**GIT AND GITHUB TUTORIALS**

**GIT: Global Information TrackerS**

Git is a free and open-source version control system OR source code management that is used to manage source code and other files over time. It was created by Linus Torvalds in 2005 for managing the development of the Linux kernel

Git allows multiple developers to work on the same codebase simultaneously and keep track of changes made to the code over time. It stores each version of the code as a snapshot and allows users to easily switch between versions, collaborate on changes, and merge changes from multiple contributors into a single codebase.

**What is meant by version control system**?

Version control systems are software tools that help software teams manage changes to source code over time.

**Why we use git?**

Git is used for version control, which means it is used to manage changes made to a codebase or other files over time

**Before diving deep, let’s explain a scenario before Git:**

Developers used to submit their codes to the central server without having copies of their own

Any changes made to the source code were unknown to the other developers

There was no communication between any of the developers

**Now let’s look at the scenario after Git:**

Every developer has an entire copy of the code on their local systems

Any changes made to the source code can be tracked by others

There is regular communication between the developers

**Features of Git:**

1. Open source
2. Scalability
3. Security
4. speed

**Benefits of Git:**

**Saves Time**  
Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account and find out its features.

**Offline Working**  
One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.

**Undo Mistakes**  
One additional benefit of Git is we can **Undo** mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.

**Track the Changes**  
Git facilitates with some exciting features such as **Diff, Log,** and **Status**, which allows us to track changes so we can **check the status, compare** our files or branches.

Now, we will discuss some other points about why should we choose Git.

1. Speed
2. Secuirty
3. Integrity
4. Distributed
5. Clean History
6. Open Source
7. Scalable
8. Staging Area
9. Branching
10. Data Assurance
11. Workflow
12. Trendy
13. Local Repository
14. Collaboration

**What Is GitHub**

GitHub is a web-based platform for hosting and collaborating on software development projects that use the Git version control system. It provides a range of tools and features to help developers manage code repositories, collaborate with other developers, and track issues and bugs.

GitHub allows users to create and manage public and private repositories, which can contain source code, documentation, and other files. Users can also fork and clone repositories from other users, which allows them to make their own changes to the code and contribute back to the original project

**Why do you need GitHub?**

It's used for storing, tracking, and collaborating on software projects. It makes it easy for developers to share code files and collaborate with fellow developers on open-source projects. GitHub also serves as a social networking site where developers can openly network, collaborate, and pitch their work.

**What are GitHub's Features?**

Easy Project Management. ...

Increased Safety With Packages. ...

Effective Team Management. ...

Improved Code Writing. ...

Increased Code Safety. ...

Easy Code Hosting.

**Git Stages:**

In Git, there are three main stages or areas that are involved in tracking and managing changes to files in a repository. These stages are commonly referred to as the "Git stages" or the "Git workflow":

1. **Working Directory**: The working directory, also known as the working tree, is the area where you make modifications to your files. It represents the current state of your project's files.
2. **Staging Area (Index**): The staging area, also referred to as the index, is an intermediate area between the working directory and the repository. It acts as a holding area for changes that you want to include in the next commit. Files in the staging area are considered ready to be committed but have not been permanently recorded yet.
3. **Repository (Commit):** The repository, also known as the commit history or version history, is where Git permanently stores committed changes. It contains a complete history of all commits made to the project

**Repository:**

**Local Repository:** A local repository is a repository that exists on your computer. It is where you work on and track changes to your project. It is created and stored on your local machine, allowing you to make and save changes to your files.

**Remote Repository**: A remote repository is a repository that exists on a remote server or hosting platform, typically on the internet. It serves as a central storage location for your project and enables collaboration with others. Remote repositories allow multiple developers to contribute to a project, share code, and keep track of changes made by different team members.

In summary, a local repository is where you work on your project locally, while a remote repository is a shared repository that allows collaboration and code sharing with others

**Central repository** is a type of remote repository that serves as a centralized storage location for a project. It is often used in a version control system like Git to facilitate collaboration among multiple developers.

The central repository acts as the main copy of the project's codebase, where all team members can push their changes and retrieve the latest version of the code. It serves as a central point of reference for sharing, merging, and tracking changes made by different contributors.

