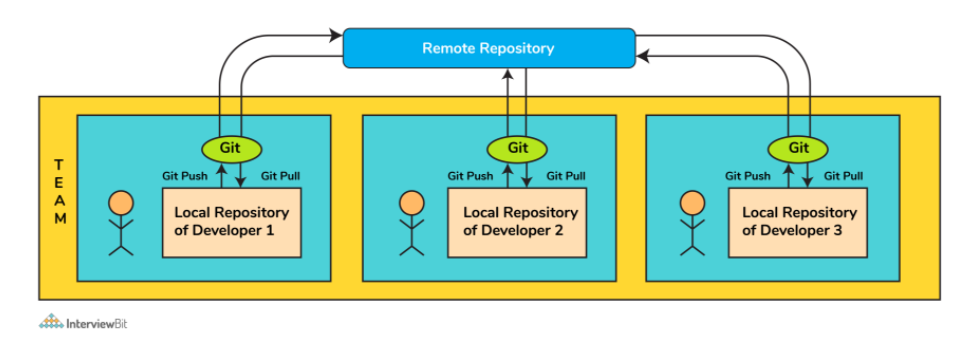
## What is Git and why is it used?

Git is an open-source distributed version control system and source code management (SCM) system with an insistence to control small and large projects with speed and efficiency.

GIT is a distributed version control system and source code management (SCM) system with an emphasis to handle small and large projects with speed and efficiency.

It is most suitable when there are multiple people working on projects as a team and is used for tracking the project changes and efficiently supports the collaboration of the development process.

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.



It 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.

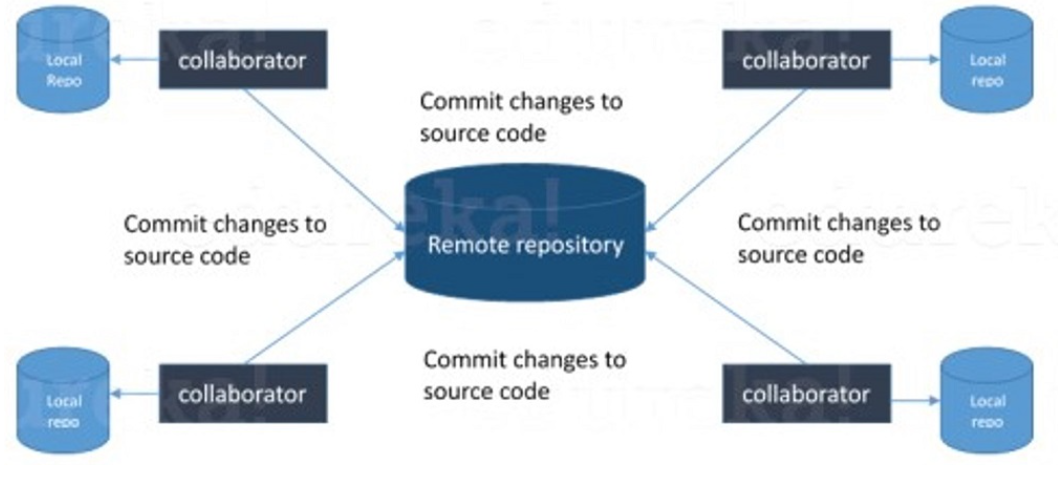
Git favors both programmers and non-technical users by keeping track of their project files. It enables multiple users to work together and handles large projects efficiently.

Git is a Distributed Version Control system(DVCS). It lets you track changes made to a file and allows you to revert back to any particular change that you wish.

It is a distributed architecture that provides many advantages over other Version Control Systems (VCS) like SVN. One of the major advantages is that it does not rely on a central server to store all the versions of a project’s files.

Instead, every developer “clones” a copy of a repository I have shown in the diagram with “Local repository” and has the full history of the project available on his hard drive. So when there is a server outage all you need to do to recover is one of your teammate’s local Git repository.

There is a central cloud repository where developers can commit changes and share them with other teammates.



## 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.

Some of the best GIT client for LINUX is

a) Git Cola

b) Git-g

c) Smart git

d) Giggle

e) Git GUI

f) qGit

## What is a version control system (VCS)?

A distributed VCS is a system that does not depend upon a central server to keep a project file and all its versions. In distributed VCS, each collaborator or developer gets a local copy of the main repository and this is called a clone.

A VCS keeps track of the contributions of the developers working as a team on the projects. They maintain the history of code changes done and with project evolution, it gives an upper hand to the developers to introduce new code, fixes bugs, and run tests with confidence that their previously working copy could be restored at any moment in case things go wrong.

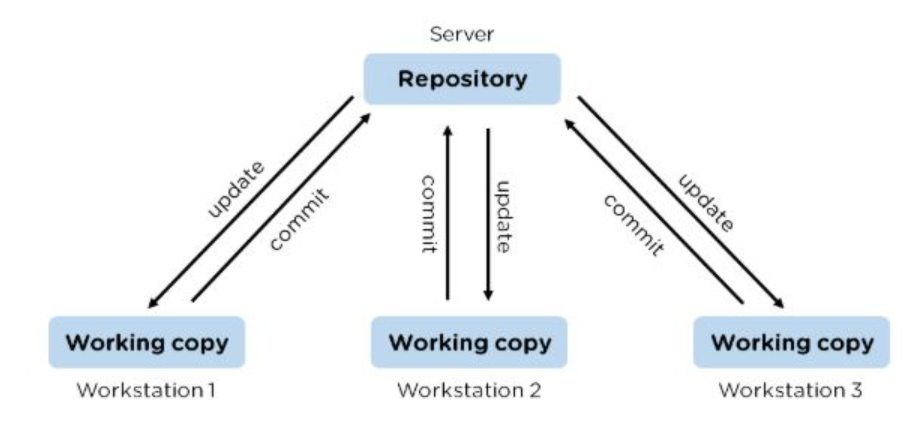
As you can see in the below diagram, every collaborator maintains a local repository on their local machines. They can commit and update the local repositories without any issues.

A version control system (VCS) records all the 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

Using a pull operation, a developer can update his local repository with the latest changes from the central server.

Using the push operation, they can send their changes from the local repository to the central server.



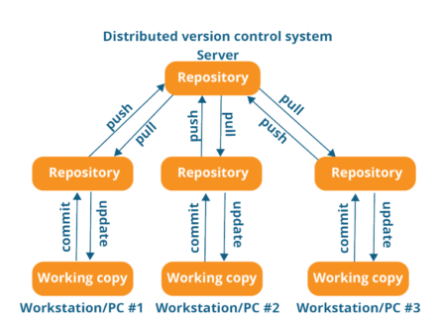
A version control system (VCS) is a software tool used to create different project versions and store them in a repository. All modifications to the code are recorded and tracked by the VCS.

**Types of version control systems:**

* **Local version control systems** have a database that maintains all the file changes on disk under revision control in a special format.
* **Centralized version control systems** contain one repository, and each user gets their own working copy.
* **Distributed version control systems** contain multiple repositories, each accessible to separate users with their own working copy.

## **What is a distributed VCS?**

* These are the systems that don’t rely on a central server to store a project file and all its versions.
* In Distributed VCS, every contributor can get a local copy or “clone” of the main repository.
* As you can see in the above diagram, every programmer can maintain a local repository which is actually the copy or clone of the central repository which is present on their hard drive. They can commit and update their local repository without any hassles.
* With an operation called “pull”, they can update their local repositories with new data from the central server and “pull” operation affects changes to the main repository from their local repository.



## **What are the benefits of using Version Control System?**

* With the Version Control System(VCS), all the team members are allowed to work freely on any file at any time. VCS gives you the flexibility to merge all the changes into a common version.
* All the previous versions and variants are neatly packed up inside the VCS. You can request any version at any time as per your requirement and you’ll have a snapshot of the complete project right at hand.
* Whenever you save a new version of your project, your VCS requires you to provide a short description of the changes that you have made. Additionally, you can see what changes are made in the file’s content. This helps you to know what changes have been made in the project and by whom.
* A distributed VCS like Git allows all the team members to have a complete history of the project so if there is a breakdown in the central server you can use any of your teammate’s local Git repository.

## 6. What are the advantages of using GIT?

a) Data redundancy and replication

b) High availability

c) Only one .git directory per repository

d) Superior disk utilization and network performance

e) Collaboration friendly

f) Any sort of projects can use GIT

g) Faster release cycles

h) Easy team collaboration

i) Widespread acceptance

j) Maintains the integrity of source code

k) [Pull requests](https://www.simplilearn.com/tutorials/git-tutorial/git-pull-request)

**(i) Free & Open Source:**

* Git is issued under GPL’s (General Public License) open-source license. You need not pay anything to use Git. It is absolutely free. As it is open-source, you can modify the source code according to your needs.

**(ii) Speed:**

* As you are not required to connect to any network for executing all the actions, it performs all the tasks quickly. Obtaining version history from a locally stored repository can be one hundred times speedier than obtaining it from the remote server.
* Git is written in C, which is the underlying programming language that evades runtime overheads linked with other high-level languages.

**(iii) Scalable:**

* Git is highly scalable. So, if the number of collaborators increases in the coming time, then Git can easily accommodate this change.
* Despite the fact that Git represents an entire repository, the data kept on the client’s side is very small as Git compacts the entire vast data through a lossless compression technique.

**(iv) Reliable:**

* As every collaborator has its own local repository, on the instances of a system crash, the lost data can be recuperated from any of the local repositories. At all times, you will have a backup of all your files.

**(v) Secure:**

* Git utilizes the SHA1 (Secure Hash Function) to name and identify objects inside its repository. Each artifact and commit are check-summed and recovered through its checksum during checkout.
* The Git history is saved in a manner in which the ID of a specific version (a commit in terms of Git) relies on the total development history running up to that commit. Once a file version is pushed to Git, then there is no way to change it without being noticed.

**(vi) Economical:**

* In the case of a centralized version control system, the central server must be strong enough to attend requests of the entire team. This is not a problem for smaller teams, however as the team expands, the hardware limitations of the server can be an impediment for performance.
* In the case of distributed version control systems like Git, the team members don’t require interaction with the server expect when they are required to push or pull changes. All the heavy lifting occurs at the client end, thus the server hardware can be kept quite simple certainly.

**(vii) Supports Non-linear Development:**

* Git provides rapid branching & merging and contains particular tools for envisaging and traversing a non-linear development history. A basic notion in Git is that a change will be merged more frequently than it is written as it is sent across different reviewers.
* Git Branches are extremely lightweight. A branch in Git refers only to a single commit. The complete branch structure can be created, with the help of parent commits.

