To check your name and email if you have configuration

1 Git config user.name
2 Git config user.email

3 Git config -- global user.name “your name”
4 Git config -- global user.email “your email”

- to initialise the project
5 Git init

- to check the status of the project
6 Git status

- to add the to staging from workspace
7 git add <file name>

To add multiple files staging from workspace
8 git add file1 file 2…

To add all files staging from workspace
9 git add .

To commit the changes to the repo
10 git commit - - m “message”

To commit the changes to the repo in a editor
11 git commit

To check all the commits
12 git log

to check all the commits in a oneline mode
13 Git log - - oneline

To unstage the file from staging area to work space back
14 Git rm - - cached filename

To amend commit
15 git commit - - amend

gitignore
Create a file - .gitignore
and add files that you don’t want git to track.

Phase 2 :
git branch

To check the head position
16 git branch

to create a branch
17 git branch branchname
Eg: git branch coffee

Switch branch
18 git switch branch-name

To create a branch and switch batch
19 git switch -c branch-name

20 git checkout -b branch-name

To view the information about each branch
21 git branch -v

To delete branch
Note : You can't delete the branch that you currently using
22 git branch -d branch-name

To delete branch forcefully without merging.
Note : You can't delete the branch that you currently using
23 git branch -d branch-name

Phase 3:

GIT MERGE

There are 3 kinds of merges we do.
1. Fast forwarding
2. Merge commit without conflict (with no forward commits in the other branch)
3. Merge commit with conflict. (With 1 or more forward commit in the other branch)

How to merge
1. Switch to the branch that you want to receive the merge.
24 git switch branch-name

2. Now merge the other branch into your branch
25 git merge branch-name

Phase 4 :

before staging:

To show the unstaged changes since last commit or
to show the changes in our workspace that are not staged yet for the next commit.
26 Git diff

to show all staged and unstated changes since last commit.
to show the changes in your workspace since your last commit.
Changes can be anything (before staging/after staging.)
27 git diff HEAD

After staging:
To show the changes between staging area and last commit.

28 git diff —staged
or
29 Git diff head

or
30 git diff - - cached

To show the changes between staging area and last commit in a specific file.
31 git diff HEAD filename
or
32 git diff —staged filename

To show changes between two branches
33 git diff branch1 branch2 git diff b1..b2

To show changes between two commits
34 git diff commit1 commit2

Phase 5

Git stash

to switch the branch without commit the current branch.
35 Git stash

To pop the stash
36 git stash pop

To copy the stash
37 Git stash apply

to list the stash list
38 Git stash list

to delete a specific stash
39 Git stash drop stash@{0}

to delete all stash delete
40 Git stash clear

Phase 6

Detaching
Time travel

To detach the head to a specific commit
41 git checkout commit -id (first 7 digits)

To do a new commit at detached head.
1. Create a new branch at detached head
2. Then do the commit
3. Now switch back to master or your branch
4. Merge it

or
1. Git branch branch name newcommitid
2. Git merge it to master or your branch

To detach the head to a specific commit using head

42 Git checkout HEAD~1

HEAD~1 - LAST COMMIT
HEAD~2- LAST BUT ONE commit

Reattach HEAD

43 Git switch branchname

Switching between branches

44 git switch -

To undo the unstaged or working stage changes

45. git checkout - - filename
or

45 Git checkout HEAD filename
or
git checkout HEAD~1 filename
or
Git checkout commit-id filename

To undo everything
46 git checkout .

To undo or discard the unstaged or working stage changes - RESTORE
47 git restore filename
To undo or discard the unstaged or working stage changes to a specific commit file- RESTORE
48 Git restore - - source commit-id filename
or
49 git restore - - source HEAD~1 filename

To undo everything

50. git restore .

To undo or discard the staged changes RESTORE

50 Git restore - - staged file name
or
51 Git restore - - cached filename

undoing a commit or deleting a commit but not the file changes

Eg : by mistake you have commited in a wrong branch.

52 git reset commit-id

permanently deleting a commit along with the file changes

53 git reset - - hard commit-id

deleting a commit in a different way- revert
1. it will create a new commit and there it will delete the file changes.

54 git revert commit-id

git clone <remote repo link>

Config ssh keys:

https://docs.github.com/en/authentication/connecting-to-github-with-ssh

Two ways you can link your git repo to GitHub.

1. **Push** your existing repo (Local machine) to the GitHub.
2. Start a project on Github and **clone** it to the local machine.

Lets see first one :

1. **Push** your existing repo to the GitHub.
    1. Create a **new repo** on Github.
        use GUI

    2. Connect your local repo to the **GitHub repo also called remote.**

        remote - destination link or remote repo url link or github repo url link.
        Note: by convention, it has to be origin.

