Git is a distributed version control system (DVCS) commonly used for software development and other version control tasks.

**key concepts related to Git:**

**1.Version Control:** Git allows multiple people to work on projects simultaneously. It keeps track of changes in the source code over time. This means you can revert back to a previous stage, compare changes over time, and see who last modified something that might be causing an issue.

**2.Distributed System:** Unlike centralized version control systems, Git is distributed. This means that every user's copy of the code is also a repository that can contain the full history of all changes.

**3.Commits:** In Git, a commit is a snapshot of your repository at a specific point in time. Commits are the building blocks of Git repositories.

**4.Branches**: Git allows you to create branches, which are essentially separate lines of development. You can work on a new feature or bug fix without affecting the main codebase. Branches can later be merged back into the main codebase.

**5.Merging**: Merging in Git refers to integrating changes from one branch into another. This is a common practice when working on collaborative projects.

**6.Pull Requests**: In platforms like GitHub and GitLab, pull requests are a way to propose changes to a repository. They allow contributors to notify others about changes they've pushed to a repository on GitHub.

**7.Remote Repositories**: Git allows you to work not only on your local machine but also on remote servers or other Git repositories. You can push changes to these remote repositories or pull changes from them.

* Create a folder in your machine
* Create some files in the same folder
* Put some code in the file

public class Main {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

* Convert folder into working directory **git init**
* Open GIT Bash from the created folder and run the command **git init**

**git init**: Initializes a new Git repository in the current directory.

* Configure User Name

**git config --global user.name "Sekhars604“**

* Configure Email ID

**git config --global user.email** [**sekharstivoli@gmail.com**](mailto:sekharstivoli@gmail.com)

* To check the configurations

**git config --global --list**

**In Git, there are three main types of files:**

**1.Tracked Files:**

**Committed Files:** These are files that are tracked and have been included in a Git commit. They are part of the project's history.

**Staged Files:** These are changes to tracked files that have been marked to be included in the next commit. You stage files using git add.

**Unmodified Files:** Files that are tracked and have not been changed since the last commit.

**2.Untracked Files:**

These are files in your working directory that are not part of the Git repository. Git doesn't track them, and they won't be included in commits until you explicitly use **git add** to stage them.

**3.Ignored Files:**

These are files that you've told Git to deliberately ignore. You specify these files or patterns in a .gitignore file in your repository. Ignored files are typically build artifacts, logs, or files that contain sensitive information (like API keys) and should not be included in the version control system.

Here's a brief overview of how these types of files interact in a typical Git workflow:

* You start with untracked files in your working directory.
* You use git add to stage specific changes, making them staged files.
* When you're ready, you use git commit to create a new commit with the staged changes, turning them into committed files.
* Unmodified files remain unmodified unless you make changes to them.
* Ignored files are excluded from tracking and commits altogether.
* By understanding these file types and how they relate to Git's workflow, you can effectively manage your code and version control.
* Check file status

**git status**

* Move one file to Staged Area

**git add**

\*\***Note: File Name is case sensitive**

• Check file status

**git status**

* Move all the file Staged Area

**git add .**

• Check file status

**git status**

• Move all the file from staged area to local repository

**git commit -m "message”**

\*\* Note: There is no option we can commit few files

* List of commits

**git log**

• Another command to check All Commits

**git log --oneline**

* Restore file from staged area to working directory

**git restore --staged <file name>**

* Revert Latest Commit

**git revert HEAD --no-edit**

Note: System will delete the files using the revert latest commit

* Check all Commits

**git log --oneline**

• Note down the commit Hash

* Git Reset

**git reset <Commit Hash>**

GIT Branch

* To create branch

**git branch <branch Name>**

* to check the branch

**git branch**

* To change the branch

**Git checkout <branch name>**

* To merge

**Git merge >branchname>**

* To delete the branch

**Git branch -d <branch name>**

* To copy files from one branch to another branch

**Git checkout <branchname> <filename>**

**\*\*Note : When you copy files from another branch file directly in stagged area**

**Git ignore :**

When sharing your code with others, there are often files or parts of project, you do not want to share. Git ignore file is only one for working directory & apply to all branch.

Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

Create gitignore file : **touch .gitignore**

Mention the files or directories you want to ignore in .gitignore file (ex : want to exclude all .txt files. \*.txt). It will exclude all .txt files.

**Git rebase**

git rebase is a Git command used to change the base of a branch. It allows you to move or combine a series of commits from one branch to another branch.

**git rebase master**

**Git Rearrange**

Git Rearrange is used to reorder the commits.

**git rebase -i HEAD~2**

**Git merge commit**

A merge commit is just like another commit, the state of your repository at a given point in time plus the history it evolved from.

Note: We can’t merge the first commit.

• Note: It always merge with the above commit.

**git rebase -i HEAD~3**

**Once file opens change pick to squash**

**git log --oneline**

**git show --name-only --oneline <Commit-Hash>**

**Git Certain commit**

Git certain commit used to do The specific existing commits to include in another branch

Note down the Commit Hash for those commit we want to merge with master branch.

Checkout to master branch

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