**(viii) Easy Branching:**

* Branch management through Git is very straightforward and easy. It requires just a few jiffies to create, delete, and merge branches. Feature branches give an insulated environment to each change to your codebase.
* When a developer requires to begin working on something, irrespective of the size of work, they create a new branch. This makes sure that the master branch constantly holds a production-quality code.

**(ix) Distributed Development:**

* Git provides every developer a local copy of the whole development history, plus the changes get cloned from one such repository to another. These changes are introduced as added development branches and can be merged in the same manner as a locally developed branch.

**(x) Compatibility along with present Systems or Protocol:**

* Repositories can be published through HTTP, FTP or a Git protocol on top of either a plain socket or ssh.
* Since Git is an open source version control system it lets us run multiple versions of our project so that it shows the changes that are made to the code over time and if needed we can keep track of the changes that we have made. This means that a large number of developers can make their own changes and upload those changes so that the changes can be attributed to the particular developers.

## **6. What is Git Bash?**

Git Bash is an application that installs Bash, Git, and a few Bash utilities that are commonly used on a Windows OS. In Git Bash, interaction is possible with Git elements and the repository through different commands.

## 7.What’s the difference between [Git and GitHub](https://www.simplilearn.com/tutorials/git-tutorial/git-vs-github)?

|  |  |
| --- | --- |
| Git | GitHub |
| Git is a software | [GitHub](https://www.simplilearn.com/tutorials/git-tutorial/what-is-github) is a service |
| Git can be [installed](https://www.simplilearn.com/tutorials/git-tutorial/git-installation-on-windows) locally on the system | GitHub is hosted on the web |
| Provides a desktop interface called git GUI | Provides a desktop interface called GitHub Desktop. |
| It does not support user management features | Provides built-in user management |
| 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. |

| **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.

[Git](https://bit.ly/31MeW9b) is a version control system of distributed nature that is used to track changes in source code during software development. It aids in coordinating work among programmers, but it can be used to track changes in any set of files. The main objectives of Git are speed, data integrity, and support for distributed, non-linear workflows.

[GitHub](https://bit.ly/2rVhL7Q) is a Git repository hosting service, plus it adds many of its own features. GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, basic task management tools for every project.

Git is a version control system that is used in the management of the source code history.

GitHub, on the other hand, is a cloud-based hosting service that is used in the management of Git repositories. GitHub is designed to help in the better management of open-source projects.

## 8.How is Git different from Subversion (SVN)?

|  |  |
| --- | --- |
| GIT | SVN |
| Git is a distributed decentralized version control system | SVN is a centralized version control system. |
| Git stores content in the form of metadata. | SVN stored data in the form of files. |
| The master contains the latest stable release. | In SVN, the trunk directory has the latest stable release |
| The contents of Git are hashed using the SHA-1 hash algorithm. | SVN doesn’t support hashed contents. |
| It belongs to the 3rd generation of Version Control tools | It belongs to the 2nd generation of Version Control tools |
| Clients can clone entire repositories on their local systems | Version history is stored on a server-side repository |
| Commits are possible even if offline | Only online commits are allowed |
| Push/pull operations are faster | Push/pull operations are slower |
| Works are shared automatically by commit | Nothing is shared automatically |
| Git is less preferred for handling extremely large files or frequently changing binary files | SVN can handle multiple projects stored in the same repository |
| GIT does not support ‘commits’ across multiple branches or tags | Subversion allows the creation of folders at any location in the repository layout |
| Gits are unchangeable | Subversion allows committers to treat a tag as a branch and to create multiple revisions under a tag root |

|  | **Git** | **SVN** |
| --- | --- | --- |
| **Server Architecture** | The computer on which your Git has installed acts as both client and server. Each developer has a local copy of the complete version history of the project on their individual computers. Git changes occur locally.  Hence, the developer is not required to be connected to the network at all times. Only for push and pull operations, developers would need internet connection to connect to remote server. | SVN has a separate client and server. It is not locally available. You will be required to be connected to the network to perform any action.  Also, in SVN, since everything is centralized, so in case the central server gets crashed or corrupted, it will result in entire data loss for the project. |
| **Branching** | Git is mostly preferred by developers due to its effective branching model. Git branches are lightweight but powerful.  They are only references to a particular commit. You can create, delete or modify a branch anytime with no impact on other commits. So, fork, branch and merge are easy with Git. | SVN has a complicated branching and merging model and its time-consuming to manage.  In SVN, branches are generated as directories within the repository. This directory structure is mainly problematic. When the branch is ready, you need to commit back to the trunk. Since you are not the only one who is merging the changes, so the version of the truck may not be regarded as developers’ branches. This can lead to conflicts, missing files and jumbled changes in your branch. |
| **Access Control** | Git presumes that all the contributors will be having the same permissions. | SVN permits you to define read/write access controls at each and directory level. |
| **Auditability** | In Git, the changes are tracked at the repository level. Git does not bother too much about maintaining the precise history of changes made in your repository. The distributed nature of Git lets any collaborator change any part of their local repo’s history. With Git, it’s difficult to figure a true history of changes in your codebase.  For example, you will lose history after rename in Git. | In SVN, the changes are tracked at the file level.  SVN maintains a pretty consistent and precise change history. You can recover exactly the same data as it was at any instant in the past.  SVN history is permanent and always definite. |
| **Storage Requirements** | Git and SVN store the data in the same manner. The disk space usage is equal for both of them. The only difference comes into picture in case of binary files. Git is not friendly to binary files.  It can’t handle the storage of large binary files. | SVN has an xDelta compression algorithm that works for both binary and text files.  So, SVN can handle storing large binary files in comparatively lesser space than Git. |
| **Usability** | Both Git and SVN use command line as primary UI.  Git is largely used by developers/technical users. | SVN is largely used by non-technical users as it's easier to learn. |
| **Content** | Cryptographic SHA-1 Hash. | No hashed content. |
| **Global Revision Number** | Not available | Available |

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Git** | **SVN** |
| Type of version control | Distributed | Centralized |
| Access to networks | Not mandatory | Mandatory |
| Global revision number | Not available | Available |
| Content | Cryptographic SHA-1 Hash | No hashed content |

## 9.Why GIT better than Subversion?

GIT is an open-source version control system; it will allow you to run ‘versions’ of a project, which show the changes that were made to the code overtime also it allows you keep the backtrack if necessary and undo those changes.  Multiple developers can checkout, and upload changes and each change can then be attributed to a specific developer.

## 10. What is Subgit? Why to use Subgit?

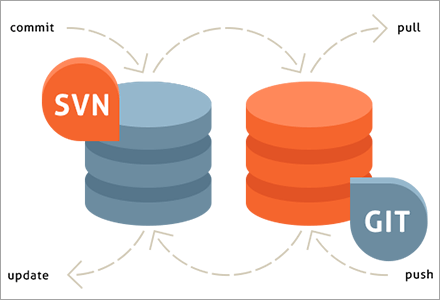
‘Subgit’ is a tool for a smooth, stress-free SVN to Git migration.  Subgit is a solution for a company -wide migration from SVN to Git that is:

a) It is much better than git-svn

b) No requirement to change the infrastructure that is already placed

c) Allows to use all git and all sub-version features

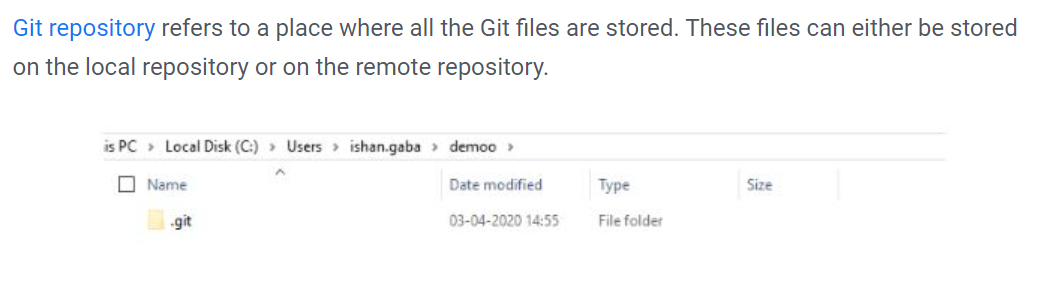
d) Provides genuine stress –free migration experience.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/SubGit.png)

1. You can create an SVN||Git mirror using this tool. SubGit should be installed on your Git server. It will detect all the settings of your remote SVN repository, including SVN revisions, branches, and tags, and converts them into Git commits.
2. It also preserves the history including tracking merge data.
3. SubGit is a tool that is used to migrate SVN to Git. It transforms the SVN repositories to Git and allows you to work on both systems concurrently. It auto-syncs the SVN with Git.
4. SubGit is a tool for SVN to Git migration. It can create a writable Git mirror of a local or remote Subversion repository and use both Subversion and Git as long as you like.
5. Now you can also include some advantages like you can do a fast one-time import from Subversion to Git or use SubGit within Atlassian Bitbucket Server. We can use SubGit to create a bi-directional Git-SVN mirror of an existing Subversion repository. You can push to Git or commit to Subversion as per your convenience. Synchronization will be done by SubGit.

## 11. What is a git repository?

A repository contains a directory named .git, where git keeps all of its metadata for the repository. The content of the . git directory are private to git.



Repositories contain a batch of files that are different versions of a project. These files are imported from the repositories into the local servers of users for further modifications and updates in the content.

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.

A few popular Git hosting services are:

* GitHub
* GitLab
* Bitbucket
* SourceForge
* Pikacode
* Visual Studio Online
* GitEnterprise

## 12. What is a. git Directory?

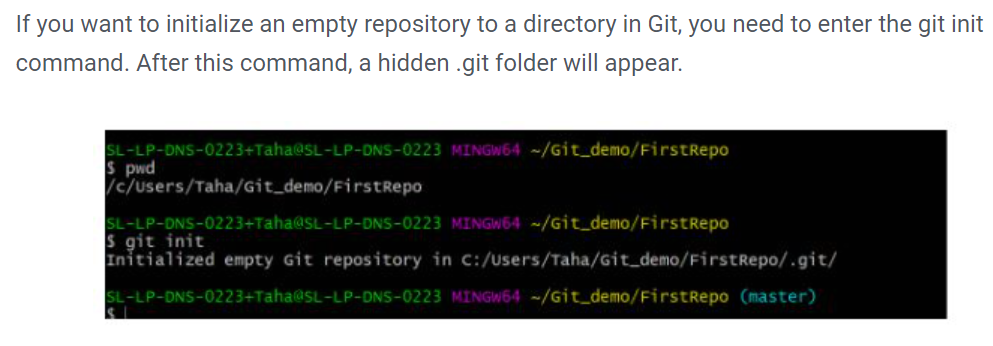
The moment you create a repository, you will find a .git directory present inside it. This .git directory contains all the metadata of the repository and maintains a track of all the changes made to the files in your repository, by keeping a commit history.

