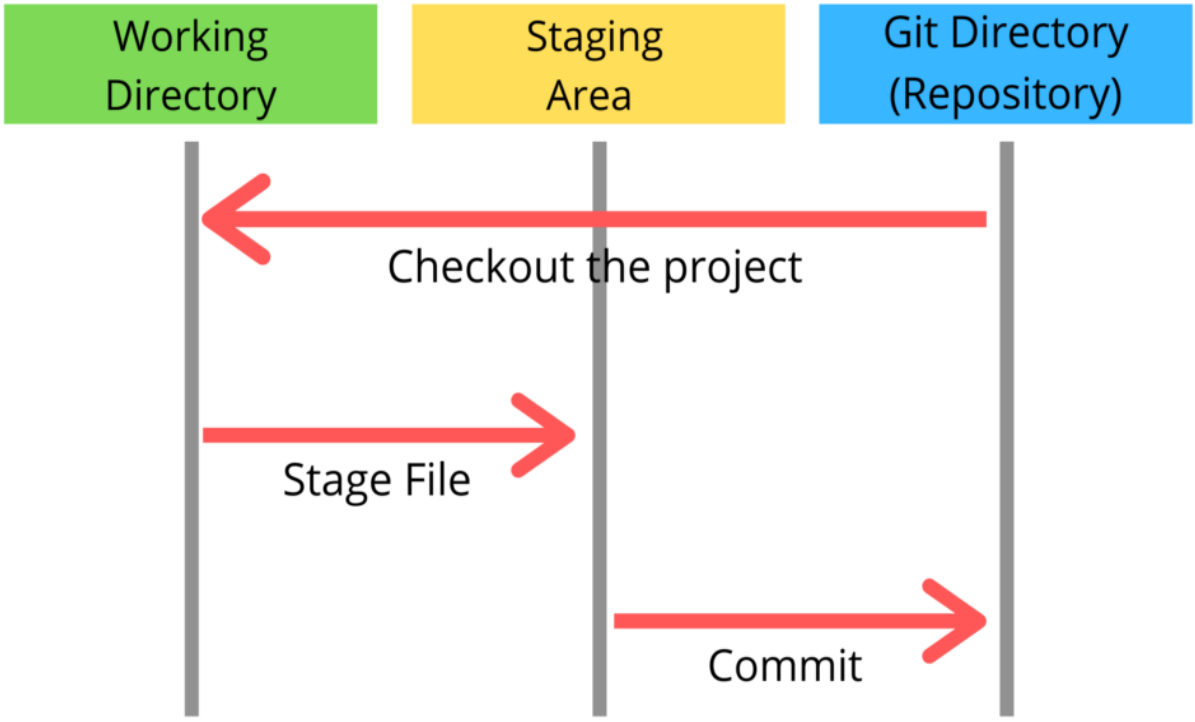
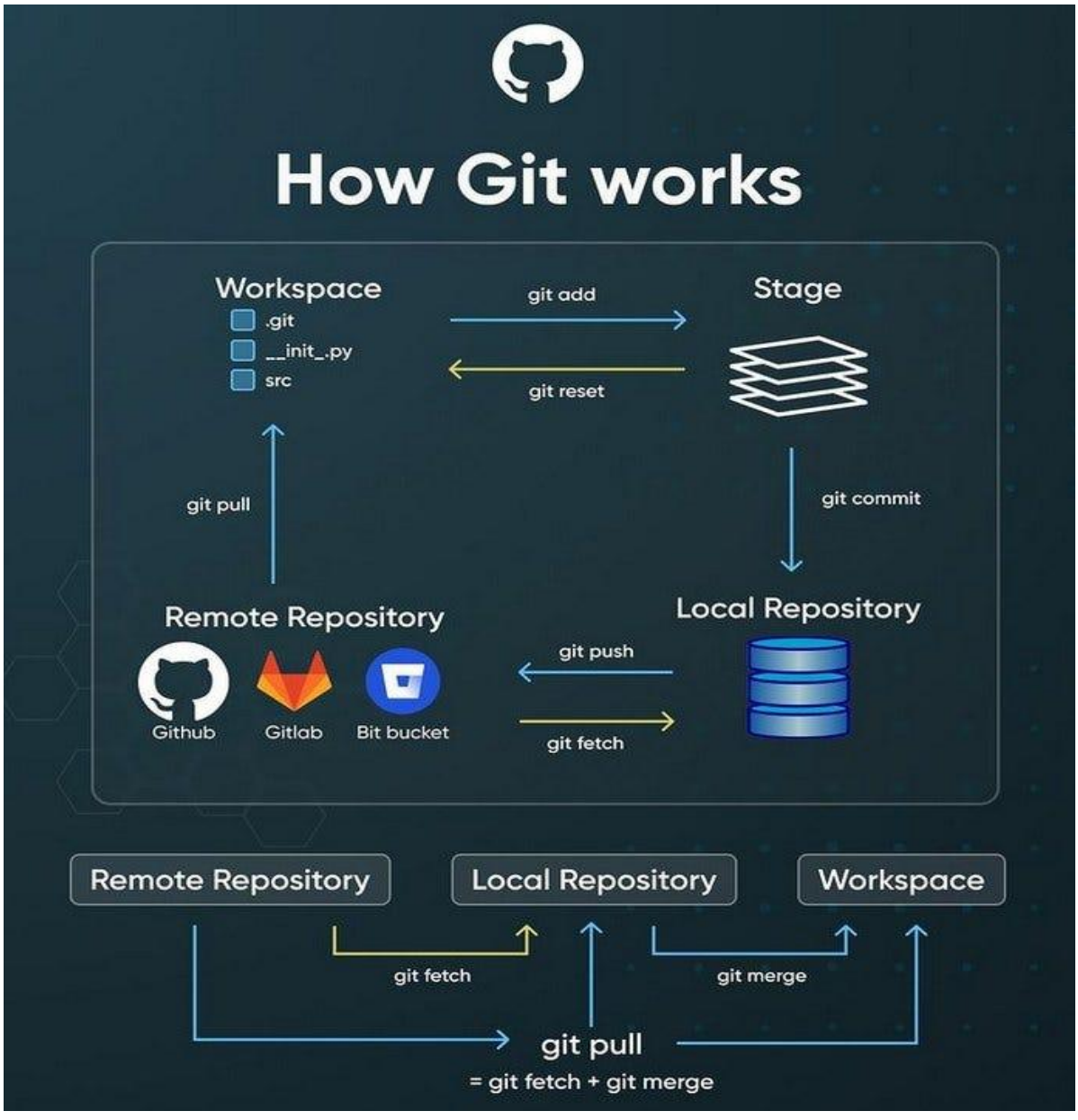
INTRODUCTION TO

