically bypassing --global to the git config command.

- Lastly, git searches for the config values in the git directory of the local repository that we are currently working on.

- These config values are specific to that particular repository alone and can be accessed by passing --local to the git config command. This is the default config file that gets accessed and modified upon in case we do not specify any levels.

**27. 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 <> . This ensures that a new branch is checkout out and not a commit/tag thereby ensuring that a detached state wouldn't happen.

**28. What does git annotate command do?**

This command annotates each line within the given file with information from the commit which introduced that change. This command can also optionally annotate from a given revision.

Syntax: git annotate [<options>] <file> [<revision>]

You can get to learn more about this command from the official git documentation here.

**29. 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

Advanced GIT Interview Questions

**30. 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.

**31. 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.

**32. What is best advisable step in cases of broken commit: Create an additional commit OR amend an existing commit?**

It is always advisable to create an additional commit rather than amending the existing commit due to the following reasons:

- Doing the amend operation destroys the previously saved state of that commit. If only the commit message gets changes or destroyed, it's acceptable but there might be cases when the contents of the commits get amended. This results in the loss of important information associated with the commit.

- Over usage of git commit --amend can have severe repercussions as the small commit amend can continue to grow and gather unrelated changes over time.

**33. 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 " "

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>

**34. 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.

**35. 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 # remove file from index

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

**36. 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.

**37. 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.

**38. 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: How clone specified branch in git?**

Ans: Git clone -b branch name URL

**39. Can you tell the differences between git revert and git reset?**

git revert

This command is used for creating a new commit that undoes the changes of the previous commit.

Using this command adds a new history to the project without modifying the existing history.

git reset

This command is used for undoing the local changes done in the git repository

This command operates on the commit history, git index, and the working directory.

**40. Why do we need git? What makes GIT unique from tools like SVN?**

**Scenario: -**

Let’s say I have maven repo cloned on to my local, did some changes and I have built the code now target folder will be generated, So now when I do git operations as git add, git commit or any other git operations target folder should not be considered, how would you achieve the same?

**or**

I don’t want to push the target into git? How do you avoid that?

**or**

When you do **maven install** it will create a target directory, when I want to push the updated code into bitbucket, the target folder shouldn’t be considered?

**Ans:** we can **rm -rf** to delete the target but every time we are not doing this, we create a shell script, which will clone a repository and clean install everything, it pushes back once the code is fine,

**41. How to avoid the target folder to store in git?**

Move to root directory of the project in parallel to .git folder, create [.gitignore](https://git-scm.com/docs/gitignore) file and copy the below content:

target/

add an item to. gitignore so git doesn't pay attention to it (if files haven't been added to previous revisions).

Add the path to your file that you would like to ignore to yours .gitignore file.

Example:

\*.class

target

.classpath

.settings

If you don’t want to add the target folder into the commit, you should add the folder. gitignore. The. gitignore file is usually created in the root path of a git repo. You can use the below ways to created a .gitignore file, and ignore the target folder:

touch .gitignore

echo target >> .gitignore

# If you committed files and subfolders of target in git, use below commits

git rm 1/\* --cached -r

git commit -m 'remove target in version control'

Now the target folder is totally ignored by git, and it won’t be committed in git.

How to clone specific branch from the git repository.

You can clone a specific branch from a Git repository using the**git clone –single-branch –branch command**. This command retrieves all the files and metadata associated with one branch. To retrieve other branches, you’ll need to fetch them later on.

The references of other branches were also cloned with the above command.

**How Git helps DevOps professionals**

**IaaC: Infrastructure as a code.**

We need to manage the infra using code rather than using it manually.

**Ansible, Terraform**

We need to ensure that this code is stored in the version control system.

**Have you worked on push and pull or have you worked on Admin activities also?**

As a DevOps Engineer, not deep into admin activities, as a DevOps engineer level, we have used.

**Scenario:**

Let’s say your organization has GitHub and bitbucket to store code, you have cloned repo in your local and changed the cloned folder name to some other name. after a few days one of your team members asks you that share the clone link, what would you do? The problem here is you changed cloned directory name (by default code will clone to directed with repo name), now if you search with changed directory name in GitHub or bitbucket it won’t list.

**Ans:** when I cloned a repository. git folder was there, In. git, we can save every information about the clone to required. We write the command git remote #v then it can give me an alternative repository from there I can share that repo link with my colleague.

**What is Git workflow? Why do you need to determine a Git workflow?**

Identifying a single Git workflow is a necessary step in ensuring rapid delivery. Software development teams encompass contributors from various backgrounds and experiences, and they’re likely to feel comfortable with a workflow they’ve used previously. Without a single workflow, a team’s development could be chaotic and slow down cycle time. [Git workflows empower teams](https://about.gitlab.com/blog/2019/10/28/optimize-gitops-workflow/) to determine roles and responsibilities, set boundaries, and identify areas of improvement.

A Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner. Git workflows encourage users to leverage Git effectively and consistently.

**The top six Git workflows to strengthen software development**

**1. Centralized Git workflow**

A centralized Git workflow enables all team members to make changes directly to the main branch, with every change logged in running history. A [centralized workflow](https://about.gitlab.com/topics/version-control/what-is-centralized-version-control-system/) involves every contributor committing to the main branch without using any other branch. This strategy works well for small teams because team members can communicate so that multiple developers aren’t contributing to the same piece of code simultaneously. Centralized workflow can be seamless if team members communicate well, but there are limitations. If multiple developers commit to the same branch, it’s challenging to find a stable moment to release changes. Consequently, developers must keep unstable changes local until they’re ready for release.

