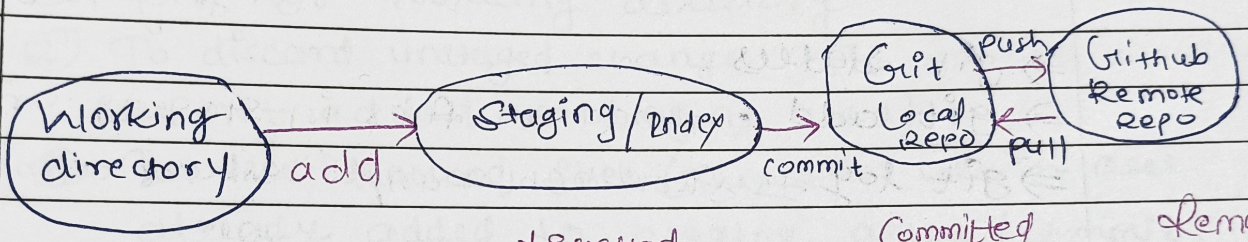
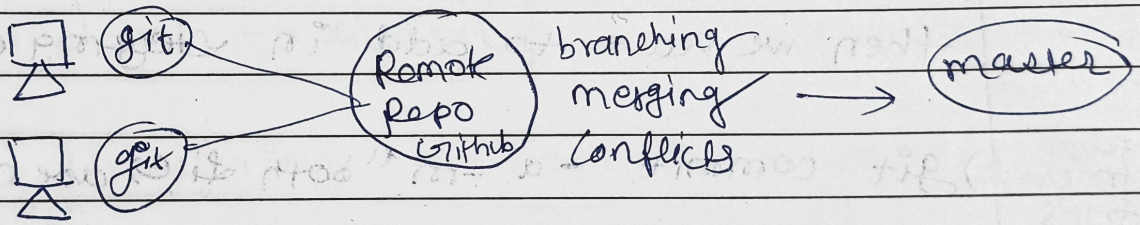
Date: 07/05/24

Git Version control System →
there is two types of VCS → ① centralized ② distributed
problem with centralized VCS is that there required N/w to push & pull code
But in Distributed VCS, here is developer have their own local repo then they can push / pull code to central repo. which is github.

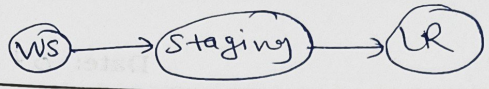
Grit → git is distributed VCS which support distributed environment. open source



untracked file | tracked file | Committed file | Remote file



ls → list of all files
Pwd → current working directory
git --version → current version of git.
cdrive → rightclick open git bash → show current
create new files in folder
ls →



Date: / /

git init # initialize empty repository
git status # show status of all tracked/untracked files
git add -a # add all files from WS to stage
git add a.txt # add specific files
git status # Now added file is ready for commit
git commit
before commit need to do some configuration cmd
who is commiting & all (one time activity)
⇒ git config --global user.email "
⇒ git config --global user.name "Rohini"
⇒ git commit -m "comment" # move file from Staging to LR
⇒ git status
⇒ git add . # add all files in staging area
⇒ git log # show how many files are committed

(*) Modify files
git status # files are modified
then we have to add in staging area then LR

used for only tracked files ⇒ git commit -a -m "both files are modified.."

⇒ ls # show all files from working directory
⇒ git ls-files # show files which is added in stage

RM

* Req 1 Remove files from both staging & working directory.
⇒ git rm file1.txt
⇒ git rm -r . # remove all the file from both

* Req 2 Remove files from only from staging
⇒ git rm --cached file2.txt
↑ represent stage area

Req 8:

⇒ Delete file only from WD

~~git~~ rm files.txt

(*) git checkout command

tracked files → files atleast once added in a staging or LR

untracked files → files never added in staging/LR

We can use checkout command to discard unstaged changes in the tracked files of working directory

① only for working directory

② To discard unstaged changes (The changes which are not added to staging area)

③ In the tracked files (The files which are already added to staging area (commit)

checkout → undo

```
git checkout -- file1.txt
```

* Git Branching command

```
git init
touch a.txt
ls
git add a.txt
git commit -m "a.txt"
```

① `git branch` # use to view available branches

`git status`

② `git branch branchname` # create new branch

③ git checkout # switch to from one branch to another branch

`git checkout branchname`

Date: / /

git branch

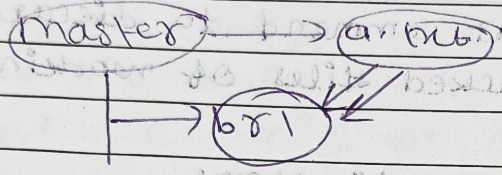
* mark will be shown infront of active branch

④ git checkout -b branchname

** creating and switching to branch

git branch

master → child branch (inherited files from master to child) but vice versa not possible.



whatever the changes happened in the branch will not be impacted in the master branch after creating the branch after done any new changes in master will not be affected in child branch.

all the branches are independent.