All the information regarding commits, hooks, refs, object databases, remote repository addresses, etc. are kept inside this folder. This is the most crucial part of Git. When you clone any Git repository on your local machine, this .git is the directory that actually gets copied.

## **13. What happens if the .git directory gets deleted?**

**If the .git/ directory gets deleted, then you will lose track of your project’s history. The repository will no longer be under version control.**

## 14. How can you initialize a repository in Git?



git init - This [command](https://www.simplilearn.com/tutorials/git-tutorial/git-commands) helps to create an empty repository while working on a project.

## 15. 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.

Once you clone a repo, you can make edits locally in your system rather than directly in the source files of the remote repo.

## **16. What is tagging in Git?**

**Tagging allows developers to mark all the important checkpoints through the course of their projects’ progress. Instead of commit IDs, tag names can be used while commits are checked out and pushed to a remote repo.**

## **17. 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.

## **18. 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.**

Forking creates a copy of the original repository, and it remains in the GitHub account. Whereas, in cloning, the repository is copied to the local machine using Git. Forking is used to propose changes to the repository owners. In cloning, the changes are directly pushed to the original repository, provided the user has write access.

A **branch** occurs within a repository and is a way to keep developing and modifying the software without affecting the main project.

|  |  |  |
| --- | --- | --- |
| **Fork** | **Branch** | **Clone** |
| **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.** | **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.** | **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.** |

Forking creates a copy of the original repository, and it remains in the GitHub account. Whereas, in cloning, the repository is copied to the local machine using Git. Forking is used to propose changes to the repository owners. In cloning, the changes are directly pushed to the original repository, provided the user has write access. A branch occurs within a repository and is a way to keep developing and modifying the software without affecting the main project.

## **19. What is the use of git fork? How is forking different from cloning?**

To fork a project means to create a remote, server-side copy of the original repository. You can rename this copy and start doing a new project around this without affecting the original project.

The fork is not the core concept of Git.

The fork operation is used by Git workflow and this idea exists longer for free and open-source software like GitHub. Generally, once you have forked the project, you will rarely contribute to the parent project again.

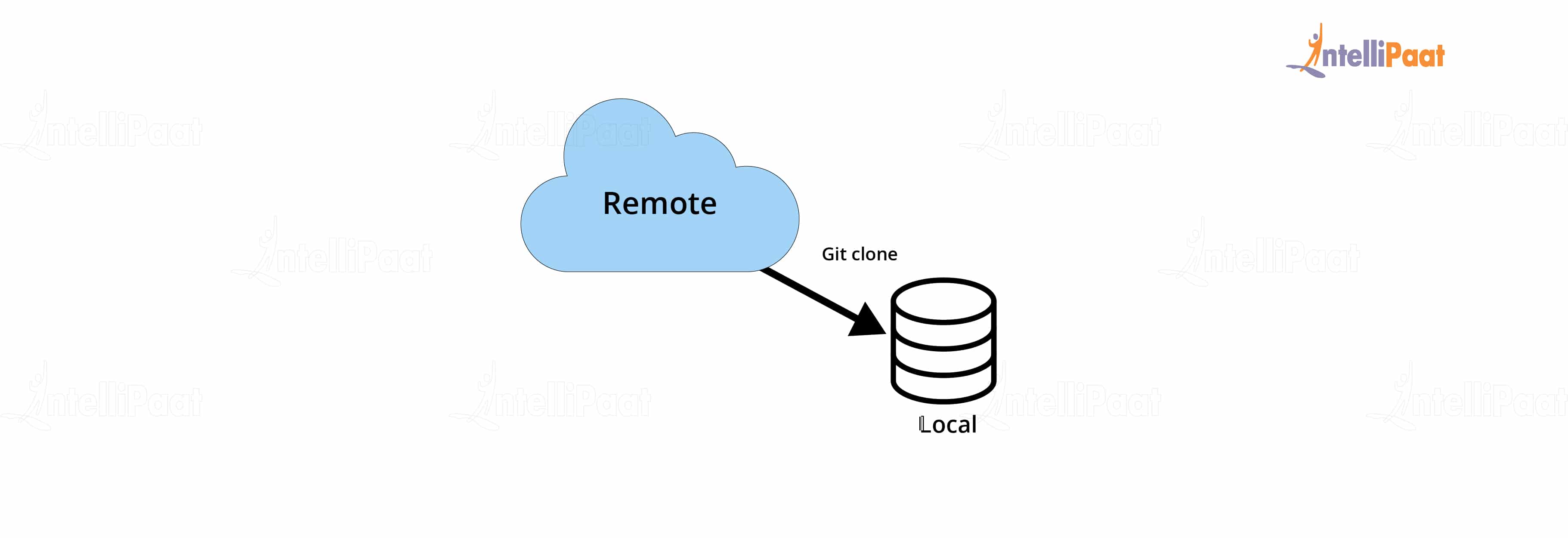
At any time, you can contribute back to the original project by using the pull requests.

**In the forked copy, all the main data like codes and files get copied from the original repository, however, branches, pull requests and other features do not get copied. Forking is an ideal way for open-source collaboration.**

Cloning is essentially a Git concept. A clone is a local copy of any remote repository. When we clone a repository, the entire source repository along with its history and branches gets copied to our local machine.

Unlike forking, there is no direct connection between the cloned repository and the original remote repository. If you want to do pull requests and continue back to the original project, then you should get yourself added as a collaborator in the original repository.

Cloning is also a great way for creating a backup of the original repository as the cloned copy also has all the commit history.



## **20. How will you find out what all files have been changed in a particular Git commit?**

**By using the hash value of the particular commit, you can execute the below command to get the list of files that have been changed in a particular commit:**

**git diff-tree -r {hash}**

**This will list down all the files that have been modified, and also the files that have been added. The -r flag is used to list individual files along with their path instead of collapsing them in their root directory names only.**

**You can also use the below command:**

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

**–no-commit-id will retrain the commit hash numbers to come in the output. W**

**-name will exclude the file paths and**

**only give the file names in the output.**

**git diff** shows the changes between commits, commit and working tree etc.

**git diff** is similar to **git status**, but it shows the differences between various commits and also between the working directory and index.

**Git diff** is a multi-use command that can be executed to show the differences between two arbitrary commits, changes between the working tree & a commit, changes between working tree & an index, changes between two files, changes between index & a tree, etc.

**git status** command is used to inspect a repository. It shows the state of the working directory and staging area. It will list down the files that have been staged, which haven’t been staged and the files that are untracked.

| **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. |

## **21. What is the difference between git checkout [branch name] and git checkout -b [branch name]?**

**The command git checkout [branch name] will switch from one branch to another.**

**The command git checkout -b [branch name] will create a new branch and also switch to it.**

## **22. How will you know in Git if a branch has already been merged into master?**

**git branch --merged – It lists the branches that have been merged into the current branch.  
git branch --no-merged – It lists the branches that have not been merged.**

**git branch –merged master – This will list out all the branches that have been renamed into master.**

**By default, this command tells the merge status of local branches only. If you want to know about both local and remote branch merge status, then you can use -a flag. If you want to check only for remote branches,**

**then you can use -r flag.**

## 23. What is git pull origin?

pull is a get and a consolidation. 'git pull origin master' brings submits from the master branch of the source remote (into the local origin/master branch), and then it combines origin/master into the branch you currently have looked out.

The git pull origin master fetches all the changes from the master branch onto the origin and integrates them into the local branch.

git pull = git fetch + git merge origin/ master

## 24. 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. 

The **git config** command is a convenient way to set configuration options for your Git installation.  Behaviour of a repository, user info, preferences etc. can be defined through this command.

**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  
  “<<your\_name>>”
* git config --global user.email “<<your\_email>>”

## **25. 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 <<installation\_path>>/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.**

## **26. What is the functionality of git clean command?**

**The git clean command removes the untracked files from the working directory.**

## 27. 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
* Git rm [file] - deletes the file from your working directory and stages the deletion.
* Git log - list the version history for the current branch.
* Git show [commit] – shows the metadata and content changes of the specified commit.
* Git tag [commitID] - used to give tags to the specified commit.
* Git checkout [branch name] - used to switch from one branch to another.
* Git checkout -b [branch name] - creates a new branch and also switches to it.

## 28. 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.

## **29. Mention the various Git repository hosting functions.**

* Github
* Gitlab
* Bitbucket
* SourceForge
* GitEnterprise

## **30. What is a commit message?**

Commit message is a feature of git which appears when you commit a change.

The command that is used to write a commit message is **git commit -a**.  
**-a** on the command line instructs/saying git to commit the new content of all tracked files that have been modified.

Also, mention you can use **git add <file>** before **git commit -a** if new files need to be committed for the first time.

git commit -m "x files created"

## **31. What are the different ways you can refer to a commit?**

* In Git each commit has a unique hash. These hashes are used to identify the corresponding commits in various scenarios, for example, while trying to checkout a particular state of the code using the git checkout {hash} command.
* Along with this, Git maintains a number of aliases to certain commits, known as refs. Also, every tag that is created in the repository effectively becomes a ref and that is exactly why you can use tags instead of committing hashes in various git commands. Git also maintains a number of special aliases that are changed based on the state of the repository, such as HEAD, FETCH\_HEAD, MERGE\_HEAD, etc.
* In Git, commits are allowed to be referred to as relative to one another. In the case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, for example, HEAD^2 can be used to follow the second parent.
* And finally, refspecs are used to map local and remote branches together. However, these can also be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local git environment.

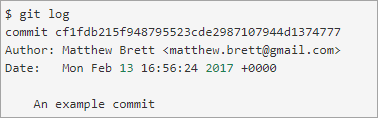
## 32. What does the committed item contain?

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
* A SHAI name, a 40-character string that uniquely distinguishes the commit object.

You can view this through the **git log** command.