**What is the benefit of a centralized Git workflow?**

After developers apply a stash and solve any merge conflicts, they can just commit as usual without dealing with [automatic merge commits](https://about.gitlab.com/blog/2020/03/25/git-merge-fifteen-year-git-party/), unless someone pushed their changes at the same time. Because this strategy is simple, it is well-suited for small teams, Git beginners, and projects that don’t get a lot of updates.

**2. Feature branching Git workflow**

Every feature gets its own branch when developers commit to this workflow. Rather than commit directly to the main branch, developers create a branch, make changes, and then merge it into main.

Ideally, a branch should have a lifespan of a few hours. The longer the branch lives, the higher the risk to find integration conflicts when merging back to main. After all, at this scale, there are plenty of teams working on other branches and directly streaming changes to the main branch, incrementing entropy and chances of running into conflict with local changes.

**What is the benefit of a feature branching Git workflow?**

This Git workflow has the benefit of keeping a clean main branch that isn’t polluted with unfinished features. Teams of any size can use this feature branching, because it permits multiple developers to work on the same feature simultaneously. Software that’s still in development sees the most benefit from feature branching, but this workflow can be used for more mature applications as well.

**3. Trunk-based development Git workflow**

Trunk-based development facilitates concurrent development on a single branch called trunk. When developers are ready to push changes to the central repository, they’ll pull and rebase from it to update the working copy of the central branch. Successful trunk-based development requires a developer to resolve merge conflicts locally. Regularly updating the local branch reduces the impact of integration changes, because they’re spotted when they’re still small, avoiding merge hell.

**What is the benefit of trunk-based development Git workflow?**

Trunk-based development decreases the likelihood of merge conflicts and keeps code clean, because there are many frequent, small merges made each day. With continuous integration, a trunk-based workflow ensures fast feedback and a team-oriented approach to code ownership and development.

**4. Personal branching Git workflow**

Personal branching is similar to feature branching, but rather than have a single branch per feature, it’s per developer. This approach works well if team members work on different features and bugs. Every user can merge back to the main branch whenever their work is done.

**What is the benefit of personal branching Git workflow?**

Personal branching has similar advantages as feature branching, and also benefits from having fewer branches, so branch management is easier. Personal branches can be used for bug fixes and other small changes, and they help developers innovate if they’re interested in experimenting. Personal branching is useful for long-running features that may not fit into a single release cycle. This strategy can work well for small teams in which every team member develops their own part of the application.

**5. Forking Git workflow**

A forking approach to [version control](https://about.gitlab.com/topics/version-control/) starts with a complete copy of the repository. Forking effectively creates a local copy of a Git repository and provides the ability to create a new collaboration structure. In other words, every developer in the team has two repositories: a local workspace and a remote repository.

This workflow is popular for projects that have multiple developers contributing to it, particularly open-source projects. After all, keeping track and providing privileges to collaborate to a repository with thousands of contributors is difficult to maintain. If a maintainer enables contributors to try their changes on their forked copy, managing change proposals is easier and safer.

**What is the benefit of a forking Git workflow?**

With a forking workflow, contributors can push changes to a server-side repository, decreasing the likelihood of including low-quality code and bugs into the source code. Only a project maintainer can integrate changes to the source code. When large teams collaborate on software development projects, forking enables secure, quality-driven development.

**6. GitFlow Git workflow**

With GitFlow, the main branch should always be releasable to production, and there should never be untested or incomplete code on the main branch. When using this Git workflow, no one commits to the main branch but rather uses a develop branch with feature branches. When the develop branch is ready to go to production, a contributor creates a release branch where testing and bug fixing occur before being merged back to the develop branch. The release branch makes the code review process easier, because there’s a dedicated place to resolve conflicts when merging into the main branch. With this strategy, the main branch always reflects production.

**What is the benefit of a GitFlow Git workflow?**

The benefit of GitFlow as a Git production workflow is that it allows larger teams to work on complex software while still being able to quickly fix bugs in production. In addition, the release branch allows for a staging period where users can test the software before it’s released, which doesn’t hinder code development. Teams of any size can use GitFlow, but smaller teams may find one of the other strategies easier to use due to its complexity. When dealing with multiple environments and regular deployments, GitFlow may offer the [workflow flexibility](https://nvie.com/posts/a-successful-git-branching-model/) some teams require.

There are three workflows that are accepted and are followed by various tech companies.

1. **Centralized workflow**
2. **Feature Branching**
3. **GitFlow workflow**

**Centralized workflow**

* This workflow doesn’t require any other branch other than master.
* All the changes are directly made on the master and finally merged on the remote master, once work is finished.
* Before pushing changes, the master is rebased with the remote commits.
* Results in a clean and linear history.

**Feature Branching**

* Master only contains the production-ready code
* Any Development work is converted into a feature branch
* There can be numerous feature branches, depending on the application’s development plan
* Once the feature is complete, the feature branch is merged with the master.

**GitFlow workflow**

* In this workflow, we have a master, a development, and then feature branches
* A feature branch is never merged directly with the master.
* Rather than Master, the feature branch is merged with the development branch.
* Once there are enough features on the Develop branch, it is merged with the Master with a corresponding version number.

**What is Forking?**

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

**Glossary: -**

**Git:** an open-source, distributed version control system

**GitHub:** A platform for hosting and collaborating on Git repositories.

**Commit:** A Git object, a snapshot of your entire repository compressed into an SHA

**Branch:** A lightweight movable pointer to a commit

**Clone:** A local version of a repository, including all commits and branches

**Remote:** A common repository on GitHub that all team members use to exchange their changes.

**Fork:** A copy of a repository on GitHub owned by a different user

**Pull request:** A place to compare and discuss the differences introduced on a branch with reviews, comments, integrated tests, and more.

**HEAD**: Representing your current working directory, the HEAD pointer can be moved to different branches, tags, or commits when using **git checkout.**