GIT


Working Directory
Staging Area
Git Directory (Repository)
Checkout the project
Stage File
Commit


How Git works
Workspace
.git
__init_.py
src
git add
git reset
Stage
git pull
git commit
Remote Repository
Github
Gitlab
Bit bucket
git push
git fetch
Local Repository
Remote Repository
Local Repository
Workspace
git fetch
git merge
git pull
= git fetch + git merge

1. What is Git, and what is its purpose ?

Git is a distributed version control system that helps track changes in files and collaborate on projects. Its purpose is to manage and maintain different versions of code or files, enabling multiple developers to work on the same project simultaneously.

2. What is Repository ?

- A Git repository is a virtual storage of your project.
- It allows you to save versions of your code, which you can access when needed.
- It tracks and saves the history of all changes made to the files
Two types of Repository:

1. **Local Repository**
2. **Remote Repository**

3. How is Git different from other version control systems?

Git is different from other version control systems because it is distributed, meaning each developer has a complete copy of the entire project history. This allows for offline work and faster operations compared to centralized systems. Git also has powerful branching and merging capabilities.

What are the basic Git commands you use frequently?

git init

This command let us create a new reposiitory. A hidden .git directory is added to the folder. Most of the git command do not work outside initialized project , so this is the first command you will run in a project Go to project folder > run git init

```
$ git init <repository name>
```

git commit

This command saves your changes to your local repository. Everytime you commit you have to add a small message about the changes you made. This will help to keep track of the changes later.

```
$ git commit -m <commit message>
```

git push

This command push your changes from local repository to your remote repository. One can only push the committed changes. It also creates the repository with the branch name you enter if repository does not exist on remote location. If branch is already connected to remote then run - git push

```
$ git push <variable name> <branch>
```

git pull

This command fetches latest changes from remote repository to your local. This is helpful when multiple people are working on same repository. It will help to keep your local repo updated with latest code. If branch is already connected to remote then run - git pul

```
$ git pull <Repository Link>
```

git clone

This command creates a local copy of a remote repository . When you clone a repo the source code gets automatically downloaded to local machine. This local repo will point to remote repo and can PUSH and PULL changes to it

```
$ git clone <url>
```

git add

This command add your changes to staging area where you can compare you local version with remote repo code. It is mandatory to stage the code before commit(push to remote) using git add command. To stage all files use (.) - git add . in the same repo

```
$ git add <file>
```

git staus

The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.

Mostly, it is used to display the state between Git Add and Git commit command. We can check whether the changes and files are tracked or not.

```
$ git status
```

git diff

Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.

```
$ git diff
```

git branch

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch

```
$ git branch <branch name>
```

git checkout

In Git, the term checkout is used for the act of switching between different versions of a target entity. The git checkout command is used to switch between branches in a repository. Be careful with your staged files and commits when switching between branches.

```
$ git checkout <branchname>
```

git merge

This command merges the specified branch's history into the current branch.

In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches.

```
$ git merge <branch name>
```

git remote add <name> <url>

Adds a remote repository with a specified name and URL.

```
$ git checkout <branchname>
```

git remote -v

List the remote connections you have to other repositories.

```
$ git remote -v
```

git log

Displays a history of commits.
Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.

```
$ git log
```

git stash

Temporarily saves changes that are not ready to be committed.

```
$ git status
```

git fetch

Downloads changes from a remote repository without merging them.

Git fetch Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.

```
$ git fetch< repository Url>
```

git revert

Creates a new commit that undoes a previous commit.

In Git, the term revert is used to revert some changes. The git revert command is used to apply revert operation. It is an undo type command. However, it is not a traditional undo alternative. It does not delete any data in this process; instead, it will create a new change with the opposite effect and thereby undo the specified commit. Generally, git revert is a commit.

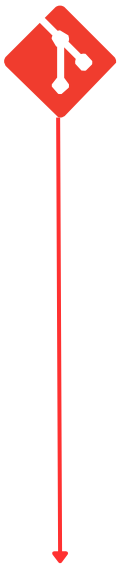
```
$ git revert
```

git rm

Removes a file from the repository.

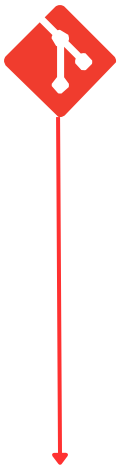
In Git, the term rm stands for remove. It is used to remove individual files or a collection of files. The key function of git rm is to remove tracked files from the Git index. Additionally, it can be used to remove files from both the working directory and staging index.

```
$ git rm <file Name>
```



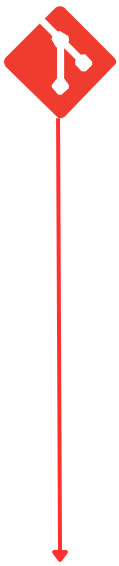
4. What is a merge conflict, and how would you resolve it?

A merge conflict occurs when Git encounters conflicting changes in different branches during a merge or a pull operation. To resolve a merge conflict, you need to manually edit the conflicting file(s) to reconcile the differences. After resolving the conflicts, you can stage the changes and complete the merge using "git add" and "git commit" commands.



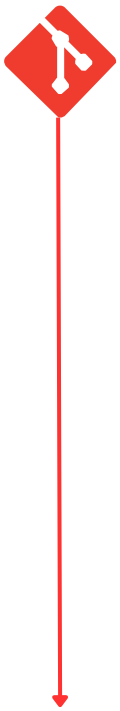
5. How do you create a new branch in Git?

To create a new branch in Git, you can use the command "git branch <branch-name>". For example, to create a branch named "feature-branch", you would run "git branch feature-branch". To switch to the newly created branch, you can use "git checkout <branch-name>" or "git switch <branch-name>".



