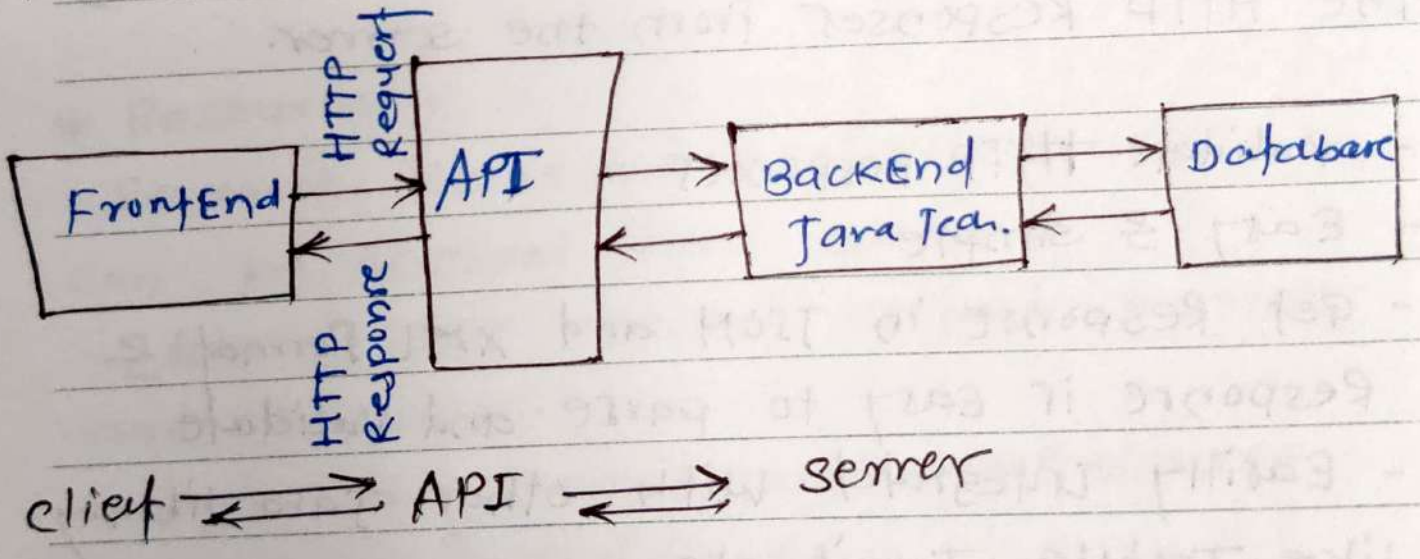
I) * Rest Assured

Rest Assured is a Java based library, that simplifies testing RESTful API. / Webservice.

* API (Application Programming Interface)

- API is a interface between client & server.
- Application programming interface is a way for two or more computer program to communicate with each other.

* Architecture



* Types of API -

i) SOAP - Simple object Access protocol (old)

ii) REST - Representational State Transfer.

→ popularly used REST API

- All Webservicer are API but All ~~Webservicer~~ API are Not ~~API~~. Webservicer.

* API Testing
i) Manually - Postman
ii) Automation - RestAssured.

II) What is RestAssured and Why it's used.

- It is a opensource Jara-library, used to test Restful API. It is capable to Validating the HTTP Responses from the servrer.

- Validate HTTP Responres
- Easy & simple
- Get Response in JSON and XML Format & Responre ir Easy to parse and Validate
- Easily Integrated with other Jara library like TestNG, Junit etc.
- Faster As compare to UI Testing.

* create HTTP Request with All the details.

Example - https://regrer.in/api/users/2

- Base URI - https://regrer.in
- Resources - api/users/
- path parameter - /2
- Query parameter - ?page=2
- Example - https://regrer.in/api/users?page=2

* Base URI
- Base URI is addressed where different resources are located.

* Resources -
Resource Represent API/ collection which can be Accessed from server.
- Every API Request contain BaseURI and Resources.
eg. Google.com/maps → Resource
↑ base URI

* Path Parameter -
- path parameter are used to point specific Resource within a collection such as user identifies by Id.
www.google.com/Images/12 → Path parameter

* Query parameter -

- Query parameter are used to sort or filter the Resourcer.
- Query parameter identified with ?

http://amazon.com/orders?sort-by=123

↑ Bare URL ↑ Rerource ↑ query parameter

* HTTP Methods

The four Basic Create, Read, Update, Delete (CRUD) operation are performed using the port, PUT, GET, DELETE methods in REST interface

• GET -

→ GET is a HTTP Request used to Retrive the information from server.