**Example:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/commit-object.png)

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

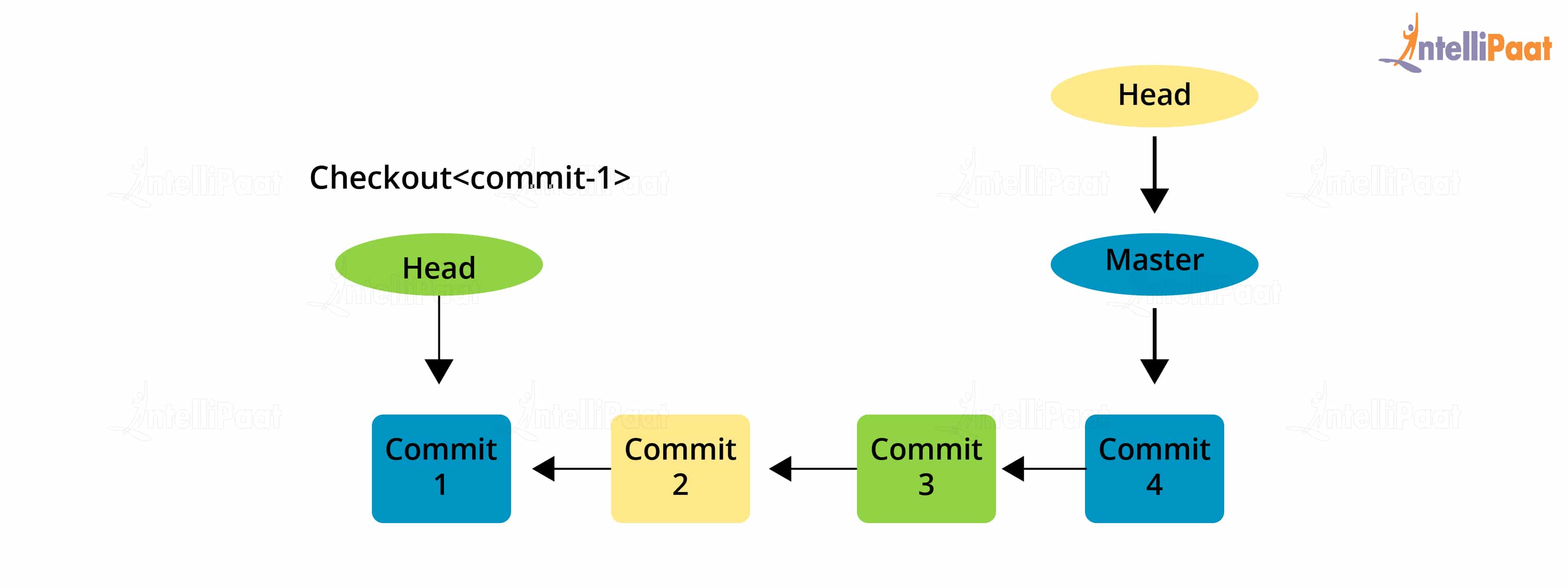
There are couple of reason

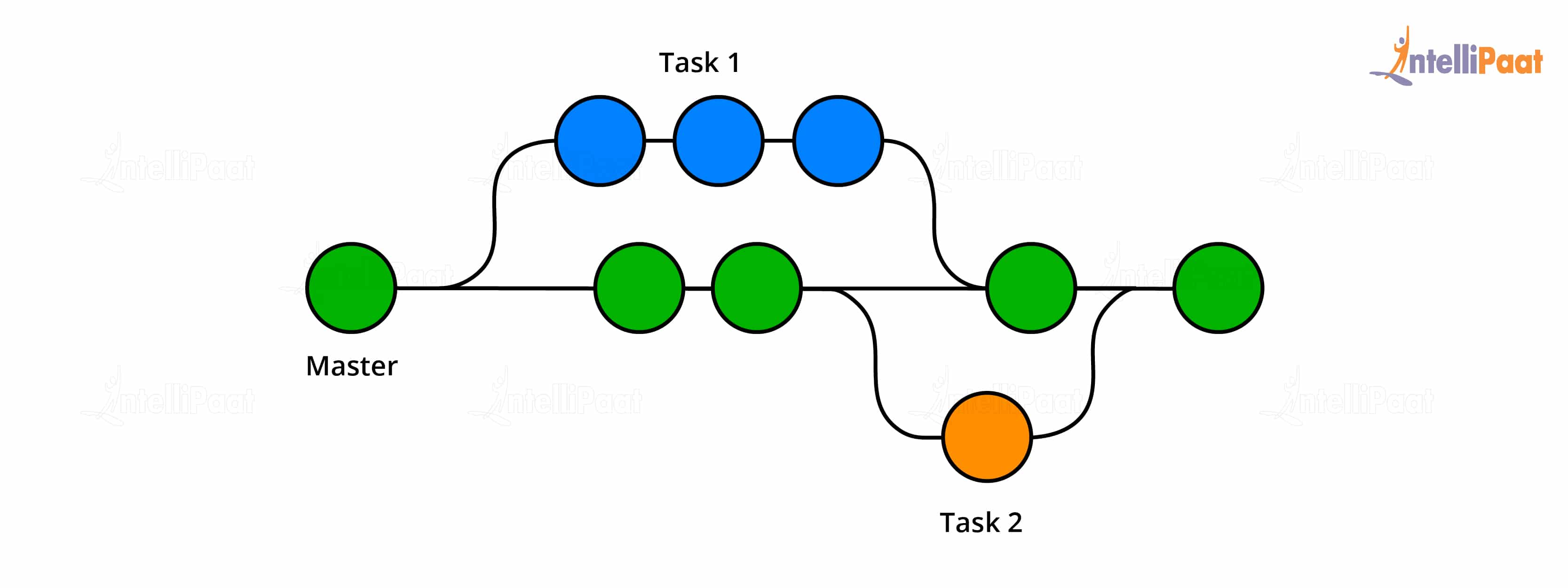
a)  The amend operation will destroy the state that was previously saved in a commit.  If it’s just the commit message being changed then that’s not an issue.  But if the contents are being amended then chances of eliminating something important remains more.

b) Abusing “git commit- amend” can cause a small commit to grow and acquire unrelated changes.

## **34. What is HEAD in Git, and how many HEADs can be created in a repository?**

The reference to a commit object is called the HEAD. Every repository has a ‘Master’ which is the default head. There can be multiple heads in a repository.





## 35. 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.

I will suggest you to first give a small introduction to sanity checking.

Sanity or smoke testdetermines whether it is possible and reasonable to continue testing.

## **36. What is ‘bare repository’ in Git?**

**A “bare” repository in Git contains information about the version control and no working files (no tree) and it doesn’t contain the special .git sub-directory. Instead, it contains all the contents of the .git sub-directory directly in the main directory itself,**

**whereas the working directory consists of:**

1. **A .git subdirectory with all the Git related revision history of your repository.**
2. **A working tree, or checked out copies of your project files.**

**To co-ordinate with the distributed development and developers team, especially when you are working on a project from multiple computers ‘Bare Repository’ is used. A bare repository comprises of a version history of your code.**

**Repositories that are created through *git init*command are the standard/non-bare Git repositories.**

**In the top-level folder of such repository, you will find two things:**

1. **A .git subdirectory keeping all metadata and track of the history of your repo.**
2. **A working tree.**

**The repositories which are created using *git init –bare*command are known as bare Git repositories.**

**They are mainly used for sharing. They do not contain any working tree.**

**They keep the git revision history of your repository in the root folder rather than having it inside the .git subfolder.**

**It just contains bare repository data. This is how a bare Git repository is different from a standard Git repository. Also, a bare repository does not have a default remote *origin*repository as it serves as an origin repository for multiple remote users.**

**Since a bare repository does not contain any workspace, the *git push*and *git pull* commands do not work over a bare repo. You are not required to commit any changes to a bare repo.**

|  |  |
| --- | --- |
| **Standard way** | **Bare way** |
| **You create a working directory with the git init command.** | **Does not contain any working or checked out copy of source files.** |
| **A. git subfolder is created with all the git-related change history.** | **Bare repositories store git revision history in the root folder of your repository instead of the .git subfolder.** |

## **37. 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:**

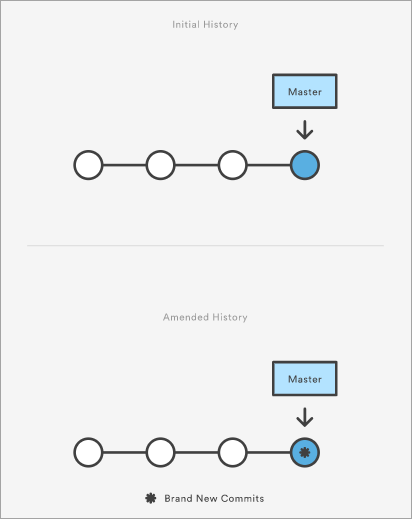
**git revert <commit id>**

## **38. How will you 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.

**To fix a broken commit or to change the last commit, the most convenient method is to use the command *git commit -amend***

**It allows you to combine staged changes with the previous commit as an alternative for creating an entirely new commit. This replaces the most recent commit with the amended commit.**

**[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/broken-commit.png)**

**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.**

## 39. Name some Basic & Advanced Operations in Git.

**Some basic operation in Git include:**

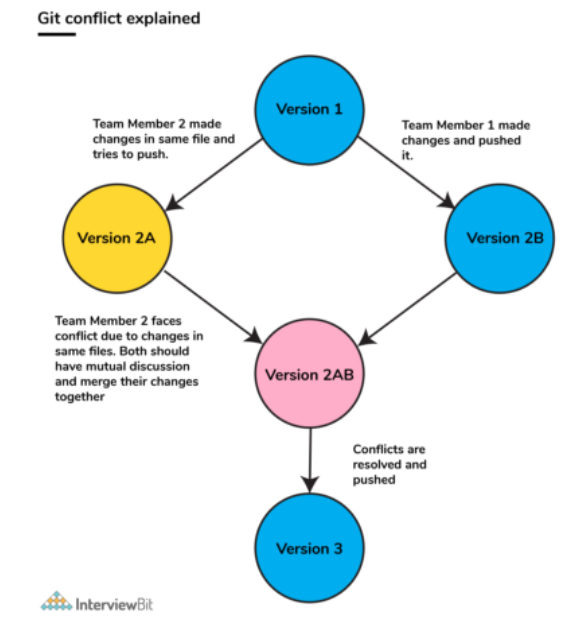
* Initialize
* Add
* Commit
* Push
* Pull
* Branching
* Merging
* Rebasing

## **40. What is a ‘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.

Git can handle on its own most merges by using its automatic merging features. There arises a conflict when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts are most likely to happen when working in a team environment.