6. What is the difference between Git and GitHub?

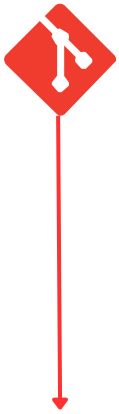
Git is the version control system itself, while GitHub is a web-based hosting service for Git repositories. Git is responsible for tracking changes, managing branches, and performing version control tasks locally. GitHub provides a platform for hosting and collaborating on Git repositories with additional features like issue tracking, pull requests, and project management tools.



7. Explain the concept of git clone and git fork.

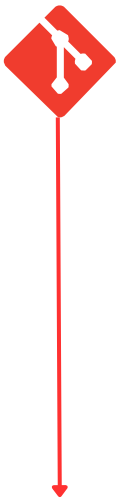
"git clone" is a command used to create a local copy of a remote Git repository. It downloads the entire repository, including its history, branches, and files. On the other hand.

"git fork" is a feature provided by hosting platforms like GitHub. It allows you to create a copy of a repository under your account, enabling you to make changes and contribute to the project without modifying the original repository.



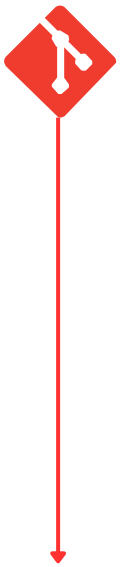
8. How do you revert a commit in Git?

To revert a commit in Git, you can use the command "git revert <commit-hash>". This command creates a new commit that undoes the changes made in the specified commit. The commit hash can be obtained from the commit history using "git log".



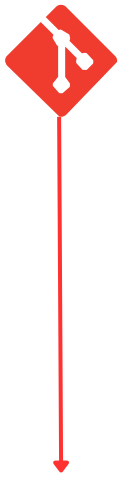
9. What is the purpose of Git rebase?

Git rebase allows you to integrate changes from one branch onto another by moving or combining commits. It helps create a linear commit history, keeping the project timeline clean and easier to follow. Rebasing is commonly used to incorporate changes from a feature branch into the main branch before merging.



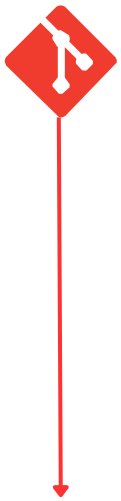
10. How do you remove a file from a Git repository?

To remove a file from a Git repository and stage the deletion, you can use the command "git rm <file>". This command removes the file from both your working directory and the Git repository. After executing the command, you need to commit the change using "git commit" to permanently remove the file from the repository.



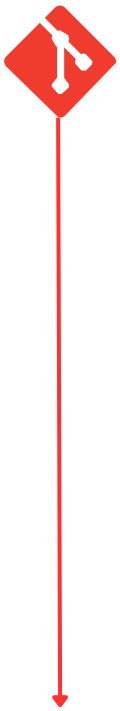
11. What is the purpose of Git stash?

Git stash is used to temporarily save and hide changes in your working directory that are not ready to be committed. It allows you to switch branches or work on a different task without committing incomplete changes. Later, you can apply the stashed changes to your working directory using "git stash apply" or "git stash pop".



12. How can you view the commit history in Git?

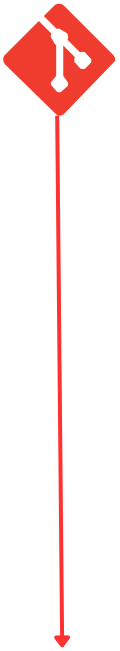
You can view the commit history in Git using the command "git log". This command displays a detailed list of commits in reverse chronological order, showing the commit hash, author, date, and commit message. It provides a comprehensive overview of the project's commit history.



12. Explain the concept of Git pull and Git fetch.

"git pull" is a command that fetches changes from a remote repository and automatically merges them into the current branch. It combines the "git fetch" and "git merge" commands. On the other hand.

"git fetch" only downloads the latest changes from a remote repository, updating your local copy, but it doesn't automatically merge the changes. It allows you to review the changes before merging them using "git merge" or other Git commands.



