**BASIC WINDOW COMMANDS**

**GET COMMANDS**

* driver.get("url");
* driver.getTitle();
* driver.getCurrentUrl();
* driver.getPageSource();
* getText();

**NAVIGATE COMMANDS**

* driver.navigate().to("url");
* driver.navigate().back();
* driver.navigate().forward();
* driver.navigate().refresh();

**BROWSER COMMANDS**

* driver.manage().window().maximize();
* driver.manage().window().minimize();
* driver.manage().window().fullscreen();
* driver.close();
* driver.quit();
* driver.manage().window().setSize(obj\_D);

**DROPDOWN HANDLING**

**select - class**

Select s= **new** Select(WebElement);

* s.selectByValue("value");
* s.selectByIndex("index");
* s.selectByVisibleText("text");

**ALERT POP-UP HANDLING**

**Alert - interface**

* driver.switchTo().alert().accept();
* driver.switchTo().alert().dismiss();
* driver.switchTo().alert().getText();
* driver.switchTo().alert().sendKeys("text");

**SCREENSHOT**

**TakesScreenshot - interface**

TakesScreenshot ts = (TakesScreenshot)driver;

File src = ts.getScreenshotAs(OutputType.***FILE***);

File des = **new** File("path");

FileHandler.*copy*(src, des);

**WINDOW / CHILD WINDOW HANDLING**

String parantid =driver.getWindowHandle();

Set<String> childids= driver.getWindowHandles();

driver.switchTo().window(window id);

**I-FRAME HANDLING**

* driver.switchTo().frame("index");
* driver.switchTo().frame("name/id");
* driver.switchTo().frame(WebElement);
* driver.switchTo().defaultContent();
* driver.switchTo().parentFrame();

**JAVA\_SCRIPT\_EXECUTOR**

**JavaScriptExecutor - interface**

JavascriptExecutor js = (JavascriptExecutor)driver;

**scroll page only :**

js.executeScript("window.scrollBy(0,5000)");

**scroll upto the element :**

js.executeScript("arguments[0].scrollIntoView(true)",WebElement);

**SYNCHRONISATION**

**IMPLICIT WAIT**

driver.manage().timeouts().implicitlyWait(Duration.*ofSeconds*(20));

**EXPLICIT WAIT**

**WebDriverWait ,** **ExpectedConditions – class**

WebDriverWait wait = **new** WebDriverWait(driver,Duration.*ofSeconds*(20));

WebElement value = wait.until(ExpectedConditions.*exeption*(By.*xpath*("")));

**FLUENT WAIT**

Wait <WebDriver> FluentWait = **new** FluentWait <WebDriver> (driver)

.withTimeout(Duration.*ofSeconds*(10))

.pollingEvery(Duration.*ofSeconds*(1))

.ignoring(NoSuchElementException.**class**);

FluentWait.until(ExpectedConditions.*elementToBeClickable*(By.*xpath*("")));

**ACTION CLASS**

**Actions – class**

**Action – interface**

Actions act = **new** Actions(driver);

act.doubleClick(WebElement).build().perform(); //Double click

act.doubleClick(WebElement).click().click().click().build().perform(); // Multiple Click

act.contextClick(WebElement).build().perform(); //Right click

act.moveToElement(develops).build().perform(); //MouseHover

act.clickAndHold(elementsrc).moveToElement(elementdes).release().build().perform();//Drag&Drop

act.dragAndDrop(elementsrc,elementdes).build().perform();//DragAndDrop

**EXCEL FILE HANDLING**

File file = **new** File(path);

FileInputStream fis = **new** FileInputStream(file);

XSSFWorkbook wb = **new** XSSFWorkbook(fis);

//Read data operation

XSSFSheet sheet = wb.getSheetAt(0);

XSSFRow row = sheet.getRow(3);

XSSFCell cell = row.getCell(1);

String data =cell.getStringCellValue();

String DATA = wb.getSheetAt(0).getRow(1).getCell(1).getStringCellValue();//Print

//File Write operation

wb.getSheetAt(0).getRow(1).createCell(3).setCellValue("Passed");

FileOutputStream fos = **new** FileOutputStream(file);

wb.write(fos);

String datatest = wb.getSheetAt(0).getRow(1).getCell(3).getStringCellValue();//Print