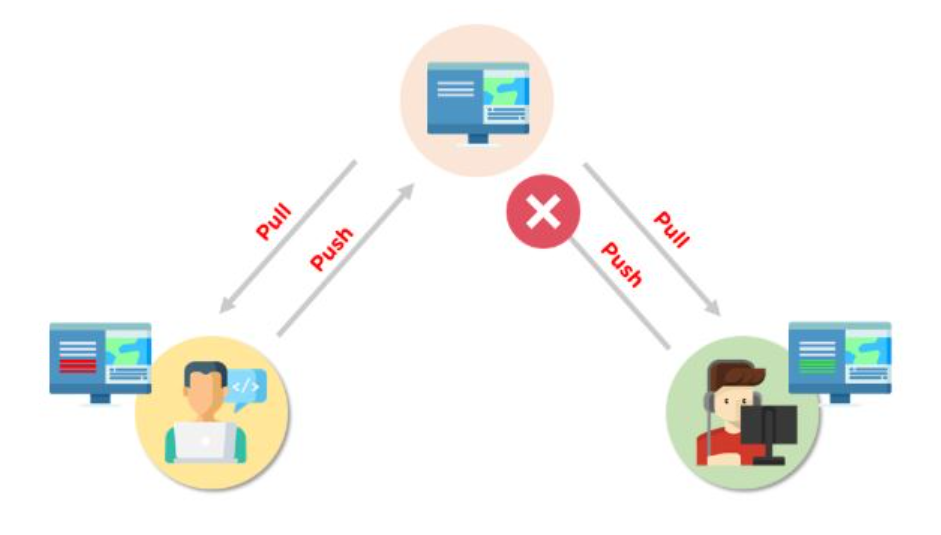
These conflicts have to be solved manually after discussion with the team as git will not be able to predict what and whose changes have to be given precedence.



A merge conflict can occur during merging a branch, rebasing a branch, or cherry-picking a commit. Once a conflict is detected, Git highlights the conflicted area and asks you to resolve it. Once the conflict is resolved, you can proceed with the merge.

**Follow the below steps to resolve a competing line change merge conflict:**

1. Open Git Bash (Git command line).
2. Use ***cd <repository-name>*** command to go to the local Git repository which is having the merge conflict.
3. Use the ***git status*** command to produce the list of files affected by the merge conflict.
4. Open the text editor that you use and traverse to the file that has merge conflicts.
5. To see the start of the merge conflict in your file, look the document for the conflict marker <<<<<<<. At the point when you open the file, you’ll observe the modifications from the HEAD or base branch after the line <<<<<<< HEAD. Then, you’ll observe =======, which partitions your modifications are from the modifications in the other branch, trailed by >>>>>>> BRANCH-NAME.
6. Choose in the event that you need to keep just your branch’s changes, just keep the other branch’s changes, or make a fresh change, that may include changes from the two branches. Erase the conflict markers <<<<<<<, =======, >>>>>>> and do the changes that you need in the final merge.
7. Use ***git adds.*** command to add or stage your changes.
8. Finally, use the ***git commit -m “message”*** command to commit your changes with a comment.



* 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.

## **41. 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.

It is a script through which you can instantly browse your working Git repository in a web browser.

This script sets up gitweb and a webserver to browse the local repository. It automatically directs a web browser and runs a web server through an interface into your local repository.

## **42. What does ‘hooks’ consist of in git?**

**This directory consists of Shell scripts which are activated after running the corresponding Git commands.  For example, git will try to execute the post-commit script after you run a commit.**

**Git hooks are certain scripts that Git runs before or after an event like commit, push, update or receive. You will find the ‘hooks’ folder inside .git directory in your local repository. You will find the build-in scripts here pre-commit, post-commit, pre-push, post push.**

**These scripts get executed locally before or after the occurrence of an event. You can also modify these scripts according to your needs and Git will execute the script when that particular event occurs.**

## **43. What is the use of ‘git log’?**

**To find specific commits in your project history- by author, date, content or history ‘git log’ is used.**

## **44. What is ‘git add’ is used for?**

***git add* adds file changes in your existing directory to your index.**

* This command adds files and changes to the index of the existing directory.
* You can add all changes at once using git add . command.
* You can add files one by one specifically using git add <file\_name> command.
* You can add contents of a particular folder by using command

git add /<folder\_name>/

## **45. What is the function of ‘git reset’?**

**The function of *Git Reset* is to reset your index as well as the working directory to the state of your last commit.**

## **46. What is git is-tree?**

***git is-tree* represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.**

**The git ls-tree command is used to list the contents of a tree object.**

## **47. 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:**

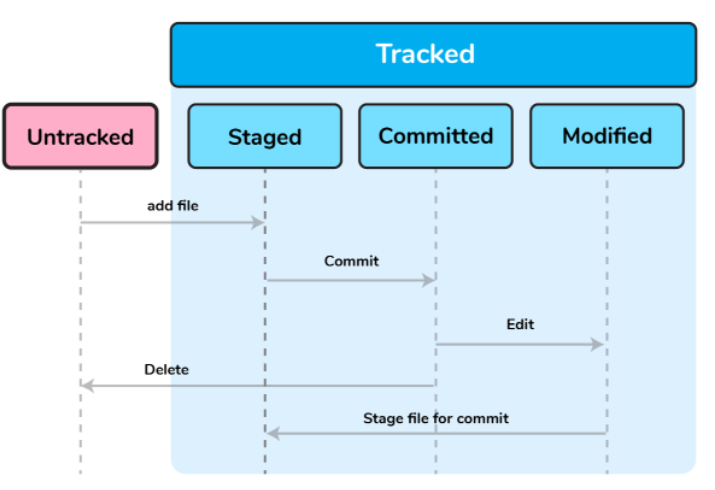
***git revert --no-commit HEAD~3...HEAD***

## 48. 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. It is helpful in understanding a git more comprehensively.

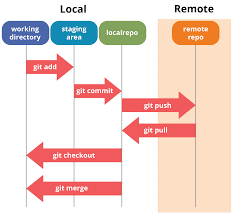
## 49. Define “Index”.

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’.

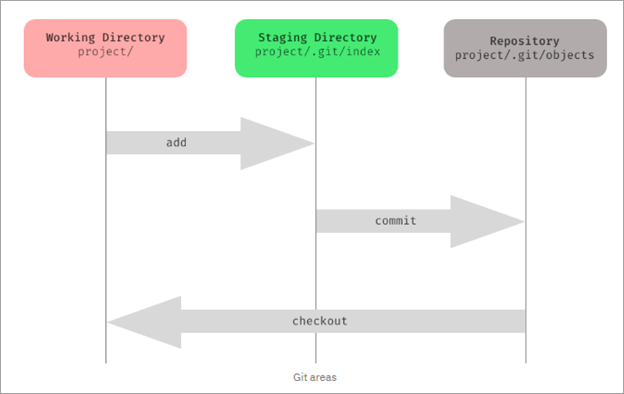


In the above image, the “staged” status indicates the staging area and provides an opportunity for the people to evaluate changes before committing them

From the diagram it is evident that every change is first verified in the staging area I have termed it as “stage file” and then that change is committed to the repository.



From Git’s perspective, there are three areas where the file changes can be kept i.e. working directory, staging area, and repository.

[[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/of-staging-area-or-indexing-in-Git.png)](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/of-staging-area-or-indexing-in-Git.png)

First, you make changes in your project’s working directory stored on your computer file system. All the changes remain here until you add them to an intermediate area called staging area.

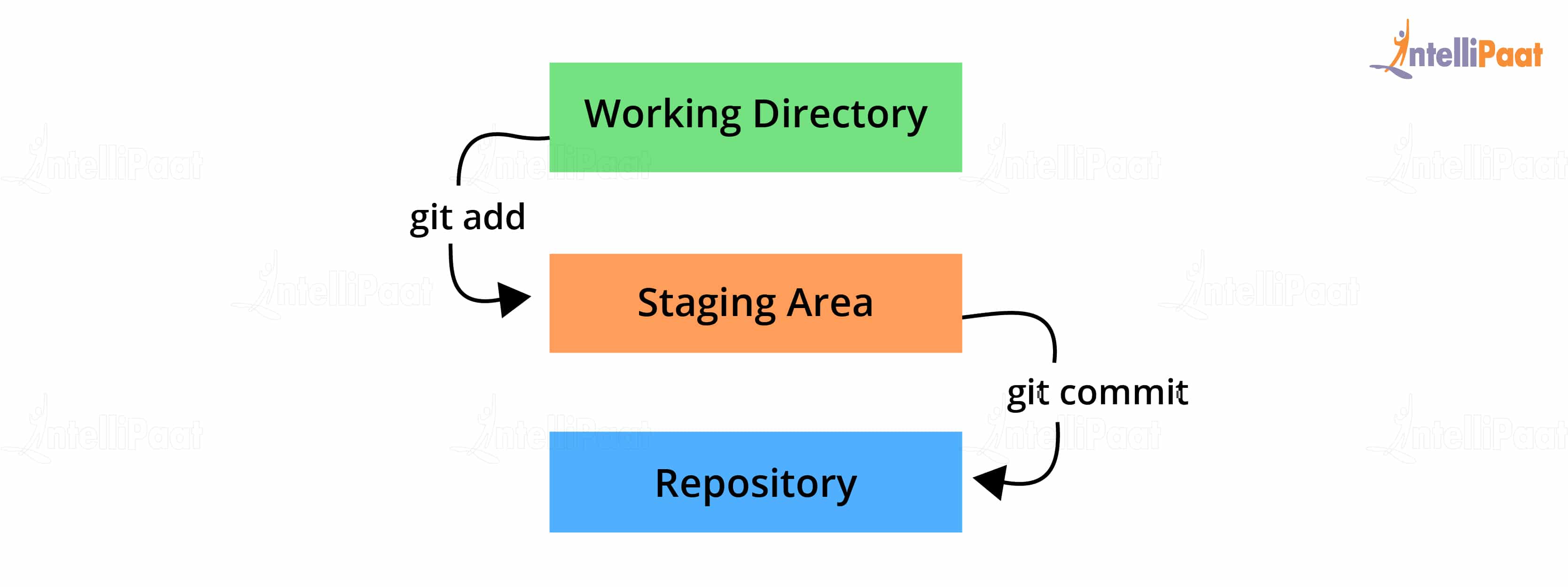
You can stage the changes by executing git add . command. This staging area gives you a preview of your next commit and basically lets you fine-tune your commits. You can add or remove changes in the staging area until you are satisfied with the version you are going to commit.