13. What are Git hooks, and how can they be used?

Git hooks are scripts that are executed before or after specific Git events, such as committing, merging, or pushing. They allow you to automate certain actions or enforce custom workflows in a Git repository. Git hooks can be used to enforce code quality checks, run tests, trigger deployments, or perform any custom actions based on your project's needs. Git provides both client-side and server-side hooks that can be customized and configured.

14. How do you discard local changes in Git?

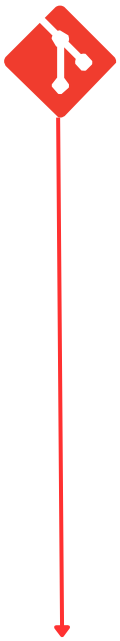
To discard local changes in Git and revert a file to its last committed state, you can use the command "git checkout -- <file>". This command replaces the file in your working directory with the version from the last commit. If you want to discard all local changes and revert to the last commit for the entire repository, you can use "git reset --hard HEAD" command. This will reset your working directory and staging area to the last commit.

15. How can you view the commit history in Git?

You can view the commit history in Git using the command "git log". This command displays a detailed list of commits in reverse chronological order, showing the commit hash, author, date, and commit message. It provides a comprehensive overview of the project's commit history.

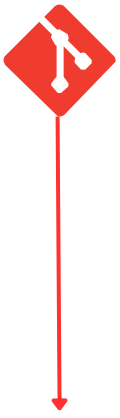
16. Why git is used?

Git is used for version control, enabling tracking of changes in files, facilitating collaboration among developers, managing branches and merges, supporting distributed development, offering flexibility for different workflows, and integrating with various tools and services.



17. What is a version control system?

A version control system (VCS) is a software tool that helps track and manage changes to files or a set of files over time. It provides a systematic way to keep track of revisions, modifications, and additions made to files, allowing multiple people to work on the same project simultaneously. A VCS maintains a history of changes, enabling users to view, compare, and revert to previous versions of files. It also facilitates collaboration, branch management, and merging of changes, ensuring a smooth and organized development process.



18. How can you view the commit history in Git?

You can view the commit history in Git using the command "git log". This command displays a detailed list of commits in reverse chronological order, showing the commit hash, author, date, and commit message. It provides a comprehensive overview of the project's commit history.

19. 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.

20. 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.

21. What does the command git config do?

The git config command is a convinient way to set Configuration option for defining the behavior of the Mepository, usen information and preferences, git installaria
based configuration, and many such things.