⇒ If We Wand of data from server, which are already ported.

• post - (port)

→ If We hare to create New data on server then We are uring port Http Requerf.

- used to sent data on serrer.

• PUT -

→ If We hare to update the data, which are already created on server.

• PATCH -

- If We hare to partially update the data which are already created on serrer

• DELETE -

Delete the Data which are already ported or created on server. | delete Resource from serrer.

• Header's -

Additional information that is passed between client and server along with Requerf and Response.

- used for Authentication, catching & messaging body information, handling cookier etc.

- Header's will be in key & value pair

* payload / body :-
— Information that the user wants to sent the server
— Payload only sent with Request (POST, PUT, PATCH) (CREAT & UPDATE)
- RequestBody → sent to server
- Response Body → getting from server.

* Status code - (Response code)
Status code help to understand the status of Response quickly.

100 - 199 - Informational
200 - 299 - Succes (Request succeeded)
300 - 399 - Redirection (Request Redirected to Another URL)
400 - 499 - client Error (client side Error)
500 - 599 - Server Error (Server side Error)

- Rest Assured support BDD Format.
(Given, When, Then statement)

- **Given()** → We provide All input details
→ here, Base URI, Headers, pathParameter Query parameter. and Request Body & payload.

- When() →
→ specify Resource / submit API
eg- HTTP Request method
port, PUT, PATCH, DELETE, GET

- Then() →
→ Validate the Response
(Responsecode, ResponseTime, Response message, Response Headers, Response Body etc.)

* Example:

```
Response res = RestAssured.get("URI");
system.out.println(res.asString());
system.out.println(res.statuscode());
```

↑ Response printing
printing statuscode

__/__/__

res.statusCode(); → print status code
res.time() → print Timing
res.contentType() → print content Type
res.getHeaders("") → get Headers
res.getBody() → get Request Body which we have send.

* example (add BDD format)

```
RestAssured.baseURI ("");
given().log().all();
.body().
.when.port().
.then().statusCode(200);
```

→ above instead of using RestAssured every time we can make it static.

```
import static io.restassured.RestAssured.*;
```

Not Required to use RestAssured.

```
baseURI ("");
given().log().all(); body().
when().port().
then().statusCode(200);
```

Not Required to write RestAssured every time

* JsonObject

- JsonObject class Represent an immutable JSON object value.
- JsonObject is a map like structure that store data as a key & value pairs.

```
JSONObject data = new JSONObject();
data.put("Name","Manish");
data.put("Job","QEA");
```

JSONObject generally (MAP) convert data in JSON format

* Validate Response status :-

→ We sent Request on server, for that Request We get an Response from server.

. Response contains.

① Response status / Response
② Response Header's
③ Response Body

* Validate Response statur / Response
- statur code
- Error statur code
- Response statur Line
↓ statur code / statur string / protocol

* Requertspecification -
- Every Requert in RestAssured library is represented by an interface called Requertspecification.
- Thir interface Allowr to modify Requert like adding header's or authentication details.

eg- ① Validation

```
baseURI = " ";

Requertspecification reqspec = given();

Response rerp = reqspec.get();

ValidatableResponse val = rerp.then();
val.statur code(200);
val.statur Line("HTTP/1.1 200 OK");
```

(II) by BDD.

given().
when().get("URL").
.statuscode(200).
statusLine("HTTP/1.1 200 OK");

* How to check Node size or parsing data.

