GIT COMMANDS

1. **git** init : Create an empty **git** repository or reinitialize an existing one.
2. **git** clone [url] : Clone a repository into a new directory.
3. **git** add [file] : Add file(s) to the staging area.
4. **git** add . : Add all files in the current directory to the staging area.
5. **git** add -p : Interactively choose hunks of patch between the index and the work tree.
6. **git** add -u : Add all changes to tracked files in the working directory.
7. **git** add -A : Add all changes to tracked and untracked files in the working directory.
8. **git** commit -m "[message]" : Record changes to the repository with a message.
9. **git** commit -a : Stage all modified files and commit them (skipping the ` **git** add` step).
10. **git** status : Show the status of the working directory and staging area.
11. **git** diff : Show changes between commits, commit and working directory, etc.
12. **git** diff [commit1] [commit2] : Show changes between two commits.
13. **git** diff --staged : Show changes between the staging area and the last commit.
14. **git** log : Display commit history.
15. **git** log --graph : Show commits as a graph.
16. **git** log --author="[name]" : Show commits by a specific author.
17. **git** log --oneline : Condense each commit to a single line.
18. **git** branch : List, create, or delete branches.
19. **git** branch -a : List all branches (local and remote).
20. **git** branch -d [branch] : Delete a specified branch.
21. **git** branch -m [oldbranch] [newbranch] : Rename a branch.
22. **git** checkout [branch] : Switch branches or restore working tree files.
23. **git** checkout -b [branch] : Create a new branch and switch to it.
24. **git** checkout -b [branch] [commit] : Create a new branch based on a specific commit.
25. **git** checkout -- [file] : Restore a file in the working directory to its state at the last commit.
26. **git** merge [branch] : Merge changes from another branch into the current branch.
27. **git** merge --abort : Abort the current conflict resolution process, and try to reconstruct the pre-merge state.
28. **git** merge --no-ff [branch] : Create a merge commit even if the merge resolves as a fast-forward.

29. **git** pull : Fetch from and integrate with another repository or a local branch.

30. **git** pull --rebase : Fetch from the upstream repository and rebase the current branch on top of it.

31. **git** push : Update remote refs along with associated objects.

32. **git** push -u [remote] [branch] : Push the current branch to the remote repository and set the upstream branch.

33. **git** push [remote] --delete [branch] : Delete a remote branch.

34. **git** remote : Manage set of tracked repositories.

35. **git** remote add [name] [url] : Add a remote repository to be tracked.

36. **git** remote rm [name] : Remove a remote repository from being tracked.

37. **git** remote show [remote] : Show information about a remote.

38. **git** remote prune [remote] : Delete remote-tracking branches that no longer exist on the remote.

39. **git** fetch : Download objects and refs from another repository.

40. **git** fetch --prune : After fetching, remove any remote-tracking branches which no longer exist on the remote.

41. **git** reset [file] : Reset the staging area for file(s) to the last commit.

42. **git** reset --hard [commit] : Reset the working directory and staging area to a specific commit.

43. **git** reset --soft [commit] : Reset the current branch's HEAD to a specific commit, keeping all changes staged.

44. **git** reset --mixed [commit] : Reset the current branch's HEAD to a specific commit, unstaging all changes.

45. **git** revert [commit] : Revert one or more commits.

46. **git** rm [file] : Remove files from the working tree and from the index.

47. **git** mv [file-original] [file-renamed] : Rename a file.

48. **git** tag [name] : Create a tag for the given commit.

49. **git** tag -a [name] -m "[message]" : Create an annotated tag.

50. **git** tag -d [name] : Delete a tag.

51. **git** tag : List tags.

52. **git** show [object] : Show various types of objects.

53. **git** show-ref : List references in a local repository.

54. **git** bisect : Use binary search to find the commit that introduced a bug.

55. **git** cherry-pick [commit] : Apply the changes introduced by some existing commits.

56. **git** cherry : Find commits that are on one branch but not another.

57. **git** blame [file] : Show what revision and author last modified each line of a file.

58. **git** clean : Remove untracked files from the working directory.

59. **git** grep [pattern] : Print lines matching a pattern.

60. **git** rebase : Reapply commits on top of another base tip.

61. **git** rebase -i [commit] : Interactively rebase commits.

62. **git** reflog : Reference logs, or "history of what you've done".

63. **git** bisect : Use binary search to find the commit that introduced a bug.

64. **git** archive : Create a tar or zip archive of a **git** repository.

65. **git** stash : Stash changes in a dirty working directory away.

66. **git** stash pop : Apply the most recently stashed changes.

67. **git** stash apply : Apply the most recently stashed changes without removing them from the stash list.

68. **git** stash list : List all stashed changesets.

69. **git** submodule : Initialize, update or inspect submodules.

70. **git** worktree : Manage multiple working trees attached to the same repository.