1. To initialize the new git repository >> **git init**
2. To clone a repository from remote source >> **git clone <repository url>**
3. To add changed files to staging area >> **git add <file name>**
4. To commit changes to the staging area >> **git commit -m <message>**
5. To check the current status of git >> **git status**
6. To view commit history in git repository >> **git log**
7. To move from one branch to another existing branch >> **git checkout <branch name>**
8. To create a new branch with current branch >> **git checkout -b <branch name>**
9. To pull the changes to remote branch >> **git pull**
10. To push the changes to the remote branch >> **git push**
11. To check the difference with previous commit with new changes >> **git diff <file name>** here file name not mandatory.
12. To merge the changes of another branch changes to current branch >> **git merge <branch name>**
13. Git fetch is a **primary command used to download contents from a remote repository** and update the local git repository >> **git fetch**
14. To manage remote repositories >> **git remote**
15. The git config function is a convenience function that is used to **set git configuration values** on a global or local project level >> **git config**
16. `git reset` is a command in Git that is used to reset the state of the current HEAD to a specified state. It allows you to undo changes made to your working

directory and staging area in different ways depending on the options you provide.

Here are some common uses of `git reset`:

a. **Resetting the Staging Area (Index)**:
   If you have staged changes using `git add` but you want to unstage them without losing the changes in your working directory, you can use `git reset` with the `--mixed` or default option:

   ```bash
   git reset HEAD <file>
   ```

   This command unstages the changes in `<file>` and moves them back to the working directory.

b. **Resetting to a Specific Commit**:
   You can reset the current branch to a specific commit, either preserving or discarding changes made after that commit:

   - To preserve changes made after the commit:
     ```bash
     ```

```bash
    git reset --soft <commit-hash>
```

   - To discard changes made after the commit:

```bash
    git reset --hard <commit-hash>
```

   Replace `<commit-hash>` with the hash of the commit you want to reset to. This is useful if you want to undo all changes made after a certain point in history.

c. **Undoing the Last Commit**:

   If you want to undo the last commit but keep the changes in your working directory, you can use `git reset` with the `--soft` option followed by `HEAD~1`:

```bash
   git reset --soft HEAD~1
```

   This command resets the current branch to the commit before the last one, keeping the changes from the last commit in your working directory and staging area.

d. **Moving the Branch Pointer**:

By default, `git reset` moves the branch pointer to the specified commit. This can be useful if you want to "rewind" the history of your branch to a previous state.

e. **Uncommitting a Merge**:

If you've merged changes into your branch and want to undo the merge while keeping the changes in your working directory, you can use `git reset` with the `--merge` option followed by `HEAD~1`:

```bash
git reset --merge HEAD~1
```

This command will undo the merge commit, but keep the changes from the merge in your working directory, allowing you to resolve any conflicts before committing again.

Remember to use `git reset` with caution, especially when using the `--hard` option, as it can result in the loss of changes that have not been committed. Always make sure you understand the consequences of the reset operation before executing it.

17. If we want to remove a file from our working directory we use >> **git rm <file name>**
18. **git grep** is a command in Git that allows you to search through the content of your Git repository for lines that match a specific pattern. It's a powerful tool for quickly finding occurrences of text within your project's files >> **git grep <message word>**

    You can also provide additional options to customize the behavior of git grep, such as limiting the search to specific files or directories, ignoring case sensitivity, or displaying the line numbers where matches occur. Here are some common options:

- **-i or --ignore-case**: Ignore case differences when searching.
- **-n or --line-number**: Prefix each line of output with the 1-based line number within its file.
- **-w or --word-regexp**: Use this option to match the pattern as a whole word.
- **-- <pathspec>:** Limits the search to the specified files or directories.

19. To stash the current changes in the working directory >> **git stash**

    Suppose if we want to revert back changes which we stashed before we use **git stash pop** for previous stash if we want to get specific stash we use **git stash <stash id>**

    To get stash list we use >> **git stash list**
20. To review the changes in a specific file line by line in terminal we use >> **git blame <file name>**
21. git rebase: Reapply commits on top of another base tip >> **git rebase**

22. To view different type of objects and commits we use >> **git show**
23. To create and delete branches using Git, you can use the following commands

    1. **Create a branch**: Use the `git branch <branch-name>` command followed by the name of the new branch you want to create.