Once you verify your changes and sign off the stage changed, then you can finally commit the changes. Upon commit, they go the local repository i.e. into .git/objects directory.

Staging is also referred to as indexing because git maintains an index file to keep track of your file changes across these three areas. The files which are staged are currently in your index.

When you add changes to the staging area, then the information in the index gets updated. When you do a commit, its actually what’s in the index that gets committed, and not what’s in the working directory. You can use the git status command to see what’s in the index.

When we are making the commits, we can make changes to it, format it and review it in the intermediate area known as ‘Staging Area’ or ‘Index’.



## **50. What work is restored when the deleted branch is recovered?**

**The files which were stashed and saved in the stash index list will be recovered back. Any untracked files will be lost. Also, it is a good idea to always stage and commit your work or stash them.**

**If you want to fetch the log references of a particular branch or tag then run the command –**

**git reflog <ref\_name>**

**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.**

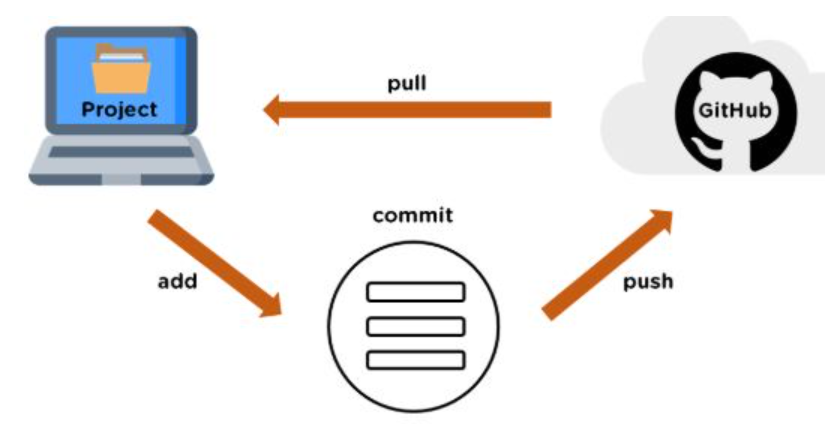
**This command tracks every single change made in the repository references (that can be branches or tags) and also maintains the branches/tags log history that was either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, 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.**

## 50. What is the function of ‘GIT PUSH’ in GIT?

GIT Push updates remote refs along with associated objects.

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.



## 51. 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**

So, a git pull brings the local branch up-to-date with its remote version, whereas a git fetch does not directly change your own local branch or working copy under *refs/heads.*

Git fetch can be used to update your remote-tracking branches under **refs/remotes/<remote>/.**

|  |  |
| --- | --- |
| Git Fetch | Git Pull |
| The Git fetch command only downloads new data from a remote repository. | Git pull updates the current HEAD branch with the latest changes from the remote server. |
| It does not integrate any of these new data into your working files. | Downloads new data and integrate it with the current working files. |
| Command - git fetch origin  git fetch --all | Tries to merge remote changes with your local ones.  Command - git pull origin master |

| **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. |

git pull = git fetch + git merge

Git pull command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

Git fetch is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

Git pull = git fetch + git merge

## **52. What is the git push command?**

**The git push command is applied for uploading content to a remote repository from a local repository. Pushing can overwrite changes, so it should be used with caution.**

## **53. What is the git pull command?**

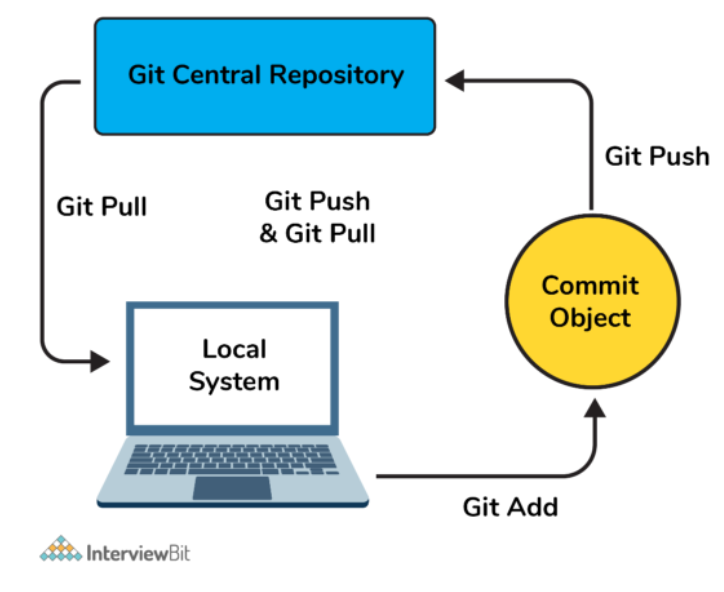
**The git pull command is for fetching and downloading content from a remote repository and integrating it with a local repository.**

## 54. Can you give differences between “pull request” and “branch”?

| **pull request** | **branch** |
| --- | --- |
| This process is done when there is a need to put a developer’s change into another person’s code branch. | A branch is nothing but a separate version of the code. |

## 55. 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.



## 56. What is the function of ‘git stash apply’?

When you want to continue working where you have left your work, **git stash apply** command is used to bring back the saved changes onto the working directory.

**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.

## 57. What is GIT stash?

GIT stash captures the current state of the working directory and index and keeps it on the stack for future use. It reverts the uncommitted changes (both staged and unstaged) from your working directory and returns you a clean working tree.

You can work on something else now, and when you come back, you can re-apply these changes. So, if you want to switch from one context to another without losing your current changes, then you can use stashing.

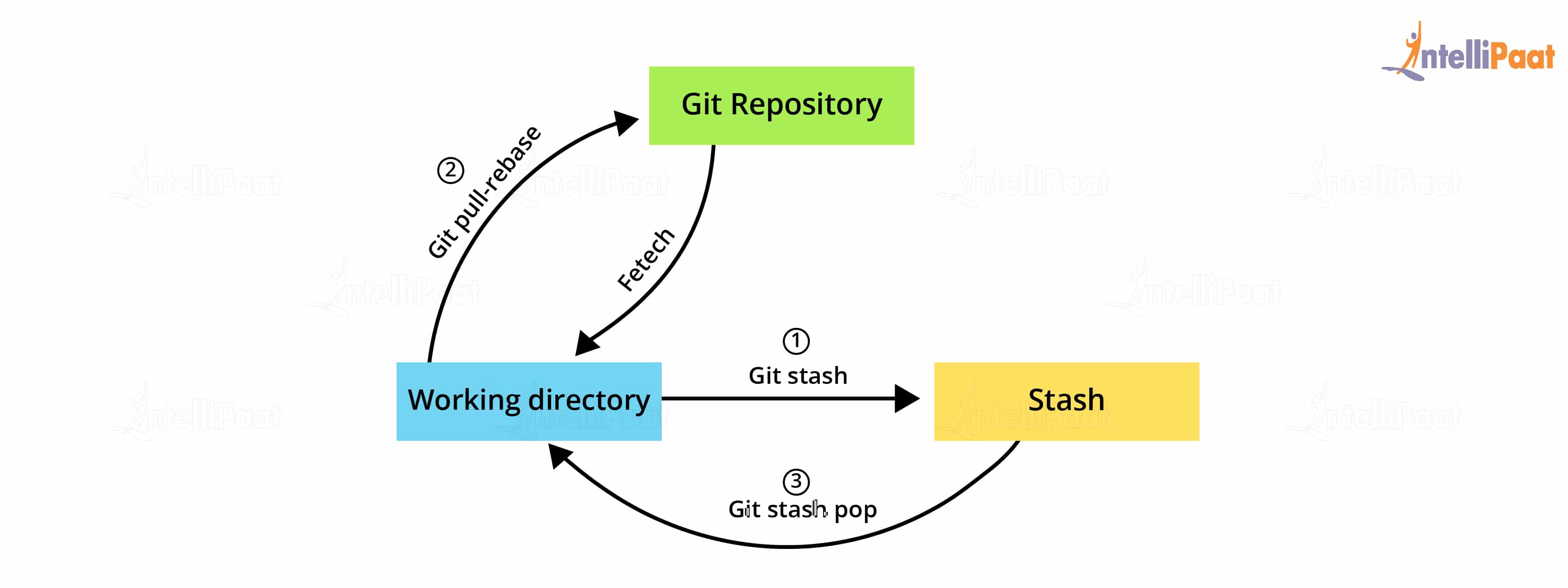
Git stash can be used in cases where we need to switch in between branches and at the same time not wanting to lose edits in the current branch. Running the

**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.

Stashing takes your working directory that is, your modified tracked files and staged changes and saves it on a stack of unfinished changes that you can reapply at any time.

The Git stash will take the working directory in the current state and index it to put on the stack at a later stage so that what we get is a clean working directory.



## 58. What is GIT stash drop?

When you no longer require a specific stash, you can remove it by executing **git stash drop <stash\_id> command**.

If you want to remove all the stashes in one go from the repository then you can run **git stash clear command**.

Now give an example.

If you want to remove a particular stash item from the list of stashed items you can use the below commands:

**git stash list:**

It will display the list of stashed items like:

stash@{0}: WIP on master: 049d078 added the index file  
stash@{1}: WIP on master: c264051 Revert “added file\_size”  
stash@{2}: WIP on master: 21d80a5 added number to log

If you want to remove an item named stash@{0} use command

**git stash drop stash@{0}**.

## **59. What is Git stash apply? How is it different from Git stash pop?**

Both the commands are used to reapply your stashed changes and start working from where you had left.

In **git stash apply**command, the changes will be re-applied to your working copy and will also be kept in the stash. This command can be used when you want to apply the same stashed changes to multiple branches.

In **git stash pop** command, the changes are removed from the stash and are re-applied to the working copy.

**git stash pop = git stash apply + git stash drop**

* 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

## 60.   What is the purpose of branching in GIT?

The purpose of branching in GIT is that you can create your own branch and jump between those branches. It will allow you to go to your previous work keeping your recent work intact.