        Note: to check remote details.
        git remote -v

        Adding remote :
        git remote add <remote-name> <remote repo url>

        Renaming remote :

        git remote rename <old-remote-name> <new-remote-name>

Removing remote :
git remove <remote-name>

3. Now push your changes to Github.

git push <remote-name> <branch name>

this will do two things.
1. First it will create a branch on the repo based on your command request.
2. Now it will push your last commit to the remote repo.

Eg:

You have a master branch in your local repo.

If you push it like ***git push origin jyotsna***

1. this will create a new branch jyotsna on the repo
2. Now it will push your last commit to the remote repo.

set upstream : linking local and remote branches.

Git push -u <remote-name> <branch-name>

1. Imagine you are in jyotsna branch in local repo,
2. and you want to commit to Jyotsna repo branch

Now as I am in jyotsna branch in my local machine, if I write,
Git push -u origin jyotsna.

Now as long as I stay in jyotsna local branch, I can simply use git push instead of git push origin branch name. Because the link has been established.

2. Start a project on Github and clone it to the local machine.

1. Create a new repo on Github
2. Clone it to your local machine
3. Work locally
4. Now you can push your changes.

2. git clone <remote-url>

Note: the default branch of git is master.
But the default branch of GitHub is main.

To check local branches:
git branch

To check remote branches:
git branch -r

To pull the remote repo changes to the local machine

git pull <origin-name> <branch>

Fetching :
You want to check or preview the changes before pulling. So that you can make sure your codebase is not messed up.

Git fetch <origin> <branch-name>

this will show you the new remote commits ahead of your local machine code.

How to check the code?
Using detached HEAD

git checkout <remote-branch name>

Eg: git checkout origin/main

Now you can check the code.and two cases you have here.
1. You liked it? Then pull the changes
2. You don’t like it? Then ignore it.

How to come back from detached head ?
Git switch <branch-name>

Note: Special case.

Person 1 has pushed a branch to the remote repo and when you pull it, it will show the changes in the remote branch,
But not in the local machine.
so if you want to see branches, do the following.

1. Git branch -r
    this will show you the list of remote branches.

Eg:
origin/main
origin/newbranch

2. Git switch branchname
eg: git switch new branch

Now this will create the branch new branch and shows you the code.

Git pull = git fetch + git merge

Collaboration problems:

1. Steve made a commit for registration file on master.
2. Bill is working on login file in his local machine and he needs support from Steve.
3. now the only way he can do this is by committing his incomplete code to master.
4. Now as this master latest commit has bugs, the website that Jack is providing for his customer is stopped.

5. Now as long as Steve works on the bill bug, the website won't be available for anybody.

Solution :
Feature Branches:
1. Treat master as official project history or branch where you push changes only when you are sure that the project has no bugs.

Eg: Create a party menu

Steve - Morning Menu
Bill - Afternoon Menu
Jack - Evening Menu

1.Steve will create morning menu branch
2. Bill will create afternoon menu branch
3. Jack will create evening menu branch

Now Steve added
tea
coffee

Meanwhile bill push his afternoon branch to remote but,
he wants Steves help in deciding the afternoon menu.

Now bill would pull the Steve branch,
And help bill with a new commit.

and so on...

Pull request:

We should not allow users to push the changes directly. In a real time scenario, we need
Someone who would scrutinise or verify your pull merge before merging it to master or
official branch.

Contributing to opensource with fork and clone.

1. Fork the project
2. Clone the project
3. Set upstream remote to the Open-source project so that you can pull the changes to the local machine.
4. Push the changes to the origin
5. Now create a pull request.

Rebasing : alternate for merging. But don’t use

```
git switch newbranch
Git rebase master or main
```

this would move all the new branch commits to the tip of master branch.

Interactive rebase

To change the commit details.

```
git rebase -I commit-id or HEAD~id
```

Git tags :

Simply pointer that. Refers to particular phases of your project. Usually we use it name the version.

there are two types
1. Lightweight tags

How to create a tag?

```
git tag tagname
```

To list the tags

Git tag -l *write version name or common tag name*

2. Annotated tags

How to write version names or semantic verisoning

2.3.1

2 is major release : major feature released
3 is minor release : minor features released
1 is patch release : bugs fixes

to check the status of tag:
Git checkout tag

Annotated tags creation

git tag -a <tagname>

Creating tag for previous commits

git tag <tagname> <commit>

deleting tags
Git tag -d <tagname>

pushing tags : it has to be seperate

Git push --tags

Git reflogs:

Git reflog

this will show all the details of project like at what time what you have done.

Git reflog show HEAD

Git reflow show main

git reflog head/branch name@{qualifier}

Time travel

git reset - - hard <reflog commit id>