Basic Git Configuration

1. **`git config --version`**
   - Displays the version of Git installed on your system.
2. **`git config --global user.name " "`**
   - Sets the global username for commits. Replace `" "` with your username.
3. **`git config --global user.email " "`**
   - Sets the global email address for commits. Replace `" "` with your email.
4. **`git config --list`**
   - Lists all the current Git configuration settings (user details, aliases, etc.).

Directory Operations

5. **`mkdir f1 f2 f3`**
   - Creates multiple directories (`f1`, `f2`, `f3`) in the current path.
6. **`mkdir -p f2/f5`**
   - Creates a nested directory (`f5`) inside `f2`. The `-p` ensures the parent directories are created if they don't already exist.
7. **`rmdir f1`**
   - Removes an empty directory (`f1`).
8. **`rmdir -R f1`**
   - Recursively removes the directory `f1` and its contents.

File Operations

9. **`touch script.js`**
   - Creates an empty file named `script.js`.
10. **`touch f4/index.html`**
    - Creates a new empty file `index.html` inside a folder `f4`. If `f4` does not exist, the command will fail.
11. **`rm script.js`**
    - Deletes the file `script.js`.
12. **`cp script.js file1/new_script.js`**
    - Copies `script.js` into the directory `file1` with a new name `new_script.js`.
13. **`mv script.js f1/new_script.js`**
    - Moves `script.js` into `f1` with a new name `new_script.js`.
14. **`mv script.js hello.js`**
    - Renames `script.js` to `hello.js`.

Git Initialization and File Tracking

15. **`git init`**
    - Initializes a new Git repository in the current directory.
16. **`git add index.html`**
    - Stages the file `index.html` for the next commit.
17. **`git status`**
    - Shows the current state of the working directory and staging area (e.g., untracked, staged, or modified files).
18. **`git commit -m" "`**
    - Commits the staged changes with a message. Replace `" "` with your commit message.
19. **`git log`**
    - Displays a list of all commits in the current branch, including details like author, date, and commit message.
20. **`rm -rf .git`**
    - Deletes the Git repository (`.git` folder) and removes version control from the directory.

Differences and Ignored Files

21. **`git diff`**
    - Shows unstaged changes between the working directory and the index (staged area).
22. **`git diff --staged`**
    - Shows differences between the staged changes and the last commit.
23. **`touch .gitignore`**
    - Creates a `.gitignore` file to specify patterns of files to ignore.
24. **In `.gitignore`, write `*.log`:**
    - Ignores all files with the `.log` extension.
25. **Write the filename in `.gitignore`:**
    - Ignores a specific file by name.
26. **`git rm -f abc/index.log`**
    - Forcefully removes the file `abc/index.log` and stages the removal.
27. **`git mv index.html home.html`**
    - Renames `index.html` to `home.html`, and the change is automatically staged.
28. **`git rm --cached .DS_Store`**
    - Removes `.DS_Store` from the repository while keeping the file locally.

Restore and Checkout

29. **`git restore --staged filename`**
    - Unstages a file that was previously staged.
30. **`git checkout -- filename`**
    - Reverts a file in the working directory to the last committed version.
31. **`git checkout -f`**
    - Forcibly reverts all files to the last committed version.

Aliases

32. **`git config --global alias.shortname realname`**
    - Creates a shorthand alias for a Git command.

Branches

33. **`git branch -v`**
    - Lists all branches with the latest commit on each branch.
34. **`git branch nameofnewbranch`**
    - Creates a new branch named `nameofnewbranch`.
35. **`git checkout branchname`**
    - Switches to the branch `branchname`.
36. **`git checkout -b branchname`**
    - Creates and switches to a new branch `branchname`.

Viewing Logs

37. **`git log --oneline`**
    - Displays commits in a compact, one-line format.
38. **`git log --oneline --graph`**
    - Displays commits in a graph structure.
39. **`git log --oneline --graph --all`**
    - Displays commits for all branches in a graph structure.

Branch Deletion

40. **`git branch -d branchname`**
    - Deletes the branch `branchname` if it is already merged.
41. **`git branch -D branchname`**

- Forcefully deletes the branch `branchname`.

Merging and Rebasing

42. **`git merge impfix`**
    - Merges the branch `impfix` into the current branch.
43. **`git rebase master`**
    - Replays changes from the current branch onto `master`.

Remote Repositories

44. **`git push -u origin main`**
    - Pushes the branch `main` to the remote repository and sets the upstream branch.
45. **`git push origin main`**
    - Pushes the `main` branch to the remote repository.
46. **`git fetch`**
    - Downloads changes from the remote repository.
47. **`git merge origin/master`**
    - Merges the changes fetched from the remote branch `origin/master` into the current branch.
48. **`git remote my-remote`**
    - Adds a new remote named `my-remote`.
49. **`git remote remove my-remote`**
    - Removes the remote `my-remote`.
50. **`git pull`**
    - Fetches changes from the remote repository and merges them into the current branch.

Stashing

51. **`git stash`**
    - Temporarily saves changes not yet committed.
52. **`git stash list`**
    - Lists all stashes.
53. **`git stash apply stash@{0}`**
    - Applies the changes from stash `stash@{0}`.
54. **`git stash drop stash@{0}`**
    - Deletes the stash `stash@{0}`.

Cleaning and Amending Commits

55. **`git clean -f -d`**
    - Forcefully removes untracked files and directories.
56. **`git commit --amend`**
    - Amends the last commit (e.g., to change the message).

Working with Commits

57. **`git checkout commitid`**
    - Switches to a specific commit.
58. **`git switch -`**
    - Switches back to the previously checked-out branch.
59. **`git revert commitid`**
    - Reverts the changes made in a specific commit.
60. **`git revert -n commitid`**
    - Reverts changes and stages them for the next commit.
61. **`git revert --abort`**
    - Cancels the revert operation.
62. **`git reset --soft commitid`**
    - Resets to a specific commit, keeping changes staged.
63. **`git reset --hard commitid`**
    - Resets to a specific commit, discarding all changes after it.
64. **`git cherry-pick commitid`**
    - Applies a specific commit onto the current branch.