## 61. What is the common branching pattern in GIT?

The common way of creating branch in GIT is to maintain one as “Main“

branch and create another branch to implement new features. This pattern is particularly useful when there are multiple developers working on a single project.

The common branching pattern is based on the git-flow.

It has two main branches i.e. master and development.

* The master branch contains the production code. All the development code is merged into the master branch at some point in time.
* The development branch contains the pre-production code. When the features are completed, they get merged to the master branch, generally through a CI/CD pipeline.
* **Feature Branches/Topic Branches:** They are used to develop new features for upcoming releases. It may branch off from the develop branch and must be merged back into the develop branch. Generally, these branches exist only in developer repositories, and not in origin.
* **Hotfix Branches:** They are used for unplanned production release when there is a need to fix any critical bug immediately in the live prod version. They may branch off from master and must be merged back into develop and master.
* **Release Branches:** They are used for the preparation of new production release. The release branch lets you do minor bug fixes and prepare metadata for release. They may branch off from development and must be merged back into master and develop.

## 62. How can you bring a new feature in the main branch?

To bring a new feature in the main branch, you can use a command “git merge” or “git pull command”.

## 63. To delete a branch what is the command that is used?

Once your development branch is merged into the main branch, you don’t need

development branch.

To delete a branch 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.

## 64. What is another option for merging in git?

“Rebasing” is an alternative to merging in git.

## 65. What is the syntax for “Rebasing” in Git?

The syntax used for rebase is

**git rebase [new-commit]**

In simple words, *git rebase* allows one to move the first commit of a branch to a new starting location. For example, if a feature branch was created from master, and since then the master branch has received new commits, *git rebase* can be used to move the feature branch to the tip of master. The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative ease and sometimes as a simple fast-forward operation.

Rebasing is the reapplying of commits on top of another base trip. A sequence of commits is applied from distinct branches into the final commit. It is a linear process of merging and an alternative to the git merge command. Rebasing makes it seem like one has created a branch from a different commit.

## **66. Explain git checkout in Git.**

Git checkout allows for the switching of the HEAD. It can be used to restore the historic versions of files as well. The command operates upon files, commits, and branches.

## 67. What does the git reset --mixed and git merge --abort commands do?

**Git reset** is a powerful command for undoing local changes to the state of a Git repo. This command resets the current HEAD to the specified stage.

It resets both the index and the working directory to the state of your last commit. Git reset has three modes i.e. soft, hard and mixed. The default mode of operation is mixed.

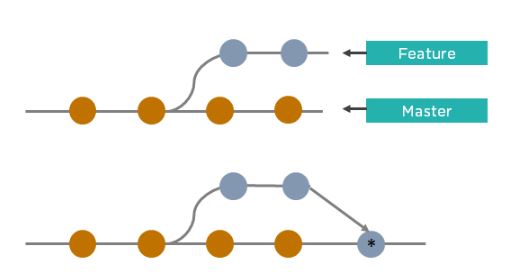
**git reset --mixed** is used to undo changes made in the working directory and staging area.

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

Rebasing is the reapplying of commits on top of another base trip. A sequence of commits is applied from distinct branches into the final commit. It is a linear process of merging and an alternative to the git merge command. Rebasing makes it seem like one has created a branch from a different commit.

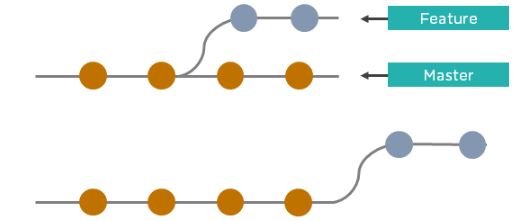
## **68. What is the difference between rebasing and merge in Git?**

* 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.
* In git rebase, a feature branch is moved into a master. Git merge maintains the history by adding a new commit.
* Git merge is used to incorporate new commits into your feature branch.

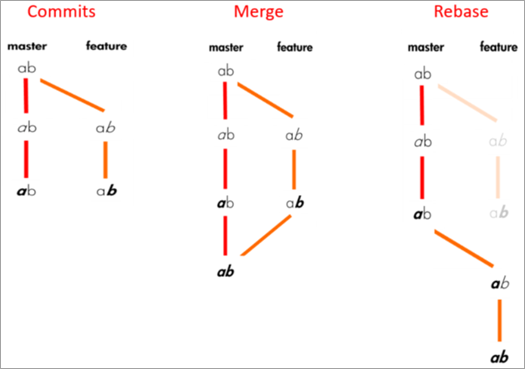


* Git merge creates an extra merge commit every time you need to incorporate changes.
* It pollutes your feature branch history.

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



* Git rebase Incorporates all the new commits in the master branch.
* It rewrites the project history by creating brand new commits for each commit in the original branch

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/01/rebase-and-merge.png)

## **69. What is revert in Git?**

The **git revert** command is a forward-moving undo operation. It is a safe way to undo changes as it will create a new commit that inverses the changes instead of deleting or orphaning commits in the commit history.

## **70. What is the difference between resetting and reverting?**

**While git reset changes the state of the branch to a previous one by removing all of the states after the desired commit,**

**The function of Git Reset is to reset your index as well as the working directory to the state of your last commit.**

**The syntax for rebase command is git rebase [new-commit]**

**git revert does it through the creation of new reverting commits and keeping the original one intact.**

| **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.** |

* **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.**

|  |  |
| --- | --- |
| **Reverting** | **Resetting** |
| **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.** | **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.** |

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

**git remote add**  just creates an entry in your git config that specifies a name for a particular URL.  Git remote adds a reference to a remote repository for further tracking

**git clone** creates a new git repository by copying and existing one located at the URI.

## **72. In Git, how would you return a commit that has just been pushed and made open?**

**One or more commits can be reverted through the use of git revert. This command, in a true sense, creates a new commit with patches that cancel out the changes introduced in specific commits. If in case the commit that needs to be reverted has already been published or changing the repository history is not an option then in such cases, git revert can be used to revert commits. If you run the following command then it will revert the last two commits:**

**git revert HEAD~2..HEAD**

**Alternatively, there is always an option to check out the state of a particular commit from the past and commit it anew.**

| **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.** |

## **73. How to remove a file from git without removing it from your file system?**

One has to be careful during a **git add**, else you may end up adding files that you didn’t want to commit. However, **git rm** will remove it from both your staging area (index), as well as your file system (working tree), which may not be what you want.

Instead, use git reset:

**git reset filename**          # or

**echo filename >> .gitingore** # add it to .gitignore to avoid re-adding it

This means that **git reset <paths>** is exactly the opposite of **git add <paths>**.

## **74. Can you explain the Gitflow workflow?**

To record the history of the project, Gitflow workflow employs two parallel long-running branches – master and develop:

* **Master** – this branch is always ready to be released on LIVE, with everything fully tested and approved (production-ready).
* **Hotfix** – these branches are used to quickly patch production releases. These branches are a lot like release branches and feature branches except they’re based on master instead of develop.
* **Develop** – this is the branch to which all feature branches are merged and where all tests are performed. Only when everything’s been thoroughly checked and fixed it can be merged to the master.
* **Feature** – each new feature should reside in its own branch, which can be pushed to the develop branch as their parent one.

## **75. Can you recover a deleted branch in Git?**

**Yes, you can. To recover a deleted branch, you should know the SHA off the top of your head. SHA or hash is a unique ID that Git creates with every operation.**

**When you delete a branch, you get the SHA displayed on the terminal:**

**Deleted branch <your-branch-name> (was <sha>)**

**You can use the below command to recover the deleted branch:**

**git checkout -b <your-branch-name> <sha>**

**If you don’t know the SHA for the commit at the tip of your branch then you can first use the git reflog command to know the SHA value and then apply the above checkout command to restore your branch.**

## **76. Tell me the difference between HEAD, working tree and index, in Git.**

* The working tree/working directory/workspace is the directory tree of (source) files that you are able to see and edit.
* The index/staging area is a single, large, binary file in <baseOfRepo>/.git/index, which lists all files in the current branch, their SHA-1 checksums, timestamps, and the file name – it is not another directory which contains a copy of files in it.
* HEAD is used to refer to the last commit in the currently checked-out branch.

## **77. 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.**

## 78. 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>]**

## **79. What is git cherry-pick?**

The command git cherry-pick is normally used to introduce particular commits from one branch within a repository onto a different branch. Another common use is to forward- or back-port commits from a maintenance branch to a development branch. This is in contrast with other ways such as merge and rebase which normally apply many commits onto 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.

Git cherry-pick is a command that allows the picking of arbitrary Git commits by reference and adding them to the HEAD. Cherry-picking is the process of picking a commit from one branch and applying it to another. It helps in undoing changes.

**git cherry-pick <commit-hash>**

## **80. What is git cherry-pick? What are the scenarios in which git cherry-pick can be used?**

**Git cherry-pick** is a powerful command to apply the changes introduced by one or more existing commits. It allows you to pick a commit from one branch and apply it to another.

git cherry-pick commitSha is the command used for cherry-picking.

commitSha is the commit reference.

This command can be used for undoing changes. For instance, if by mistake you have made a commit to a wrong branch, then you can check out the correct branch and cherry-pick the commit to where it should belong.

It can also be used in team collaboration. There can be scenarios where the **same code needs to be shared between two components of the product**. In this case, if **one developer has already written that code, then the other one can cherry-pick the same**.

**Cherry-picking is also useful in bug hotfixes where a patch commit can be cherry-picked directly into the master branch to fix the issue as soon as possible**.

Cherry-pick uses a diff to find the difference between branches.

As a merge commit belongs to a different branch, it has two parents and two changesets.

For example, if you have merge commt ref 63ad84c, you have to specify -m and use parent 1 as a base:

git checkout release\_branch

git cherry-pick -m 1 63ad84c

## 81. One of your teammates accidentally deleted a branch, and has already pushed the changes to the central git repo. There are no other git repos, and none of your other teammates had a local copy. How would you recover this branch?

Check out the latest commit to this branch in the reflog, and then check it out as a new branch.

## **82. How will you remove a file from Git without actually removing it from your local filesystem?**

**You can use the ‘cached’ option for this:**

**git rm -rf –cached $FILES**

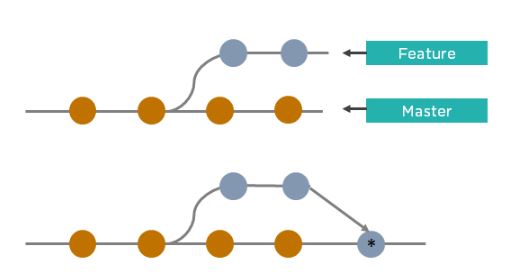
This command will remove the files from your repository without deleting them from your disk.