**TYPES OF EXCEPTIONS**

java.lang.NumberFormatException: For input string: "8,999" becouse of this coma show this

1) org.openqa.selenium.NoSuchElementException

2) Elements not interacted Exception

3) Timeout Exception

4) InvalidSelector Exception

5) ElementsClickIntercepted Exception

6) java.net Socket Exception

7) FileNotFound Exception

8) NoAlertPresent Exception

9) IillegalState Exception

10) NullPointer Exception

11) Class not Found Exception

12)ElementNOtVisible Exception

13)NoSuchFrame Exception

14)NoSuch Window Exception

15)InvalidElement State Exception

16)NoSuchAttribute Exception

17)WebDriver Exception

==============================================================================================

**LINUX COMMANDS**

Cntr + c = to restore $ symbol

Cntr + l = to clear consol

~$ ls = This show all files

~$ ls –a = This show all hidden folder

~$ pwd = its show current directory path

/home/user

~$ cd A = this is used for go into folder A

~$ cd .. = go previews folder

~$ cd = go default directory

~$ cd A/B/C = go into C folder using this path

~$ mkdir A = create New Directory A

~/A/B/C$ mkdir D = create new Directory D

~/A/B/C$ mkdir z x c v b n m = we can create multiple directory

~/A/B/C$ mkdir -v RAN = using –v we can get msg directory is created

mkdir: created directory 'RAN'

~$ mkdir -pv B/C/D/E/F/G/H/I/J/K = -p help us to create perent & child directory

~$ mkdir -pv 4/{1,2,3} = we can create multiple folder inside 4

mkdir: created directory '4/1'

mkdir: created directory '4/2'

mkdir: created directory '4/3'

~$ mkdir -pv x/{A,B}/{D,R}

mkdir: created directory 'x'

mkdir: created directory 'x/A'

mkdir: created directory 'x/A/D'

mkdir: created directory 'x/A/R'

mkdir: created directory 'x/B'

mkdir: created directory 'x/B/D'

mkdir: created directory 'x/B/R'

~$ mkdir -pv C/A/{1,2} {B,C}/3

mkdir: created directory 'C'

mkdir: created directory 'C/A'

mkdir: created directory 'C/A/1'

mkdir: created directory 'C/A/2'

mkdir: created directory 'B/3'

mkdir: created directory 'C/3'

~/C$ mkdir -pv G/A/{Z,Y} G/B/{Z,Y}

mkdir: created directory 'G'

mkdir: created directory 'G/A'

mkdir: created directory 'G/A/Z'

mkdir: created directory 'G/A/Y'

mkdir: created directory 'G/B'

mkdir: created directory 'G/B/Z'

mkdir: created directory 'G/B/Y'

~$ rmdir -pv RAN/A = rmdir means remove directory its removr only empty dir

rmdir: removing directory, 'RAN/A'

rmdir: removing directory, 'RAN'

rmdir: failed to remove directory 'RAN': Directory not empty

~$ rm -rv X = \*rm –r\* is it used to remove empty or non empty directory

removed 'X/2.txt'

removed directory 'X'

~$ rm -v 1.txt = \*rm\* is used to remove file

removed '1.txt'

~$ wc C.txt = \* we filename\*

No line | No Word | No bytes | file name

10 143 985 C.txt

~$ man wc = \*man commandname\* cmd used for show manual related to command

~$ man rm

~$ cat May.txt = \*cat filename\* used to show all content inside file

Unix - It is an operating system.

GUI is not available.

~$ vi B.txt = this cmd used for create new file or edit file

1. vi B.text
2. press ‘i’ to start insert mode
3. modify file
4. press ‘Esc’ for getting out
5. write ‘:wq’ and enter

~$ touch Z.txt = this cmd used for create new file

~$ mv -v Z.txt R.txt = to rename file or Directory name renamed 'Z.txt' -> 'R.txt'

~$ mv B.txt New = to move file \*mv src dsc\* (DON’T USE –v) cannot move directory

~$ cp 2.txt RAN = to copy file into directory

~$ cp -r New RAN = to copy dire to another dire

~$ grep "are" 'text file new.txt' = find word [ word File Name]