* Merging

when do parallel development then complete development for feature branch then code will be combine with the master branch.

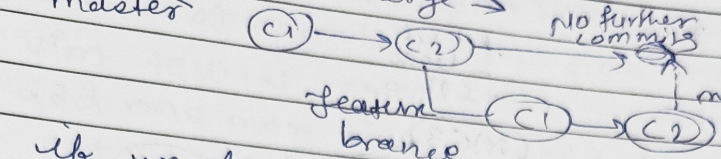
two types of merging are there →

① fast-forword merging.

② three-way-merge. (Recursive Strategy)

Date: / /

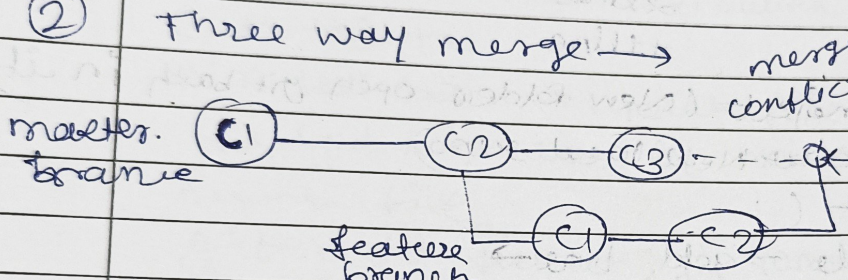
① Fast forward merge →



→ If we don't do any changes in the master branch then there will be no conflicts to merge code from feature to master.

→ We do only changes in the child branches don't touch master branch.

② Three way merge →



If we are doing parallel changes in master as well as in feature branch then there will be conflicts in merging may or may not be conflicts.

If you update existing file then definetely you will get conflicts

New merge commit will be created by git.

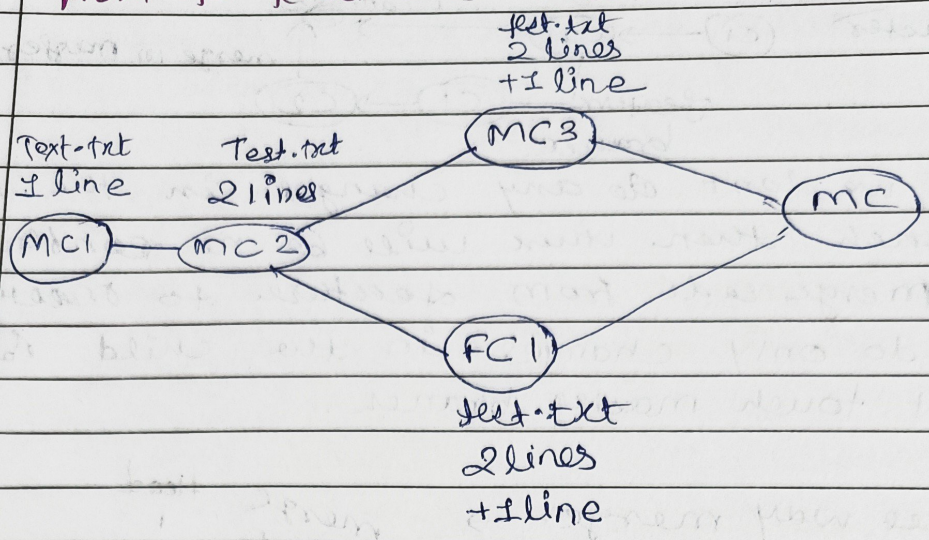
↳ git branch

git merge name_of_file

:wq! # save file & exit

Date: / /

* HOW to Resolve Conflicts →



create New project (New folder open git bash in it)

↳ git init (create New local repo)

vim test.txt (:

Insert mode → add line →

:wq! # Save file

Cat test.txt # see content of file.

git add test.txt ; git commit -m "mc1"

done 1 commit in master branch.

vim test.txt (# open file)

add text

git add test.txt ; git commit -m "mc2"

(xx) if you do any changes you need to do commit

git log --oneline # show how many commits are happened in master branch

Create feature branch →

git branch feature # branch created

git checkout feature # Switch branch from master to feature

(xx) whatever the commits are happened in master that will automatically add in feature branch.