## 83. 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.

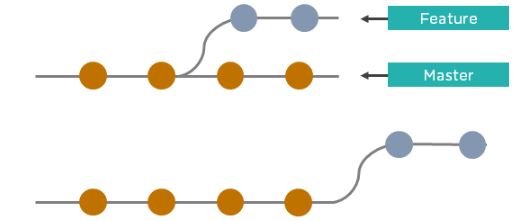
* Whenever in doubt, it is always preferred to use the git merge command.  
    
  Following are some factors that tell when to use merge and rebase commands:
* In case our branch gets contributions from other developers outside the team as in open-source or public repositories, then rebase is not preferred.  
  - 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.

Git merge is used to incorporate new commits into your feature branch.



* Git merge creates an extra merge commit every time you need to incorporate changes.
* It pollutes your feature branch history.

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



* Git rebase Incorporates all the new commits in the master branch.
* It rewrites the project history by creating brand new commits for each commit in the original branch

## 84. 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}**

If you want to write the new commit message from scratch use the following command  
**git reset -soft HEAD~N &&git commit**

If you want to start editing the new commit message with a concatenation of the existing commit messages then you need to extract those messages and pass them to Git commit.

**git reset -soft HEAD~N &&git commit -edit -m“$(git log -format=%B -reverse .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.

In Git, squashing commits means combining two or more commits into one.

Use the below command to write a new commit message from the beginning.

git reset -soft HEAD~N &&git commit

But, if you want to edit a new commit message and add the existing commit messages, then you must extract the messages and pass them to Git commit.

The below command will help you achieve this:

git reset -soft HEAD~N &&git commit -edit -m“$(git log -format=%B -reverse [.HEAD@{N}](mailto:.HEAD@%7bN%7d))”

## 85. 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.

## 86. How do you recover a deleted branch that was not merged?

To recover a deleted branch, first, you can use the git reflog command. It will list the local recorded logs for all the references. Then, you can identify the history stamp and recover it using the git checkout command.

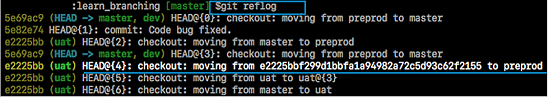
## **87. Can you tell something about git reflog?**

**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.**
* **The git log is a public record of the commit history for a branch. Reflog, on the other hand, is a private one of the repository’s local commits.**
* **Unlike reflog, the git log is a part of the Git repository and is replicated after a push, fetch, or pull. A developer cannot access a local repository’s reflog without having access to the computer where it is located.**
* **The ‘reflog’ command keeps a track of every single change made in the references (branches or tags) of a repository and keeps a log history of the branches and tags that were 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**](https://www.edureka.co/blog/what-is-git/)**and listed by the ‘reflog’ command.**
* ***Note:  The branch will be recoverable from your working directory only if the branch ever existed in your local repository i.e. 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 must be executed in the repository that had the lost branch. If you consider the remote repository situation, then you have to execute the reflog command on the developer’s machine who had the branch.**
* ***command: git reflog***

## **88. How to recover a deleted branch using git reflog?**

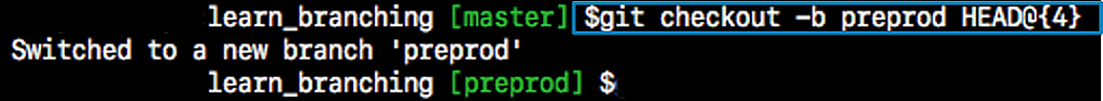
**Step 1**: **History logs of all the references**

Get a list of all the local recorded history logs for all the references (‘master’, ‘uat’ and ‘prepod’) in this repository.

**Step 2: Identify the history stamp**

As you can see from the above snapshot, the highlighted commit id: e2225bb along with the HEAD pointer index:4 is the one when ‘preprod’ branch was created from the current HEAD pointer pointing to your latest work.

**Step 3**: **Recover**

If you want to recover back the ‘preprod‘ branch then use the command  ‘git checkout’ passing the HEAD pointer reference with the index id – 4. This is the pointer reference when ‘preprod’ branch was created long commit id highlighted in the output screenshot.

I have included the frequently asked Git interview questions. If you have more questions in your mind just type it in the comment box below and we will reply you ASAP. Before going for the interview I will suggest you to [***check out this Git blog series***](https://www.edureka.co/blog/what-is-git/).

*If you found this****Git Interview Questions****relevant, check out the*[***DevOps training***](https://www.edureka.co/devops-certification-training)*by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka DevOps Certification Training course helps learners gain expertise in various DevOps processes and tools such as Puppet, Jenkins, Nagios and GIT for automating multiple steps in SDLC.*

## 89. What is Git Bisect and how do you use it?

The Git Bisect command performs a binary search to detect the commit which introduced a bug or regression in the project’s history.

Syntax: git bisect <subcommand> <options>

* Git bisect is used to find the commit that introduced a bug by using binary search. The command for Git bisect is  
  **git bisect <subcommand> <options>**
* Now since you have mentioned the command above explain to them what this command will do.
* This command uses a binary search algorithm to find which commit in your project’s history introduced a bug. You use it by first telling it a “bad” commit that is known to contain the bug, and a “good” commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is “good” or “bad”. It continues narrowing down the range until it finds the exact commit that introduced the change.

## 90. If you recover a deleted branch, what work is restored?

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

## 91. 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.

## 92. 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>

## **93. How do you integrate Git with Jenkins?**

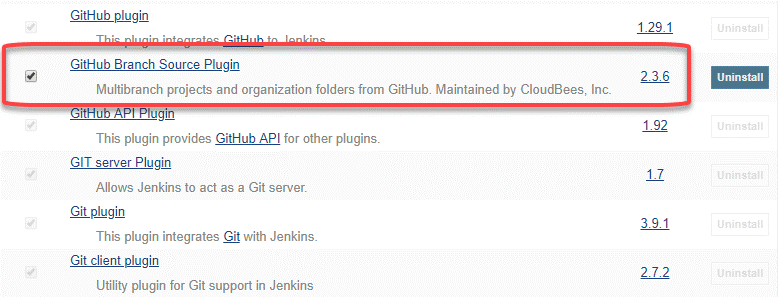
**Step 1**. Click on the manage jenkins button on your jenkins dashboard.



**Step 2**. Click on manage jenkins plugin.

**Step 3:**In the Plugins Page

1. Select the GIT Plugin
2. Click on **Install without restart.**The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option **Download now and Install after restart** In which plugin is installed after restart
4. You will be shown a “No updates available” message if you already have the Git plugin installed.

**Step 4**: Once the plugins have been installed, go to **Manage Jenkins** on your Jenkins dashboard. You will see your plugins listed among the rest.

## 94. 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.

## 95. 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

## 96. 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.

## 97. What do you understand about the Staging area in Git?

The Staging Area in Git is when it starts to track and save the changes that occur in files. These saved changes reflect in the .git directory. Staging is an intermediate area that helps to format and review commits before their completion.

## **98. What is the process for creating a repository in Git?**

If we want to create a repository in Git, then we need to run the command “git init”. With this command. git repository, we can create a directory in the project directory.

## **99. What is origin in Git?**

Origin refers to the remote repository that a project was originally cloned from and is used instead of the original repository’s URL. This allows for easier referencing.

## **100. What is the difference between git fetch and git pull?**

Git fetch retrieves new data from a remote repository but does not integrate it into our working files. It helps in checking if any changes happened in the remote repository. It does not manipulate or destroy anything in the process.

Git pull, on the other hand, updates the HEAD with the latest changes from the remote server and directly integrates it into the working copy files. Using git pull can end in merge conflict as it tries to merge remote changes with the local ones.

## **101. What is the difference between git reflog and log?**

The git log is a public record of the commit history for a branch. Reflog, on the other hand, is a private one of the repository’s local commits.

Unlike reflog, the git log is a part of the Git repository and is replicated after a push, fetch, or pull. A developer cannot access a local repository’s reflog without having access to the computer where it is located.

## **102. How to identify if a certain branch has been merged into master?**

Git branch –merged master – shows all branches that are merged into master

Git branch – merged – shows all branches that are merged into the head

Git branch – no-merged –shows all the branches that are not merged

## **103. Why do we need branching in GIT?**

With the help of branching, we can have our own branch and we can also jump between various branches. We can go to our previous work, at the same time keeping our recent work intact.

## 104. What’s the difference between reverting and resetting?

|  |  |
| --- | --- |
| Reverting | Resetting |
| 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. | 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. |

## 105. How can you discover if a branch has already been merged or not?

There are two commands to determine these two different things.

git branch --merged - Returns the list of branches that have been merged into the current branch.

git branch --no-merged - Returns the list of branches that have not been merged.

* 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.

## **104. In Git how do you revert a commit that has already been pushed and made public?**

There can be two approaches to tackle this question and make sure that you include both because any of the below options can be used depending on the situation:

* Remove or fix the bad file in a new commit and then push it to the remote repository. This is the most obvious way to fix an error. Once you have made necessary changes to the file, then commit it to the remote repository using the command: git commit -m “commit message”
* Also, you can create a new commit that undoes all changes that were made in the bad commit. To do this use the command

git revert <name of bad commit>

## 105. 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.

## 106. 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.

## 107. 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.

## 108. 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.

## 109. 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. |

## **110. What is Git bisect? How can you use it to determine the source of a (regression) bug?**

* Git bisect is used to find the commit that introduced a bug by using binary search. The command for Git bisect is  
  **git bisect <subcommand> <options>**
* Now since you have mentioned the command above explain to them what this command will do.
* This command uses a binary search algorithm to find which commit in your project’s history introduced a bug. You use it by first telling it a “bad” commit that is known to contain the bug, and a “good” commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is “good” or “bad”. It continues narrowing down the range until it finds the exact commit that introduced the change.

## **111. What is the regular way for branching in GIT?**

The best way to create a branch in GIT is to have one ‘main’ branch and then create another branch for implementing the changes that we want to make. This is extremely useful when there are a large number of developers working on a single project.

## **112. State a way to create a new branch in Git.**

If we want to create a new feature into the main branch, then we can use the command ‘git merge’ or ‘git pull’.