~$ grep -n "are" 'text file new.txt' = show lines only with WORD

~$ grep -v "are" 'text file new.txt' = skip line Witch content given WORD

~$ head -8 pipes.txt | tail -3 = its show only center selected lines of data pipe cmd

~$ head -2 'text file new.txt' = used for show data first 10 lines (-2) 2line

wheels, which in turn engaged on an outer toothed ring. The ratios are changed by locking and

unlocking various of the outer rings with brake bands, or sometimes by linking other elements

==============================================================================================

**GITBASH COMMANDS**

Restore Dollar $ = Ctrl+s

Clear console = Ctrl+l

$ git config --global user.name <username>

$ git config --global user.email <email id>

$ git config --global user.name

$ git config --global user.email

$ git config --global --unset user.

$ cd "C:\Users\RANJIT\Desktop\GitBushNew\test"

$ ls

$ touch <file name>

$ git init

$ git status

$ git status -s

$ git add --a

$ git add -A

$ git add .

$ git add <file name>

$ git commit -a -m <"message">

$ git commit -m <"message">

$ git remote add origin <git repo http url>

$ git push -u origin <branch>

$ git clone <git repo http url>

$ git push origin <branch>

$ git pull origin <branch>

$ git branch

$ git branch <branch>

$ git branch <branch>

$ git checkout <branch>

$ git checkout -b <branch>

$ git merge <branch>

$ git checkout <file name>

$ git log

$ git log -p <-count>

$ git diff

$ git diff –staged

$ git checkout –f

$ git rm --cached <file name>

$ git rm <file name>

$ git rm -f <file name>

**API-POSTMAN**

What are the Collection on API?

In the API POSTMAN tool, a collection is a group of requests that are executed together. Collections are a powerful feature of POSTMAN that allow you to save and reuse sets of API requests, making it easier to test and develop APIs.

Collections in POSTMAN can be thought of as a container that holds multiple API requests. You can organize requests within a collection into folders or subfolders, making it easier to navigate and manage large collections.

collections in POSTMAN can also contain scripts, variables, and other resources that can be used across multiple requests. For example, you might define a global variable in a collection that stores an authentication token, which can be used in all requests within the collection.

Overall, collections in POSTMAN provide a powerful way to organize, execute, and share API requests, scripts, and other resources, making it easier to test and develop APIs.

what is API chaining?

API chaining, also known as API chaining or API cascading, is the process of making multiple API requests in sequence, where the output of one API request is used as input for the next API request.

This technique can be useful in situations where you need to perform a series of related tasks or operations that require data from multiple APIs. For example, if you have an e-commerce application that needs to create a new user account and then place an order for that user, you might use API chaining to first create the user account and then use the user ID returned by the first API request as input for the second API request to place the order.

what is querry and path parameter in API?

In API development, query parameters and path parameters are two common ways of passing information to an API endpoint.

Query parameters are used to filter or modify the response of an API endpoint. They are typically added to the end of a URL as key-value pairs, separated by an ampersand (&). For example, in the following URL, "sort" and "order" are query parameters:

https://example.com/api/products?sort=name&order=asc

In this example, the API endpoint returns a list of products sorted by name in ascending order.

Path parameters, on the other hand, are used to specify a specific resource or object in the API endpoint. They are typically part of the URL path, and are identified by a colon (:) followed by the parameter name. For example, in the following URL, ":id" is a path parameter:

<https://example.com/api/products/:id>

https://example.com/api/users/:user\_id/orders/:order\_id

In this example, ":user\_id" and ":order\_id" are path parameters that identify a specific user and order.

Overall, both query parameters and path parameters are important tools in API development for passing information to an API endpoint, and choosing the right parameter type depends on the specific requirements of the API.

can we validate api without cookies

Yes, it is possible to validate an API without using cookies. Cookies are a mechanism to maintain state between the client and the server, but they are not required for API testing.

APIs can be validated using a variety of techniques, such as sending requests with headers, query parameters, or request body. These requests can include authentication tokens, access keys, or other parameters required by the API to perform the desired action.

In addition, APIs can be validated using tools like Postman or other API testing tools, which allow you to send requests and inspect the responses. These tools also allow you to automate API testing and run tests in batches or as part of a continuous integration/continuous delivery (CI/CD) pipeline.

In summary, cookies are not required for API testing, and there are several techniques and tools available for validating APIs.