       ```bash

      git branch new-feature

       ```

       This command creates a new branch named "new-feature" but does not switch to it. You'll remain on the current branch.

    2. **Switch to a branch**: To start working on the newly created branch, you can use the `git checkout` command.

       ```bash

    git checkout new-feature

       ```

      Alternatively, you can create a new branch and switch to it in one step using the `-b` option:

       ```bash
       ```

```
git checkout -b new-feature
```

This command creates a new branch named "new-feature" and switches to it.

3. **Delete a branch**: Use the `git branch -d <branch-name>` command followed by the name of the branch you want to delete. Be cautious when deleting branches, especially if they contain important work.

```bash
git branch -d new-feature
```

This command will delete the branch named "new-feature" if it has been fully merged into the current branch. If the branch has commits that are not merged yet, you may use `-D` instead of `-d` to force deletion:

```bash
git branch -D new-feature
```

However, be careful with `-D` as it deletes the branch regardless of its merge status.

24. To create a new repository locally and then push it to a remote repository, you can follow these steps:

1. **Initialize a new Git repository locally**:
   Navigate to the directory where you want to create your new repository using the command line, and then run:

   ```bash
   git init <repository-name>
   ```

   Replace `<repository-name>` with the name you want to give to your new repository.

2. **Add files to the repository**:
   Add some files to your new repository by placing them in the directory you just initialized, or use `git add` to add existing files:

   ```bash
   git add .
   ```

This command adds all files in the current directory to the staging area.

3. **Commit your changes**:

   Commit the added files to the repository:

   ```bash
   git commit -m "Initial commit"
   ```

   This command commits the staged changes with the message "Initial commit". Replace the message with a meaningful description of your changes.

4. **Create a new repository on a hosting service (e.g., GitHub, GitLab, Bitbucket)**:

   Go to the website of the hosting service you want to use, log in if necessary, and create a new repository with the same name as the one you created locally.

5. **Set the remote URL for your local repository**:

   Go back to your command line and set the remote URL for your local repository to point to the repository you just created on the hosting service:

   ```bash
   ```

```
git remote add origin <remote-repository-url>
```

Replace `<remote-repository-url>` with the URL of the repository you created on the hosting service. For example:

```bash
git remote add origin https://github.com/your-username/your-repository.git
```

6. **Push your changes to the remote repository**:

Finally, push your committed changes from your local repository to the remote repository:

```bash
git push -u origin master
```

This command pushes your changes to the `master` branch of the remote repository. If you're working with a different branch, replace `master` with the name of your branch.

Now, your local repository has been successfully pushed to the remote repository on the hosting service. You can continue working on your project and use `git push` to push any new changes to the remote repository as needed.

25. To change the message of a previous commit that is already in the staging area in Git, you can use the `--amend` option with the `git commit` command. Here's a step-by-step guide:

1. **Stage the Changes**: First, ensure that any changes you want to include in the amended commit are already staged. You can stage changes using the `git add` command.

```bash
  git add <file1> <file2> ...
```

2. **Amend the Commit Message**: Once your changes are staged, you can amend the commit message using the `--amend` option with the `git commit` command. This will open your default text editor where you can edit the commit message.

```bash
git commit --amend
```

If you want to change the commit message without opening an editor, you can use the `-m` option followed by the new commit message:

```bash
git commit --amend -m "New commit message"
```

3. **Save and Exit**: After editing the commit message, save the changes and exit the editor.

4. **Push the Changes**: If you've already pushed the commit to a remote repository, keep in mind that amending a commit changes its history. If you've shared the commit with others, it's generally not recommended to amend it. However, if you're the only one working on the branch or if you're certain that it's safe to amend the commit, you can force-push the changes to update the remote branch.

```bash
git push --force
```

Be cautious when using `git push --force`, as it can overwrite changes on the remote repository and cause issues for collaborators.

26.