**Git Cheat Sheet**

1. Git configuration

**Git config**  
Get and set configuration variables that control all facets of how Git looks and operates.  
**Set the name:**  
$ git config --global user.name "User name"  
**Set the email:**  
$ git config --global user.email "himanshudubey481@gmail.com"  
**Check the setting:**  
$ git config -list

**2. Starting a project**

**Git init**  
**Create a local repository:**  
$ git init

**Git clone**  
**Make a local copy** of the server repository.  
$ git clone

**3. Local changes**

* **Git add**  
  **Add a file** to staging (Index) area:  
  $ git add Filename [ add one files to staging]
* $ git add –A [ add all files to staging]  
  **Add all files** of a repo to staging (Index) area:  
  $ git add\*

**\* Git commit**  
**Record** or snapshots the file permanently in the version history **with a message**.  
$ git commit -m " Commit Message"

If we want to commit the file directly without adding into staging we use command

$ git commit –a –m “Message”

* **Git Remove**
* **Remove** Files from the working tree and from the index
* $ git rm
* **Remove** files from the Git But keep the files in your local repository:  
  $ git rm –cached

**\* Git checkout**

If any changes had done by mistake and we want to get back the same data again without any commit we use command

$ git checkout filename [ for one file]

$ git checkout –f filename [ for many files]

* **Git stash**
* Switch branches without committing the current branch. Stash current work:  
  $ git stash  
  Saving stashes with a message:  
  $ git stash save ""  
  Check the stored stashes:  
  $ git stash list

Re-apply the previous commits:  
 $ git stash pop

Delete a most recent stash from the queue:  
 $ git stash drop  
 Delete all the available stashes at once:  
 $ git stash clear

**\*Git Restore**

If file are already staged and you don’t want to add in staging those files

$ git restore –staged filename [for one file]

$ git restore –staged . [For all Files]

**4. Track changes**

**Git diff**  
Track the changes that have not been staged:

$ git diff  
Track the changes that have staged but not committed:  
$ git diff –staged

**Git status**  
Display the state of the working directory and the staging area.  
$ git status

**Git show Shows objects:**  
$ git show

**5. Commit History**

**Git log**  
Display the most recent commits and the status of the head:  
$ git log  
Display the modified files with location and n number of files  
$ git log –p –n numbers

**Git blame**  
Display the modification on each line of a file:  
$ git blame <file name>

**6. Branching**

1. Create a new branch: To create a new branch, you can use the command **git branch <branch\_name>**.
2. Switch to a branch: To switch to an existing branch, use the command **git checkout <branch\_name>**.
3. Create and switch to a new branch: You can combine branch creation and switching using the command **git checkout -b <branch\_name>.**
4. List branches: To list all branches in your repository, use the command **git branch**. The branch with an asterisk (\*) indicates the currently checked out branch.
5. Merge branches: To merge changes from one branch into another, you first need to switch to the target branch (**git checkout <target\_branch>**) and then use the command **git merge <source\_branch>**. For example, **git merge feature-branch** merges the changes from "feature-branch" into the current branch.
6. Delete branches: To delete a branch, use the command **git branch -d <branch\_name>**. For example, **git branch -d feature-branch** deletes the "feature-branch" branch.

**7. Pushing Updates**

**Git push**  
Transfer the commits from your local repository to a remote server. Push data to the remote server:

$ git remote add origin repository link

$ git push –u origin branch name

$ username: github username

$ password: token link

**8. Ignoring files**

**.gitignore**  
it will be useful when you don’t want to track some specific files them we use a file called .gitignore  
$ vi .gitignore

**GIT Interview Questions**

**1) what is the difference between centralized version control system and distributed version control system**

A) in centralized version control system you have the client and server architecture and server which is a remote repository has all the copies of your code like all the versions of the code are only available with the server

whereas in the distributed version control system every developer has all the versions of the code like all the copies of the code are which each and every developer who has cloned the remote repository so that's why it's called distributed

version control system because everything is distributed

**2) Different types of version control systems**

**Local version control** systems have a database that stores all file changes under revision control on disc in a special format.

**Centralized version control** systems have a single repository, from which each user receives their working copy.

**Distributed version control** systems contain multiple repositories, and different users can access each one with their working copy.

**3) What is a version control system (VCS)?**

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

**4) What is Git, and what are its key features?**

Git is a distributed version control system designed to track changes in files and coordinate work among multiple developers.

Key features of Git include distributed development, branching and merging, speed and efficiency, and the ability to handle large projects efficiently.

**5) What is GitHub?**

GitHub is a web-based platform that provides hosting for Git repositories. It offers additional features such as bug tracking, task management, wikis, and pull requests, making it easier for developers to collaborate on projects.

**6) What is the difference between Git and GitHub?**

Git is a version control system that allows you to track changes and manage code repositories.

GitHub is a web-based platform that provides hosting for Git repositories, offering additional collaboration and project management features.