JsonPath js = new JsonPath(Responsestring);
int id = js.getInt("courses.size()");
string name = js.getstring("courses.name");

① check the size of Array from particular Node
I) courser.size();
↑Node ↳method

② parsing JSONData, from Node
courser.name;
Node ↑subNode

③ parsing Jsondata from NodeArray
"courser[0].title;"
↳ get the title of 0th Array Node

eg.
string title = js.getstring("courrer[0].title");

* How to Return single Header.

```
String contentType = response.header("Content-Type");
String connection = response.getHeader("connection");
```

both methodr Return single Header Value

* How to Return List<Header>

```
Header's headerList = response.getHeaders();
for(Header list : headerList)
  {
  sysO("key" + list.getName() + "value: " +
                list.getValue());
  }
```

Return Headr Name & value in key & value pairr

Node point

```
{ ↓
  "data" : {
     "id" : "2",
    "email" : "abcd@gmail.com",
    "firstName" : "sid",
    "LastName" : "Jadhav"
  }, ↓ Node point
  "courses" : [ ← Hold'r Array
    {
      "title" : "selenium",
      "price" : "50",
    },
    {"title" : "cypress",
     "price" : 40
    }
  ]
}
```

Hold object

Hold object data

Hold object Data

- data Always be in Name & value pair
- Data is separated by comar
- Curly Bracer Hold the object
- ([]) squre Bracket used to hold the Array

* Data Parametrization -

parametrization in TestNG means we want to pass a parameter into test with multiple value. Each value will run on the same test, and output is generated for the final analysis.

Example.

```
                                    dataprovider Name
                                          ↓
@Test (dataProvider = "bookdata")
public void addData (String name, String id)
  {                                  ↑
    -1-          Method must Accept parameter

  }
                                    dataprovider Name
                                          ↓
@DataProvider (name = "bookdata")
 public Object[][] getData()
{
return new Object[][]
      {{"sid", "123"}, {"504", "451"}};
        ↑          ↑
      Name        Id
Run Test upto size of Array
```

* How to Accept Json from File.

→ When We Not modifying data.

① First convert file into Byte
- Byte data to string.

Filer.readAllBytes("file path") → Byte data

paths.get("path")

pass All the data into New String object

eg. → pass below line as payload in Body

new String(Filer.readAllBytes(
paths.get("JSONFilePath")));

* How do add File in Request given c). -|- —

.multipart("File", new File("File Path"))

Using multipart We are able to attach or send image / or file in our Request. (Not Required content type or Body to send above file)

* Authentication in Rest Assured.

- Authentication is a security term which is used providing security to our Application. In RestAssured, authentication is used to perform any Action on API. Without Authentication we are Not able to Access or create data of API.

* Types

i) Basic Authentication :- (Non-Preemptive)

— it is a basic way to Access our API by providing userId and password for a Basic Authentication in (given()) RestAssured

eg

```
given().auth().
    .basic("userName", "password")
    .when().
    .get("    ")
    .then()
```

(by default Non-preemptive)

Rest Assured Not sent credentials to server initially when server explicitly asked for it only then credentials passed to server in header along with rest of the details.

2) Digest Authentication :

→ digest Authentication also work as bast Authentication but more secured than basic Authentication.

- it create hash of the userName & password before sending it to server

example -

```
given().
    auth().
    .digest("username", "password").
    .when()
```

3) Form Authentication.

- verify user's identity

eg. given().

```
auth().
.form("userName", "password")
```

4) OAuth support -

→ ~~outhe~~ OAuth is a mechanism of Authentication by Authentication token to perform any action on API

eg -

given().auth().
.oauth2(accessToken)
.when().

- We can also Authenticate user by passing token or header to perform any Action on API

* Query parameter :-

.QO. Query parameter are used to find or (sort) the filter the Resource.

example.

https://reqres.in/api/users?page=2;

baseURI = "https://reqres.in/";

string response = given.queryParam("page", "2")
.when().get("/api/users").
.then().assertThat().
.statusCode(200).extract().response()
.asString();

II)

```
RequestSpecification reqspec = RestAssured.given();

reqspec.baseURI("  ").basePath("  ")
.queryparam("  ","  ");

Response resp = reqspec.asString();
system.out.println(resp);
```

* preemptive Authentication

```
auth().preemptive().basic("UName", "pass")
```

→ passed the credential at first Request before it asked.

* Barer Token -

→ passed in Header to Accer or do action on Any api for Authentication purpore.

* Serialization and Deserialization of Request/Response with POJO classes.

* Serialization :-
Serialization is a process of converting Java object into Requestbody (payload)

* Deserialization :-
deserialization is a process of converting ResponseBody back to Java object.

