GIT COMMANDS:

Date: _____ Page No. _____

Topic: _____

- git init
- git status
- git add .
- git commit -m "message"
- git remote add origin <URL-link>
- git push origin master
- git clone <URL-link>
- git pull <URL-link>
- git branch <branch-name> [-D]
- git checkout <branch-name>
- git log
- "git stash" && "git stash pop"
- git revert <C-ID>

08/07/25.

Date: _____ Page No. _____

Topic: _____

GIT

- Git is the most popular tool among all the DVCS tools (Distributed Version Control System).
- used for tracking changes in computer files & coordinating work on those files among multiple people.
- Primarily used for source code management in software development, but it can be used to keep track of changes in any set of files.
- Git is a software, falling under DVCS tool.

09/07/25.

Date: _____ Page No. _____

Topic: _____

⇒ How GIT Works :-

- Following tasks are common, when working with git.

1.] Creating Repository :-

- Create a git repository using "git init".
- this initialize a git repository for your project on the local system.

- mkdir project
- cd project/
- touch 1.txt & touch 2.txt
- ls
- clear
- ls
- git init
- Initialized empty Git Repository in ...../project/.git/

Date: _____ Page No. _____

Topic: _____

2.) Making Changes :-

→ track files status using "git status", whether the files are being tracked by git or not.

- ls
- git status
  - On branch master.
  - No Commits yet.
  - Untracked files:
    .....
    .....
  - use git add <file> to track.

→ No files are being tracked, to track a file use "git add <file>" or for tracking all files are "git add .".

- ls
- git add 1.txt 2.txt or
- git add .
- git status

Date: _____ Page No. _____

Topic: _____

Now our files are in staging state, files are not yet SAVED in git. to do so use commit.

→ Once the files or changes have been staged, we are ready to commit them in our repository using "git commit -m " message here type mario ".

-m ⇒ is for typing the commit message.

- ls
- git commit -m "Commiting 1.txt & 2.txt."
- git status
On branch master
nothing to commit, working tree clean.
- touch 3.txt
Untracked file is now 3.txt.

Date: _____ Page No. _____

Topic: _____

- cat >> 2.txt
  Hello
  Modified the file 2.txt that was already committed.
- git status
  Not staged to Commit:
  modified : 2.txt.
  Untracked files:
  3.txt.
- the only way to commit them are to first add, then stage, & then Commit.

- git add 3.txt
- git status
- git commit -m "added 3.txt & modified 2.txt"
- git status
  working tree clean.

Date: _____ Page No. _____

Topic: _____

3.] SYNCING REPOSITORIES :-

→ How we'll make it available to the world ?

- Once everything is ready on our local, we can ~~stat~~ start pushing our changes to the remote repository.

- Copy your ~~res~~ repository link & paste it in ~~to~~ the command:

- git remote add origin "<URL to repository>"

- Go to github.com sign up | login > click new repository > give repository name, make it public > create repository.

- copy the HTTPS or SSH (Secure shell) link and paste it in the Command.

Date: _____ Page No. _____

Topic: _____

→ to push the changes to your repository, enter

- git push origin <branch_name>
- in our case branch_name is master.
- git push origin master.
- it can also request you to enter github username & password, provide it.

- BOOM! go & check the github repo. changes & files are reflected / pushed over there.

→ // git clone //.

- if we want to download the remote repo. to our local system, use "git clone <URL>".

- this will create a folder with the repo. name & downloads all the files into this folder.
- Ex:- demo-repo.

Date: _____ Page No. _____

Topic: _____

- go to github copy the url of any repo you want to clone.
- git clone https://github.com/..... demo-repo/ # added.
  ↳ check using "ls".
- if we clone this and create 5.txt, then commit it > push it → done Successful even without using the remote Command.
- because when cloning the repo. it copy the parameters as well as to what my origin repo. was.
- file added from clone repo. will not be in our local repo. but it will be in the github repo, from where we can pull it using.
- git pull origin ~~ma~~ master.

Date: _____ Page No. _____

Topic: _____

- local repo will not get automatically updated, those using "pull" command.

- git pull <URL of link>

- this command only works in the (inside) ~~the~~ an initialized git repo.

- used to pull the latest changes, that might have pushed to the remote repo..

⇒ GIT PULL :-

it updates the local repository, from remote repo..

⇒ GIT CLONE :-

it copies / clones the whole repo. from remote to local.

Date: _____ Page No. _____

Topic: _____

4. PARALLEL DEVELOPMENT :-

- we saw, how to work on git.
- Imagine, multiple developers working on the same project or repo..
- To handle the workspace of multiple developers, we use branches.
- To create a branch from an existing branch, use :-

```
git branch <name of new branch>
```

- to delete a branch, use :-

```
git branch -D <branch-name>
```

- to list all branches, use :-

```
git branch
```

Date: _____ Page No. _____

Topic: _____

- create & switch to that branch :- , use "checkout".

- LAST Commit of master, becomes FIRST Commit of our new branch.

- git checkout <branch-name>

- Branches are isolated from each other.
- to Combine them we use "merge" Command.

- to see which branch we are in :-
  git branch
  * feature1 ✓ * denotes it.
    master
    branch1

- git checkout master # to go from current branch to master branch.

- No Branches are interfering with each other.

