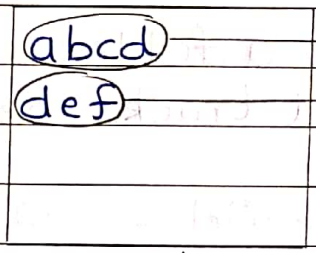
30/05/2023

Version control

Suppose that we have created a word document and in this we have written some text. Now sometimes we use ctrl+z (undo) to go back to previous state. This going back is possible as OS (operating system) is doing version controlling.



Word doc

When we hit ctrl+z, we go back to version 1 from version 2 and hence this is known as version controlling.

Note→ If there was no version control, then we have to do undo manually.

Important points related to version control

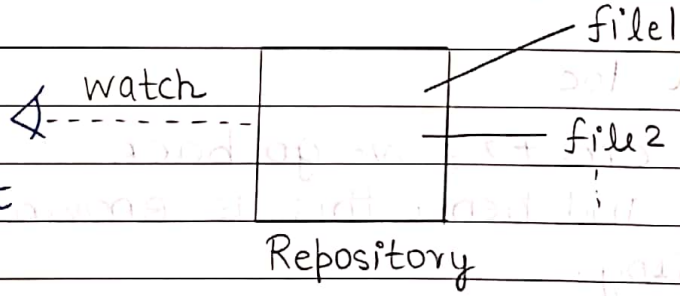
1) It makes save points that save your project.
2) It provides total development freedom. Suppose that our project has a user authentication feature and now we have to add the product authentication. We added many lines of codes in the codebase but then some senior developer came and asked to remove it. Here the version control was keeping the track of the changes and hence it can be easily removed/deleted.

Ex → Git is a version control tool. Savepoints are basically the commited changes which are working and by these savepoints we can go back to previous state when we do undo.

Git vs Github

1) Git is a version control tool whereas Github is a service to host git repositories.
↳ (folder)

Repository is basically a folder on which git tool can watch and track the changes made in that folder.



2) Github is a pool of repositories. Suppose a person P1 is from India and P2 is from US and P1 and P2 want to collaborate on a particular project, then that project is hosted on internet and this is done via github.

Installation steps for git

1) Go to google.com and type git
2) Open the 1st website named as git-scm.
3) Click on download for windows
4) Install for 64 bit system.
5) Click next until we see select components.

6) In select components, tick mark add git bash profile to windows terminal.
7) Click next and then finish.

Verifying the installation
git --version to be run on command prompt.

This will tell the version of our git, if it is installed.
Some more steps to complete setup
On command prompt, enter the following commands.

* git config --global user.name "BhavyaBhalla-27"
* git config --global user.email "bbhalla270104@gmail.com"

Verification of name and email
git config --list

We can see the above mentioned details in the last 2 entries.

Installation steps for github desktop
1) Go to google & type github desktop.
2) Simply click 1st link and download for windows
3) Open it and login with the github details
4) Now click on configure manually and click finish. (Details will be added automatically as we have mentioned from command line.

Creating a git repository

1) Create a folder say on desktop
2) Open command prompt now.
3) Migrate to the folder which we have created by
   ↳ change directory
   cd Desktop
   cd git-dummy
4) Now we are in folder git-dummy. Now in command prompt, enter

   git init → To convert to git repository

5) Now there will be a .git folder which is hidden. To see this folder we need to click on view → Show → Hidden items

Knowing about .git folder
It contains meta data of the git repository.

Folder + (.git folder) = Repository

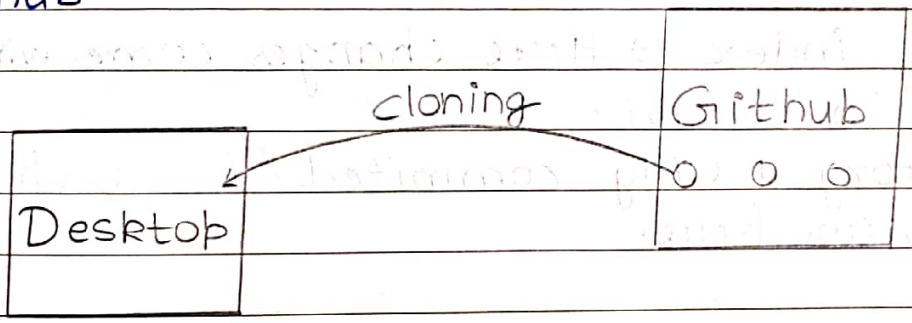
git status → Updates will be tracked
To know about the git repository.