22. What is the functionality of git-ls-tree?

The command returns a tree object representation

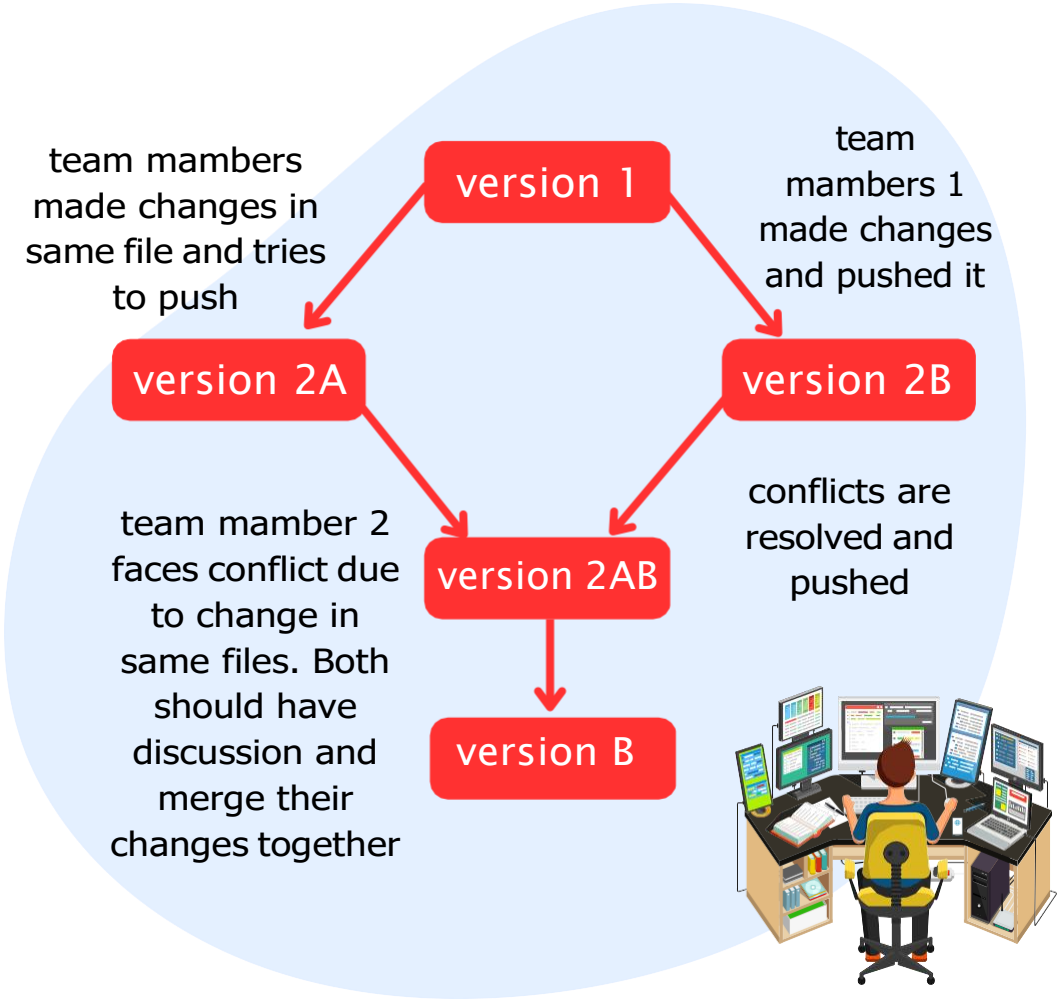