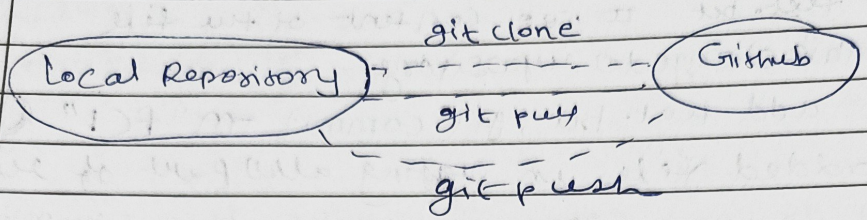
Date: / /

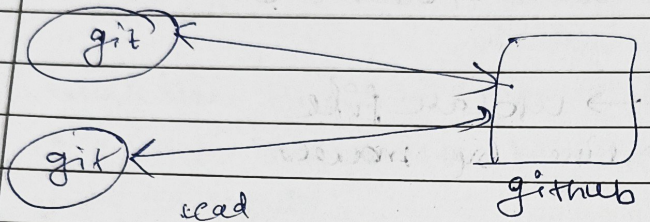
(*) git log --oneline # see no. of commits in feature branch.
same no. of commits inherit from master to child branch
vim test.txt # open file & edit
add one more line in child branch (update file)
cat test.txt # see content of the file
add this change to repository.
git add test.txt ; git commit -m "FC1" (committed file)
added file in staging also part of repo
Now do some changes in master then commit in master
git checkout master # switch branch to master
Cat test.txt
vim test.txt → update file
added new line by master
Save file
git add test.txt ; git commit -m "MC3"
git log --oneline
merge feature file in master
git merge feature # merge Not successful
open the file →
Cat test.txt
Solve conflicts manually
vim test.txt
remove unwanted file & then save file (dd) :wq!
cat test.txt
git add test.txt ; git commit -m "MC"
git log --oneline
git log --oneline --graph # see results in graph
git branch
git branch -d name-of_branch # delete branch

Git → local Repository Date: / /

Github ⇒ Remote Repository (Hosting service for git)

- It is ^hosting service for git repositories.
- git is the tool, while github is the service to use git.





read

Clone → ^create project & ask team to clone project then user will do some changes then push code into github (Remote repo)

Clone → create local repo based on remote repo.

push → whatever code done in local push into remote

Pull → if there is any new changes in remote repo then if you want to get those changes into your local repo then pull is used.

github

Clone → create folder → open folder → open git bash

cmd → git clone "github project url"

cd testrepo

modify existing file → ls → cat myfile

vim myfile ## modify file

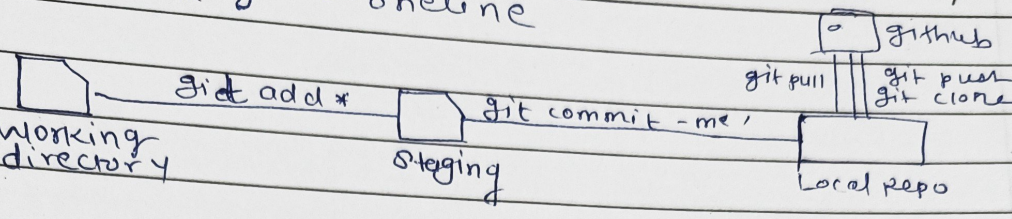
vim myfile2

git add . ## add all changes to staging

git commit -m "updated files"

git log --oneline

Date: / /



git push # push code from local to ~~Go~~ Remote

↳ git push origin 'branch name of Remote'

get copy of remote repo into local repo ⇒ clone

git pull ⇒ we have already copy in local but we want some changes happen in remote to local then use pull.

↳ git pull origin ~~repo~~ 'branch name of Remote'

Selenium with Python

Date: / /

Identify element.