O/p
on branch master
no commits yet
nothing to commit

Now create a notepad file & enter any random text and save in that folder.
Now run git status command.
Now the output will be different.

on branch master
no commits yet
untracked files
- - - -
nothing added to commit but untracked files present.

git clone command
This command is used if we want to collaborate on some project which is hosted on github.



To know the URL of that github repo, simply click on code and then copy the URL.

git clone (url) (myRepo/)
↳ copy from github
↳ folder name in which repo will be cloned.

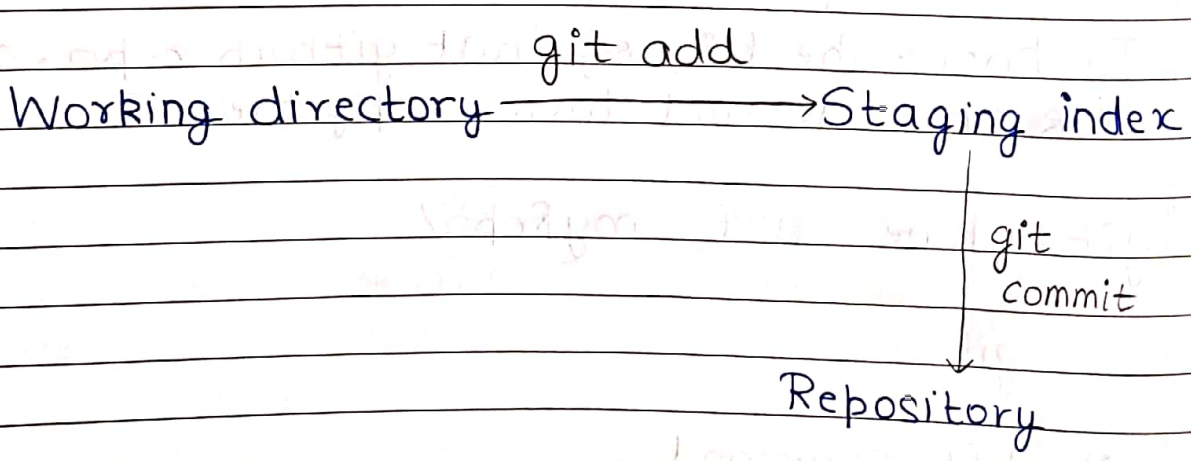
git diff command
This command helps to know what changes have been done in our repository. File name should also be specified. (git diff fname)
file name

Lifecycle of a change
Here we will get to know what all things happen when we do some kind of changes.

Note → Commit means final change.

Repository (commited)	
Working directory (Temp. changes)	Staging index

1) Working directory → Here temperory changes exist.
2) Staging index → Here changes come which are about to commit.
3) Repository → Only commited / final changes will occur here.



Note → Each commit we do have some commit id which helps us to distinguish blw old & new commit.

Reviewing a history in git repo
In this our main aim is to see what all commits we have done.

1) git log
This command will give us the latest commit and then we go on to the old commits. We can see here the commit id, author, date and commit message.
↳ sha id

Note - Commit id is some long number also having characters.

To exit this, press q.

2) git log -3
↳ can be any number
This command will show only latest 3 commits.

3) git log -p
This is combination of git log and git diff. We can see what changes have been made.

4) git log --oneline
Only commit id and commit message will be shown & that too in single line.

5) git log --stat
Commit id, author, date and what files have been changed will be shown.
↓
file names only.

6) git show sha id
↳ commit id
This command is used to know what all changes have been made in a particular commit.

Procedure to do commit

1) Create a folder say on desktop. Migrate to that file and run command git init.
2) Save a notepad file in that folder.
3) To track the added/new file, we need to run git add command.

file name
↓
git add (main.cpp)

4) Now main.cpp is in Staging index. Now we need to run git commit command.

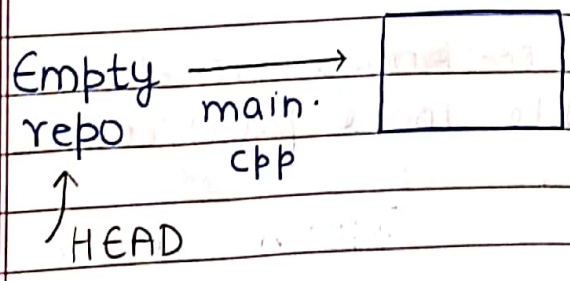
git commit -m "file added"
commit message

Note → commit message should be a meaningful message.