of current repository along with the mode and the name of each item and SHA-1 value of the blob.

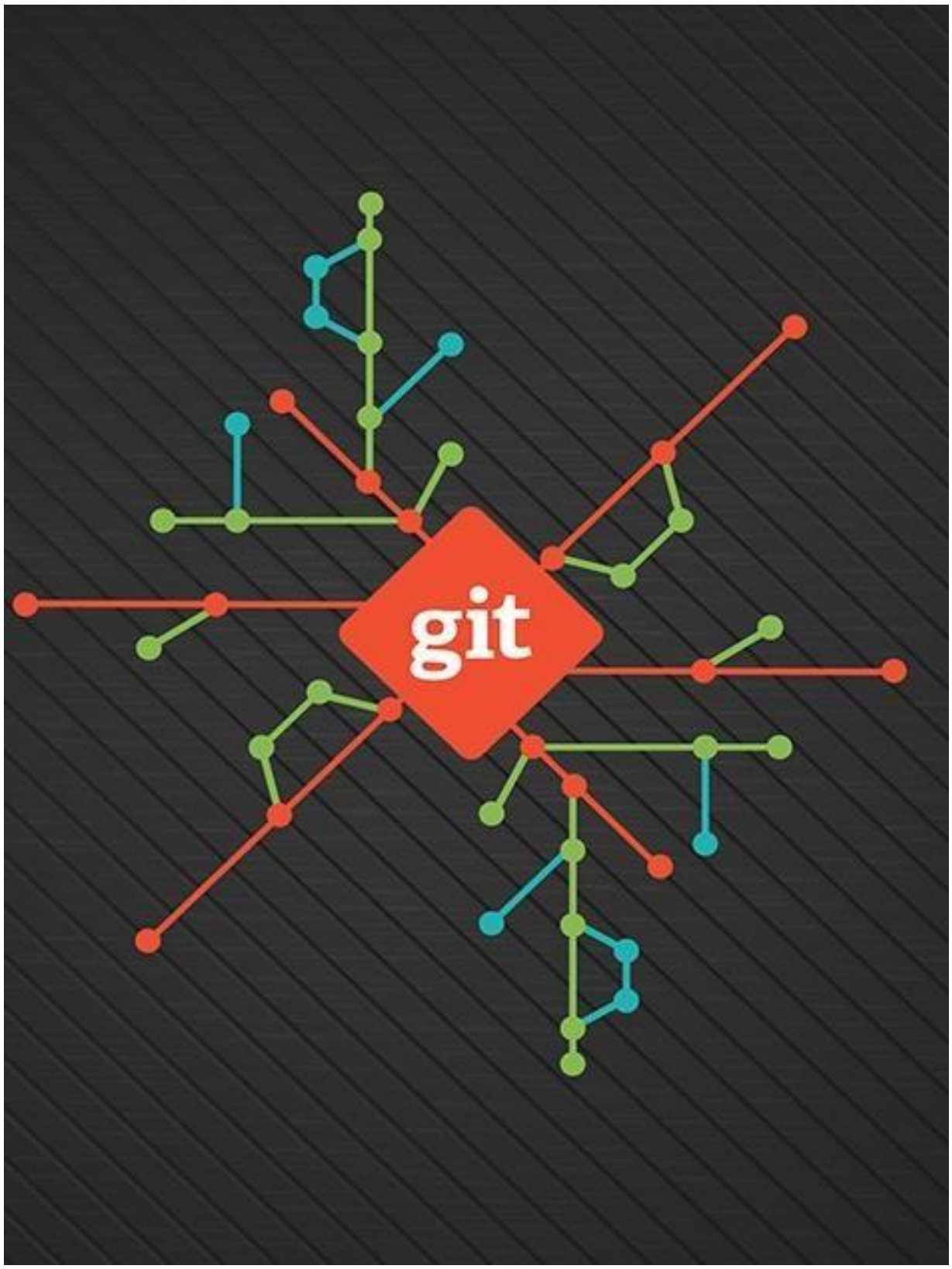
23. What is a git conflict ?