Action

- id
- name
- linktext
- Partial linktext

} use to find single element

- classname
- tagname

} use to find multiple element

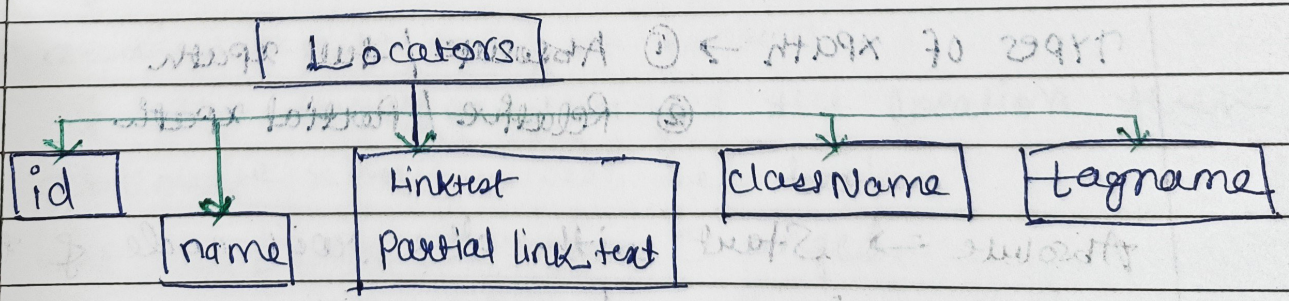
CSS selectors

1. tag id — tagname#valueofid — input#email
2. tag class — tagname.valueofclass — input.inputtext
3. tag attribute — tagname[attribute=value] — input[date="email"]
4. tag class attribute — tagname.valueofclass[attribute=value]

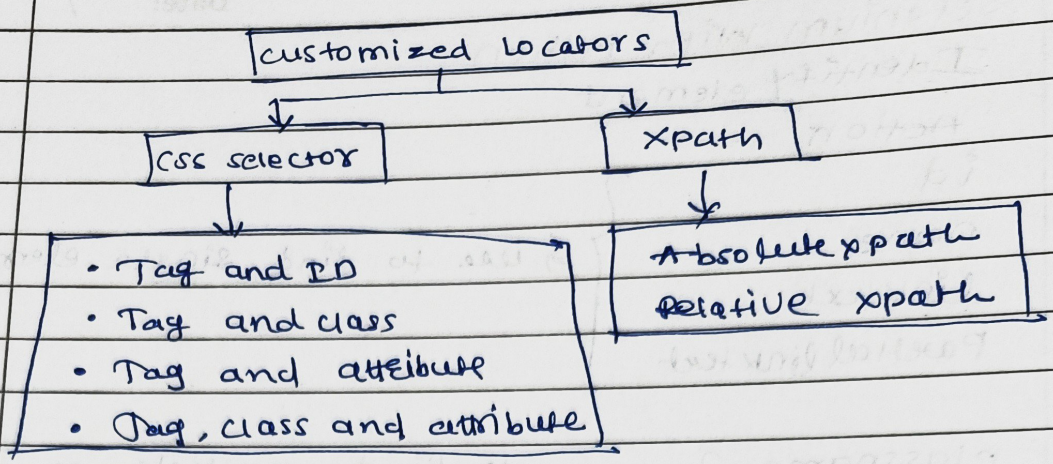
Xpath →

We can identify various elements on the web using locators.

Locators are addresses that identify a web element uniquely within the page.



Date:



xpath →

- xpath is defined as xml path.
- It is a syntax or language for finding any element on the web page using xml path expression.
- xpath is used to find the location of any element on a webpage using HTML DOM structure.
- xpath can be used to navigate through elements & attributes in DOM.
- xpath is address of elements.

DOM → Document object model

- DOM is an API interface provided by browser.
- When a web page is loaded, the browser creates a Document Object Model of the page.

TYPES of xpath → ① Absolute / full xpath
② Relative / Partial xpath

Absolute → Start with the root node & then it will go in each & every node

Relative → directly jump to that element

Date: / /

diff. betn Absolute & Relative xpaths

① Absolute xpath starts from root html node
Relative xpath directly jump to element on DOM

② Absolute xpath start with /
Relative xpath starts with //

③ In AX we use only tags / nodes
In RX we use attributes.

④ How to write xpaths manually

Syntax of writing relative xpath →
//tagname [@attribute = 'value']
//input [@id = 'name']

How to capture xpath automatically →
① Right click on element → inspect → right click → copy xpath
② Selectorshub
Chrome browser → selectorshub download for

only use relative xpath → because it is more stable than absolute xpath.
if developer implement some new feature then all hierarchy will be disturbed.
⑧ if developer introduce new element then absolute xpath will be broken.
⑩ if the developer changed the location then absolute xpath will be broken.
but attribute will not changed.

