# 04/07/2024 - GIT command and purpose

1. **git init:** Initializes a new Git repository in the current directory. This command sets up all the necessary files and directories for version control.

2. **git clone [url]:** Creates a copy of an existing Git repository from a remote source (URL) to your local machine, including all of its history and branches.

3. **git add [file]:** Adds changes in the specified file(s) to the staging area, preparing them to be included in the next commit.

4. **git commit -m "[message]":** Records the changes in the staging area to the repository with a descriptive message explaining the commit.

5. **git status**: Displays the current state of the working directory and the staging area, showing which files are untracked, modified, or staged for commit.

6. **git push [remote] [branch]:** Uploads local commits to a remote repository, allowing you to share your work with others or backup your changes.

7. **git pull [remote] [branch]:** Fetches and integrates changes from a remote repository into the current branch, keeping your local branch up-to-date.

8. **git fetch [remote]:** Downloads objects and references from a remote repository without merging them into the current branch, allowing you to review changes before integrating.

9. **git merge [branch]:** Combines changes from the specified branch into the current branch, creating a new commit that records the merge.

10. **git branch:** Lists all branches in the repository or creates a new branch when provided with a branch name.

11. **git checkout [branch/file]:** Switches to a specified branch or restores a specific file in the working directory from the repository.

12. **git log:** Displays a log of commits made in the repository, showing commit hashes, authors, dates, and messages.

13. **git diff [file]:** Shows changes between the working directory and the staging area, or between two commits, highlighting what has been added, modified, or deleted.

14. **git reset [file]:** Removes changes in the specified file(s) from the staging area, keeping them in the working directory for further editing.

15. **git rm [file]:** Deletes the specified file from the working directory and stages the deletion for the next commit.

16. **git stash:** Temporarily saves changes in the working directory that are not ready to be committed, allowing you to work on a clean slate without losing your work.

17. **git stash pop:** Restores the most recently stashed changes to the working directory and removes them from the stash list.

18. **git tag [tagname] [commit]:** Creates a tag pointing to a specific commit, often used to mark release points like versions.

19. **git remote add [name] [url]:** Adds a new remote repository with a specified name and URL, enabling you to interact with the remote.

20. **git remote -v:** Displays the URLs of the remote repositories associated with the local repository.

21. **git rebase [branch]:** Applies commits from one branch onto another, rewriting commit history for a linear project history.

22. **git cherry-pick [commit]: Applies** changes from a specific commit to the current branch, useful for applying selected changes without merging entire branches.

23. **git revert [commit]:** Creates a new commit that undoes the changes made by a specified commit, preserving the project's history.

24. **git show [commit]:** Displays detailed information about a specified commit, including the changes made, author, and message.

25. **git blame [file]:** Shows line-by-line information of a file, including the author and commit that last modified each line.

26. **git archive [branch**]: Creates an archive file (e.g., tar or zip) of the files in the specified branch, useful for distribution.

27. **git bisect:** Uses binary search to find the specific commit that introduced a bug by automatically checking out and testing commits.

28. **git submodule:** Manages submodules, which are repositories nested inside another repository, allowing you to include and track external dependencies.

29. **git clean -f:** Removes untracked files from the working directory, helping to keep your project clean and organized.

30. **git config**: Sets configuration options for Git, such as user information, aliases, and settings, either globally or per-repository.

31. **git ls-tree [branch]:** Displays the contents of a tree object, such as a specific branch, showing the files and directories it contains.

32. **git grep [pattern]:** Searches for a specified pattern in the repository's files, useful for finding code snippets or text within the project.

33. **git describe:** Gives an approximate human-readable name to a commit, based on the most recent tag reachable from it.

34. **git shortlog:** Summarizes git log output in a format suitable for release notes, grouping commits by author and showing commit messages.

35. **git reflog:** Records updates to the tip of branches and other references, allowing you to recover lost commits and see branch history.