Git usually handles feature menger automatically but Sometimes while working in a team environment, thou might be causes of conflicts such as:

1. **When two separate branches have changes to the same line in a file.**
2. **file is deleted is one branch but has been modified in the other.**

Those 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.





INTRODUCTION TO

GIT

Creating

Initializing a repository

```
git init
```

Staging files

```
git add file1.js          # Stages a single file
git add file1.js file2.js # Stages multiple files
git add *.js              # Stages with a pattern
git add .                 # Stages the current directory and all its content
```

Viewing the status

```
git status                # Full status
git status -s             # Short status
```

Committing the staged files

```
git commit -m "Message"   # Commits with a one-line message
git commit                # Opens the default editor to type a long message
```

Skipping the staging area

```
git commit -am "Message"
```

Removing files

```
git rm file1.js           # Removes from working directory and staging area
git rm --cached file1.js  # Removes from staging area only
```

Renaming or moving files

```
git mv file1.js file1.txt
```

Viewing the staged/unstaged changes

```
git diff                # Shows unstaged changes
git diff --staged       # Shows staged changes
git diff --cached       # Same as the above
```

Viewing the history

```
git log                 # Full history
git log --oneline       # Summary
git log --reverse       # Lists the commits from the oldest to the newest
```

Viewing a commit

```
git show 921a2ff        # Shows the given commit
git show HEAD           # Shows the last commit
git show HEAD~2         # Two steps before the last commit
git show HEAD:file.js   # Shows the version of file.js stored in the last commit
```

Unstaging files (undoing git add)

```
git restore --staged file.js  # Copies the last version of file.js from repo to index
```

Discarding local changes

```
git restore file.js           # Copies file.js from index to working directory
git restore file1.js file2.js # Restores multiple files in working directory
git restore .                 # Discards all local changes (except untracked files)
git clean -fd                 # Removes all untracked files
```

Restoring an earlier version of a file

```
git restore --source=HEAD~2 file.js
```

Browsing History

Viewing the history

```
git log --stat                  # Shows the list of modified files
git log --patch                 # Shows the actual changes (patches)
```

Filtering the history

```
git log -3                      # Shows the last 3 entries
git log --author="Mosh"
git log --before="2020-08-17"
git log --after="one week ago"
git log --grep="GUI"            # Commits with "GUI" in their message
git log -S"GUI"                 # Commits with "GUI" in their patches
git log hash1..hash2            # Range of commits
git log file.txt                # Commits that touched file.txt
```