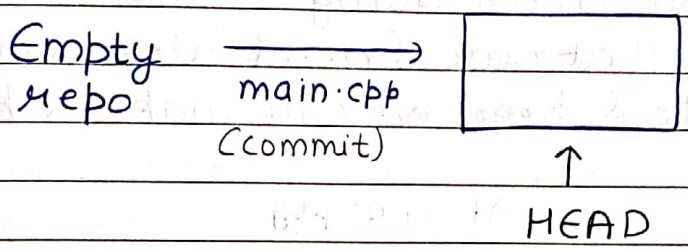
git add
- → new file (add files for tracking)
- → old file (goes to staging index)

git add . → period → This will put all the files to staging index however it is not recommended to use this command and use name of file instead of .

HEAD



As we commit main.cpp, HEAD moves to latest commit



Discard changes from working directory
To do above task, we need to use git restore command & restores it to the latest commit i.e HEAD

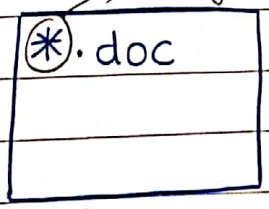
file name

git restore (fname)

Track changes from particular files only
Git repo ⇒ .js → } Track them
.css →
.doc → Don't track

For this we need to make a file .gitignore

any pattern / name will be ignored



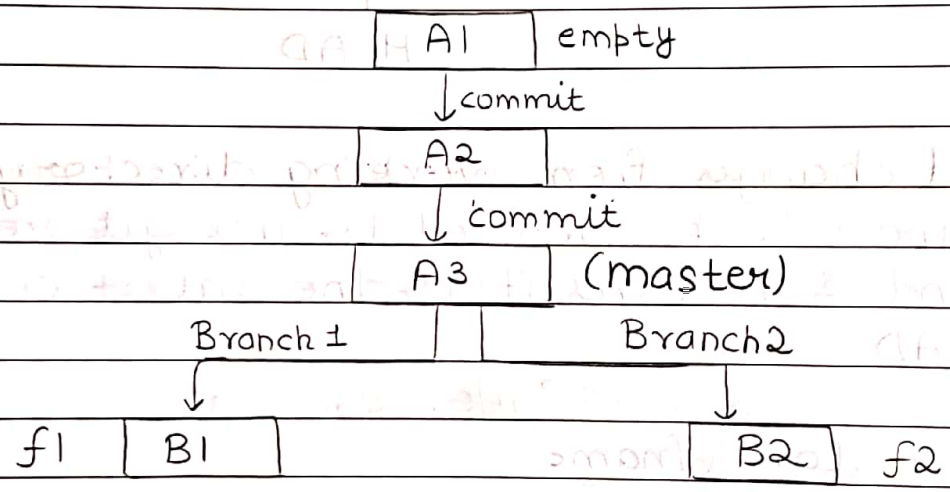
.gitignore

Note → We will commit .gitignore file also.
We can add a particular file name or some path in the .gitignore file.

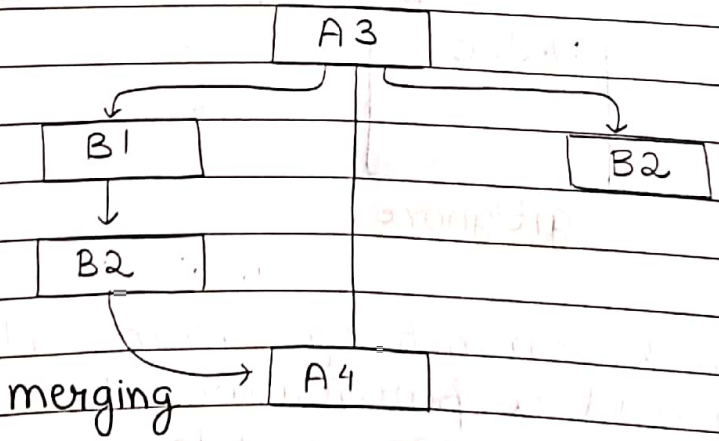
assets/images/*.png] Ignore all .png files / images

Branching, Tagging & Merging

Initially we do git init and it is an empty repo. After doing some commits, we think that our friend also want to collaborate & here we can make 2 branches.