Date: / /

Xpath options →

and ⇒ //input[@id='' or @name='']

or ⇒ //tagname[@attribute='value' and @attribute='value']

contains() → used when elements are dynamic

starts-with()

text()

contains() → ~~attribute~~ (property)

//*[contains(@id,st)]

//*[starts-with(@id,'st')]

//tagname[starts-with(@name,'submit')]

text() → to identify inner text

//a[text()='Women']

Xpath-axes →

self

parent

child

ancestor

descendant

following

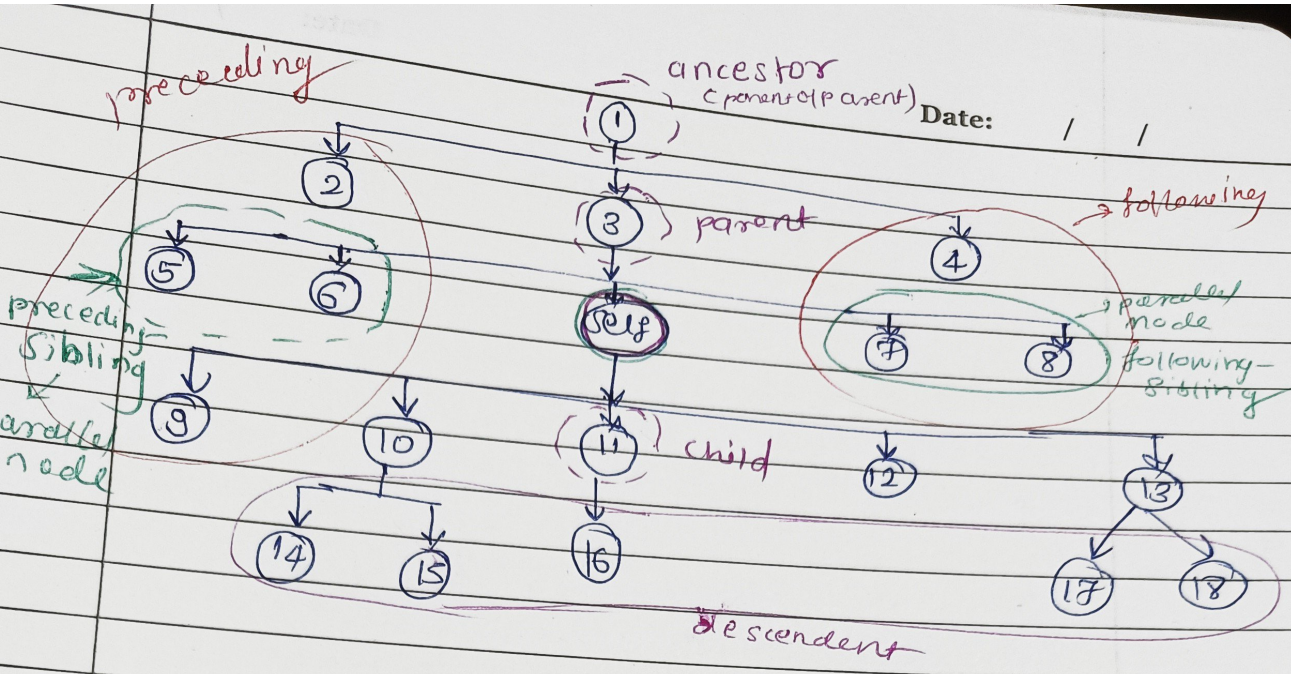
following-sibling

preceding

preceding-sibling

Every element is represent as node.

Any node is considered as self node.



child → traverse all child element of the current html tag → //*[attribute='value']/child::tagname

Parent → Traverse parent element of the current HTML tag.
//*[attribute='value']/parent::tagname

following → Traverse all element that comes after the current tag //*[attribute='value']/following::tagname

Preceding → Traverse all nodes that comes before the current Html tag //*[attribute='value']/preceding::tagname

following-sibling → traverse from current Html tag to Next Sibling Html tag //current html tag[@attribute='value']/following-sibling::Sibling tag[@attribute='value']

Preceding-sibling → traverse from current Html tag to previous Sibling Html tag. //current Html tag[@attribute='value']/preceding-sibling::previous tag[@attribute='value']

Ancestor → traverse all the ancestor elements (grandparent, parent etc.) of the current Html tag.
//*[attribute='value']/ancestor::tagname

descendant → traverse all descendent element (child node, grandchild node etc.) of the current Html tag.
//*[attribute='value']/descendant::tagname.