Formatting the log output

```
git log --pretty=format:"%an committed %H"
```

Creating an alias

```
git config --global alias.lg "log --oneline"
```

Viewing a commit

```
git show HEAD~2
git show HEAD~2:file1.txt    # Shows the version of file stored in this commit
```

Comparing commits

```
git diff HEAD~2 HEAD          # Shows the changes between two commits
git diff HEAD~2 HEAD file.txt # Changes to file.txt only
```

Checking out a commit

```
git checkout dad47ed      # Checks out the given commit
git checkout master       # Checks out the master branch
```

Finding a bad commit

```
git bisect start
git bisect bad            # Marks the current commit as a bad commit
git bisect good ca49180   # Marks the given commit as a good commit
git bisect reset          # Terminates the bisect session
```

Finding contributors

```
git shortlog
```

Viewing the history of a file

```
git log file.txt          # Shows the commits that touched file.txt
git log --stat file.txt   # Shows statistics (the number of changes) for file.txt
git log --patch file.txt  # Shows the patches (changes) applied to file.txt
```

Finding the author of lines

```
git blame file.txt        # Shows the author of each line in file.txt
```

Tagging

```
git tag v1.0              # Tags the last commit as v1.0
git tag v1.0 5e7a828      # Tags an earlier commit
git tag                   # Lists all the tags
git tag -d v1.0           # Deletes the given tag
```

Branching & Merging

Managing branches

```
git branch bugfix          # Creates a new branch called bugfix
git checkout bugfix        # Switches to the bugfix branch
git switch bugfix          # Same as the above
git switch -C bugfix       # Creates and switches
git branch -d bugfix       # Deletes the bugfix branch
```

Comparing branches

```
git log master..bugfix     # Lists the commits in the bugfix branch not in master
git diff master..bugfix    # Shows the summary of changes
```

Stashing

```
git stash push -m "New tax rules"   # Creates a new stash
git stash list                      # Lists all the stashes
git stash show stash@{1}            # Shows the given stash
git stash show 1                    # shortcut for stash@{1}
git stash apply 1                   # Applies the given stash to the working dir
git stash drop 1                    # Deletes the given stash
git stash clear                     # Deletes all the stashes
```

Merging

```
git merge bugfix           # Merges the bugfix branch into the current branch
git merge --no-ff bugfix   # Creates a merge commit even if FF is possible
git merge --squash bugfix  # Performs a squash merge
git merge --abort          # Aborts the merge
```

Viewing the merged branches

```
git branch --merged        # Shows the merged branches
git branch --no-merged     # Shows the unmerged branches
```

Rebasing

```
git rebase master          # Changes the base of the current branch
```

Cherry picking

```
git cherry-pick dad47ed    # Applies the given commit on the current branch
```

Collaboration

Cloning a repository

```
git clone url
```

Syncing with remotes

```
git fetch origin master     # Fetches master from origin
git fetch origin            # Fetches all objects from origin
git fetch                   # Shortcut for "git fetch origin"
git pull                    # Fetch + merge
git push origin master      # Pushes master to origin
git push                    # Shortcut for "git push origin master"
```

Sharing tags

```
git push origin v1.0        # Pushes tag v1.0 to origin
git push origin —delete v1.0
```

Sharing branches

```
git branch -r               # Shows remote tracking branches
git branch -vv              # Shows local & remote tracking branches
git push -u origin bugfix   # Pushes bugfix to origin
git push -d origin bugfix   # Removes bugfix from origin
```

Managing remotes

```
git remote                  # Shows remote repos
git remote add upstream url # Adds a new remote called upstream
git remote rm upstream      # Remotes upstream
```

Rewriting History

Undoing commits

```
git reset --soft HEAD^      # Removes the last commit, keeps changed staged
git reset --mixed HEAD^     # Unstages the changes as well
git reset --hard HEAD^      # Discards local changes
```

Reverting commits

```
git revert 72856ea          # Reverts the given commit
git revert HEAD~3..         # Reverts the last three commits
git revert --no-commit HEAD~3..
```

Recovering lost commits

```
git reflog                  # Shows the history of HEAD
git reflog show bugfix      # Shows the history of bugfix pointer
```

Amending the last commit

```
git commit --amend
```

Interactive rebasing

```
git rebase -i HEAD~5
```