In both branches B1 and B2, code of A3 will go. But now say in branch 1, changes were made in file 1 but these changes made in file 1 won't be reflected in file 1 of branch 2. Same is the case of file 2 in branch 2.



We are merging in the master branch so that the master branch stays updated.

1) Creating a branch → name of branch
git branch (hello)
The above command will create a branch named hello. If we don't mention the name of branch, then this means we are demanding the no. of branches & the branch which we are will be highlighted.

2) Switching blw branch → name of branch
git checkout (hello)
To move from master branch to hello branch.

Note → There is no change in procedure for commit in other branch.

→ name of branch
* git checkout -b (hello1)
The above command will create a branch named hello1 and then switches to hello1.

Add and commit in single command
git commit -(a)(m) "file added"
add message commit message

3) Merging the branches to master branch
git checkout master
git merge (hello)
↳ branch name
Now hello branch will be merged to the master branch.

4) Deleting a branch
git branch -d (hello)
↳ name of branch

Merge conflict

master
↳ main.cpp

```
1
2
3
4   main(){
      ---
    }
```

hello
↳ main.cpp

```
1
2   //Hi
3
4   main() {
      ---
    }
```

Merging hello and master

master
↳ main.cpp

```
1
2   //Hi
3
4   main() {
      ---
    }
```

hello!
↳ main.cpp

```
1
2  // Hello
3
4  main() {
     - -
   }
```

Merge hello1 and master but at line 2 there was // Hi written already & hence git got confused. This is known as merge conflict.

Resolving merge conflict

We need to manually fix the thing and then use

```
git add main.cpp   ] 1st step
git commit -m "file added"   ] 2nd step
```

Note → When we would see the details of commit, we will see the merge field also mentioning sha id of both commits in different branches.

5) Tagging a commit

Suppose that we want to tag a commit. Tag means that if we want to say that a particular commit is a β-release. One way is mentioning this in description but better way is to tag a commit.

```
git tag -a betaV1.0 (sha-id) -m "Beta release"
                    ↑ commit id
```

Now if we enter git log command, then its tag will be visible.

6) Deleting a tag
git tag -d betaV1.0
name of tag

7) git stash
Suppose that we have some modified files in working directory. Now at remote, other user have made some changes in the same file. When we will do git pull, the changes that we have done will be lost as the changes done on remote are considered as final by git. To avoid this, git stash is used as this will take the modified files to some other area & working directory will be empty. Now when we pull, the changes made by us won't be removed as they are present at some other place. Now working directory will have latest files from remote. Now apply git stash apply command to bring back the files from stash area & now it is possible that merge conflicts have occurred which need to be resolved.

To see the stash area, use command
git stash list

Note→ Good practice is to do commits on other branches and not on the master branch.

Undo commits
It rarely happens that we will be removing the commits.

1) git commit --amend
Amends the most recent commit (only latest commit)

2) git revert
It will revert given commit. We have to mention the commit id.

git revert commit id
↗ sha id

We have to press esc, then colon (:), w, q and then enter.
It is a safe operation

3) git reset
This command will delete a commit and this moves the HEAD to given commit id.

git reset --soft commit id
↗ sha id
↗ (staged changes)

* --soft ⇒ Suppose that some files were present in working directory, now when --soft is used it will not delete working directory changes.
* --mixed ⇒ HEAD will be reset and changes in working directory remains intact but show as modifications.
* --hard ⇒ working directory changes will be removed.

Note → After entering the 1st command, we have to press i and then enter the new message and then press esc, colon, w, q & then enter.

Pushing the commit

1) Make a git repo locally on the system.
2) Save any file say readme.md
   ↳ description about repo
3) Use git add and git commit commands.
4) git remote add origin (url)
   ↳ mapping blw remote and local
   ↳ from github while creating repo
5) Set details by commands used on Pg 3
6) git push -u origin master
   ↳ To push on remote

Problem that can occur while pushing

username → Bhavya Bhalla-27
password → token

Generating token

1) Settings
2) Developer settings
3) Personal access tokens → Tokens classic
4) Generate new token.(classic)
5) Any name + any expiration + click all the permissions.
6) Generate token