# Git

* git comes with a tool called [config](#config) that sets configuration and controls how Git looks and operates.
* if you ever need [help](#help), there is a manual page (man page) in git
* whenever we require to [initialize](#initialize) a repository, there are two main approaches: The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.
* files in our working directory are in two states: tracked and untracked. Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged. Untracked files are everything else – any files in your working directory that were not in your last snapshot and are not in your staging area. The status of a file can be checked using [status](#status) command
* to exactly know what [difference](#difference) we made in a file or what changes we made in files, we use *git diff* command.
* after the staging area is set up, we can [commit](#commit) our changes. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot; the commit records the snapshot you set up in your staging area. Every time you perform a commit, you’re recording a snapshot of your project that you can revert to or compare to later.
* To [remove](#remove) a file from Git, you have to remove it from your tracked files (more accurately, remove it from your staging area) and then commit. The git rm command does that, and also removes the file from your working directory so you don’t see it as an untracked file the next time around. If you simply remove the file from your working directory, it shows up under the “Changed but not updated” (that is, unstaged) area of your git status output:
* if we want to [rename](#rename) a file in git, we use *git mv* command.
* to look back the [history](#history) about what has happened, we use *git log* command.
* git offers the feature to [undo](#undo) the changes we made
* we can [unstage](#unstage) a staged file
* to collaborate on any Git project, we need to manage [remote](#remote) repositories. Remote repositories are versions of project hosted in internet. Generally, remote handles the short form of the project that we are currently working on.

git config:

to set up username for all the repositories:

* git config - -global user.name “username”

to set up username/email for that repository only

* git config user.name “username”
* git config user.email “name@email.com”

to set up user’s email for all the repositories:

* git config - -global user.email “name@email.com”

to update the default editor:

* git config - -global core.editor “vscode”

to view the config list

* git config - -list

getting help:

there are 3 ways to get the manual page (man page)

* git help <verb>
* git <verb> - -help
* man git -<verb>

for eg:

-to get help for <verb >=config

* git help config OR
* git config - -help OR
* man git -config

**initializing a repository**

track an existing project/directory i.e at local

* git init

to track the files, basically following commands are used

* git add .
* git commit -m “message”

getting copy of the existing project form Git server

* git clone “https://github.com/USERNAME/PROJECT.git

checking the status of files

* git status
* git status -s // - -short

viewing your staged and unstaged changes

* git diff
* git diff - -staged
* git diff - -cached

committing changes

the simple way to commit changes using our preferred editor

* git commit

to see exactly what changes are we committing

* git commit -v

type your commit message inline

* git commit -m “MESSAGE HERE”

skip the staging area i.e. to skip the git add part

* git commit -a -m “MESSAGE HERE”

removing files

remove tracked files

* git rm “FILENAME”

if the file has been modified and added to the index, remove it forcefully

* git rm -f “FILENAME”

keep the file in working area but remove from staging area

* git rm - -cached “FILENAME”

the following command removes all files ending with “.txt”

* git rm \\*.txt

renaming / moving files

the syntax would be

* git mv FILE\_FROM FILE\_TO

viewing the commit history

most recent commits shows up first

* git log

show the differences introduced in last two commits

* git log -p -2

this option changes the log output to formats other than default

* git log - -pretty: “%cn: %t: %h”

similarly, there are many forms about how to use the log command

* git log - -stat
* ……………………

undoing things

to try that commit again, (if you had made no changes since last commit)

* git commit - -amend

unstaging a staged file

unstage a recently staged file

* git reset HEAD “FILENAME”

calling git reset without option is not dangerous, it only touches staging area, but with option - -hard, it is.

* git reset - -hard “FILENAME”

discard changes in working directory, (the command copies another file over it)

* git checkout - - “FILENAME”

working with remotes