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

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

**8) what are the repository types?**

a local repository is the copy of a project on your computer, a central repository is a single location for collaboration, and a remote repository is a hosted version of the project accessible through a server. Local repositories are where developers make changes, central repositories are used for coordination, and remote repositories facilitate sharing and collaboration among team members.

**9) 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.

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

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

**11) What is a ‘conflict’ in 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.

**12) Explain the difference between rebasing and merge in Git?**

• Git rebase is a command that allows developers to integrate changes from one branch to another.  
• Git merge is a command that allows you to merge branches from Git.

Git rebase and merge both integrate changes from one branch into another. Where they differ is how they used. Git rebase moves a feature branch into a master. Git merge adds a new commit, preserving the history.

(If you’re working alone or on a small team, use rebase. If you’re working with a big team, use merge.)

**(OR)**

so if you want a linear history you use git rebase whereas if you are not bothered about the git history you can simply do git merge

**13). Have you faced the situation where you resolve conflicts in Git? How?**

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches. Here are the steps that will help you resolve conflicts in Git:  
1. The easiest way to resolve a conflicted file is to open it and make any necessary changes  
2. After editing the file, we can use the git add a command to stage the new merged content  
3. The final step is to create a new commit with the help of the git commit command  
4. Git will create a new merge commit to finalize the merge

**14) How can you create a repository in Git?**

This is probably the most frequently asked question and the answer to this is really simple.

To create a repository, create a directory for the project if it does not exist, then run the command “**git init**”. By running this command .git directory will be created in the project directory.

**15) 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.

**16) 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’.

**17) 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.

**18) 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.

**19) What is the purpose of GIT stash?**

GIT stash takes the present state of the working file and index and puts in on the stack for next and gives you back a clean working file. So in case if you are in the middle of object and require to jump over to the other task, and at the same time you don't want to lose your current edits, you can use GIT stash.

**20) What does the git push command do?**

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.

**21) What does git pull origin master do?**

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

**22) 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

**Git Fetch:**

git fetch is like asking your friend if there are any updates on a project without actually incorporating those updates into your work immediately.

It retrieves changes from the remote repository (like GitHub) to your local repository but doesn't automatically merge them into your current branch.

Think of it as checking if there's anything new in the project, but you're not actively making any changes to your own work.

Example: Suppose you're working on a project, and you want to see if your colleagues have made any changes to the remote repository. You run git fetch to check for updates:

This command will fetch any new changes from the remote repository and store them in your local repository but won't change your current working branch.

**Git Pull:**

git pull is like asking your friend if there are any updates on a project and immediately merging those updates into your work if there are any.

It combines git fetch and git merge into one command. It fetches changes from the remote repository and then automatically merges them into your current branch.

Think of it as checking for updates and, if there are any, integrating them into your own work.

Example: Suppose you've previously fetched the changes using git fetch, and you now want to incorporate those changes into your current branch. You can use git pull:

This command fetches the latest changes from the remote repository and merges them into your current branch, updating your local working copy with the latest changes from the remote repository.

In summary, git fetch is a way to check for updates from the remote repository without immediately making changes to your local work. git pull is a way to both check for updates and integrate them into your current branch in one step. The choice between the two depends on whether you want to inspect the changes before merging (use fetch) or directly update your work with the latest changes (use pull).

**23) what is .git and .gitignore in simple way**

**.git**: The **.git** directory is the hidden directory at the root of a Git repository. It's like the brain and memory of your Git project. It contains all the information about your project's history, branches, commits, and more. You should never modify this directory manually unless you really know what you're doing. It's crucial for Git to work and maintain your project's history.

**.gitignore**: The **.gitignore** file is a text file that specifies which files and directories should be ignored by Git. This means Git won't track or include them in your version control. You create and edit the **.gitignore** file to exclude things like temporary files, build artifacts, and sensitive information (like passwords) from being committed to your repository. It helps keep your repository clean and prevents irrelevant files from cluttering your version control history.

**24) what are pre commit hooks and what are post comment hooks**

**Pre-commit hooks** are like a checkpoint in Git before you make a commit. They are scripts or actions that run automatically just before you create a new commit. You can use pre-commit hooks to perform checks or tasks like code formatting, linting, or running tests. If the hook detects any issues, it can prevent the commit from happening until the issues are fixed.

**Post-commit hooks** are actions that run after you've successfully made a commit. They are like a follow-up to a commit. You can use post-commit hooks for tasks like sending notifications, triggering automated deployment, or updating issue tracking systems. These hooks help you automate processes that need to happen after code changes are committed.

**25) What is webhook**

Webhooks are commonly used to automate tasks or trigger actions in response to events like code pushes, pull requests, issues, and more. These events can be crucial for continuous integration, deployment, and collaboration workflows.