Date: _____ Page No. _____

Topic: _____

- Ex :-

- create a branch feature1.
- check all branches 2 right now :-
- * master
  feature1
- checkout to feature1 (git checkout feature1).
- * feature1
  master
- do ls in both initially same files.
- create 6.txt in feature1 & add it, commit it & push it to the feature1 branch.
  - git add .
  - git commit -m "added 6.txt from feature1 branch"
  - git push origin feature1
  → new branch created & pushed file 6.txt as well from feature1 to feature1.
- check "git status", working tree clean. on branch feature1
- git checkout ls.
- total 6 files (5 branched from master & 1 created).

11/07/25.

Date: _____ Page No. _____

Topic: _____

- git checkout master.
- ls
- total 5 files that were already there, (6.txt is NOT added).
- check status : on branch master, working tree clean.

⇒ #

- to check from github select on branch choose master (only 5 files visible), select branch choose feature / (total 6 files visible.)

→ || git log ||

- checks every commit detail in the repo..
- to check the history of master branch :-
- ONLY the LOGS of master.
- git checkout master
- git branch → (to Confirm).
- git log.

History of everything in master from 1st file to latest Commit / changes.

Date: _____ Page No. _____

Topic: _____

Logs

- to check in feature1 branch :-

- git checkout feature1
- git branch # optional.
- git log

- Logs of master branch as well as logs of this branch (feature1).

- Because feature1 was created from master branch, so it will have all the logs of master branch as ~~well~~ well.

- ~~LOGS of MASTER branch~~ is

- MASTER branch shows LOGs of ONLY MASTER branch &

- FEATURE1 branch created from master branch will have LOGs of master branch as well as LOGs of feature1 branch itself.

Date: _____ Page No. _____

Topic: _____

// git stash. //

- to avoid dangling files moving between branches.

- Ex :- In our case we have 2 branches :-
  master → 1 to 5.txt files (5).
  feature1 → 1 to 6.txt files (6).

- created 7.txt in feature1.
- ls → total 7 files.
- changing to master branch.
- ls → total 6 files (6.txt) is reflected here.

⇒ This is known as dangling file.

- On working in a production level environment this may get create confusion, to avoid this we use 'git stash' & 'git stash pop'.

Date: _____ Page No. _____

Topic: _____

- Ex :- modifying 5.txt in feature1 :

then, check in master branch. the changes will be reflected in the master branch.

- this is not staged (git add .), from feature1.

- then if we change & check in master the changes are reflected even after staging the files.

- ~~to avoid this use :-~~

- showing :-

M 5.txt. Modified.

A 7.txt. Added.

- to avoid this :-

- git stash

- saved working directory & index.

Date: _____ Page No. _____

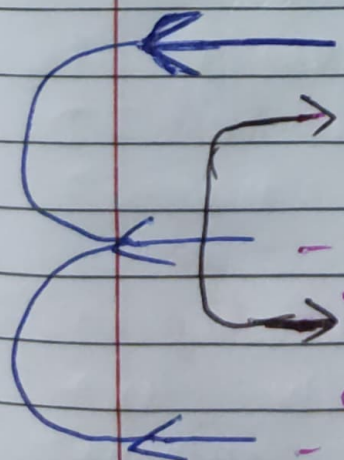
Topic: _____

- git stash (if used again).
- No Local changes to Save.

- ls (in feature1 branch).
  - total 6 files.
  - No Changes in 5 (even if we have changed).

- ls (in master branch)
  - total 5 files.
  - No changes in 5 (even if we have changed).

- ~~#~~ The main reason to use stash is,
  - while working in a branch,
  - we got to do some imp. work in the another branch.
  - the first branch is Not Completed.
  - So, we use stash & checkout to another branch - finish Work.
  - Go back to first branch & use "git stash pop" to resume our work from there.



Date: _____ Page No. _____

Topic: _____

- git stash pop

  Dropped refs/stash@{0} (.....).

- "Saves Our Work without Committing the Code?"

  - Use 'git stash'.

- to stash your untracked files, type :-

  - Use 'git stash -u'.

- git stash.
- git stash -u.
- git stash pop.

C-ID → COMMIT-ID.

Date: _____ Page No. _____

Topic: _____

// git revert <commit-id> //.

- Helps you in reverting a Commit, to a previous version.
- 'git revert <COMMIT-ID>'
- <COMMIT-ID> Can be obtained from the "git log" Command.
- ls - in feature1 $ → total 7 files.
- git log - copy the id you want to revert the Commit.
- git revert <C-ID> - it will ask for the Confirmation. In VIM - press ESC > :wq > enter.
- Successfully reverted.
- ls - to check or use.
- git log - shows the message of reverted with C-ID.
- Copy this C-ID & enter :- git checkout C-ID. → to enter the reverted message & see what was reverted.

Date: _____ Page No. _____

Topic: _____

- in place of branch name you will see the C-ID of reverted commit.

- to exit that checkout press :-
git checkout feature;

- ~~That~~ This command will take us to the feature1 branch that now has 6 files & the old content of 5.txt (this was the changes in the reverted C-ID thats way).

- ls - to check.
cat > 5.txt - to check the content.

- Reverting means Undoing the changes done in that C-ID.

// git diff //

- check the difference B/w two versions of a file / two commits.

Date: _____ Page No. _____

Topic: _____

- git diff <C-ID of version X>
  <C-ID of version Y>

- C-ID → get it from → git log.

- git diff Read .

Read → ~~for~~ latest Commit of Current HEAD (branch)
. → Current changes untracked/ unstaged.

- git diff db785..... 56799DFdez...

db785... → changes from this C-ID to
56799DFdEZ... → to this C-ID.