* Advantages-
① Easy to parse & extract Response (Json/xml) value if there are wrapped as a Java object.

② Java Objected is constructed with the support of POJO classes.
② POJO classes are created based on request/Response payload.

- We can convert Java object into Request body by POJO - plain old Java object
by HashMap

Required dependencies - GSON (converting data)

* deserialization Demo -

Main POJO class

```
GetCourse gc = given().queryParam("token","...")
  .expect().defaultParser(Parser.JSON)
  .when().get("...").
  .as(GetCourse.class);
```

↑ deserialized our JSON Response into Java object.

- How We can Access

=> gc.getLinkedIn()

=> gc.getCourses().getAPI().get(index)
.getCourseTitle();

* OAuth 2.0

When 3rd party client want to sent request to Access the data then that time We using OAuth 2.0 Authentication

1.0 - old

2.0 - New

→ Need to Authenticate 3rd party Web-site to Access data from API

* RequestSpecBuilder.

— Requestspecbuilder is a class in in Rest-Assured that contains methods for setting various request parameter such as, cookies, header's, multipartdetails, authentication Query parameter etc. It is used to construct Requestspecification for API testing.

eg- If We have common header's, token's and baseURI, We are everytime using this by declaring each request; instead of declaring everytime We are use declared in Requestspebuilder & using this object Reference every time. due to that our code is neat & clean. Requestspec buildor Return the Requestspecification

Interface

```
Requestspecification reqSpec =
        new RequestspecBuilder().
        .setBaseURI(" ").set
        .addHeader(" "," ")
        .ddd Queryparam-(" ")
        .buildd();
```

Response response = given().

common code ⟶ spec(reqspec).
when().post().
then().--;

— due to that reusability of our code increasing with RequestSpecBuilder.

* ResponseSpecBuilder

ResponseSpecBuilder is a class which is used to create ~~Request~~ ResponseSpecification

→ every Response in RestAssured Handler by ResponseSpecification.

ResponseSpecification resspec =
new ResponseSpecBuilder().
.expectStatusCode()
.expectContentType()
.build();

Response response = ~~reqspec.spec~~(
requ → given.spec(reqspec).body().when()
respo → .post().then().spec(resspec).extract()
.asString();

* passing path parameter in dynamically

eg given().log().all().
.spec(DeleteOrder)
.pathParam("key", ProductId).
.when().
.delete("api/order/{productId}")
.then().
.AssertThat().statusCode()
.extract().Response();

dynamic passing

* How to bypass SSL & Http certification

given().relaxedHTTPSValidation()

* BDD (CUCUMBER FRAMEWORK)

* BDD - Behaviour-driven development.

* Gherkin -
- business Readable, Domain specific language that let's you describe software's behaviour
- Gherkin is a language that developer use to define tests in cucumber.

* keyword used in ~~Gherkin~~ Cucumber :-
scenario, Feature, Feature file, scenario Outline, stepDefination.

* scenario :-
- cucumber testcase are Represented as a scenario.
- scenario contains steps equivalent to test steps.

- Given - precondition are mentioned in Given keyword
- When - purpose of when steps to describe the user action
- Then - used to describe an expected outcome or result (validation)
- And - describe positive condition
- But - describe Negative condition.

* Example -

scenario - Make minimum due payment.

~~Given - user fills all details and select minimum amount option~~

Given - User is on pay credid card page

When - User fills are details and select minimum Amout option.

And User clicks on pay putton.

Then credid card confirmation page is displayed.

*And - Addition to previous step & represent positive statement

* But - Statement that are addition to previous steps and represent Negative statements.

* Feature & featurefile.

- Feature represent Busines Requirement.
- Feature file act as a Test suite ~~where~~ which consist All scenario.

In cucumber, Feature file contain scenarios we can simply create featurefile with .feature extension.

- Feature file should contain either scenario, or scenario outline.

* stepDefination -
stepdefination is a Jara class file in which contain all coading part of feature file.

* scenario outline.
- scenario outline is a same as scenario which take dynamic input from table which declared as examples.
- scenario outline Helpr to run our test case multiple time with multiple datasets.
- To Achieve parameterization in our test case, we use scenario outline.
- It is used to run the same scenario multiple time, with different combination of values.

* Junit Runnerfile

↓ Required when Execution done with Junit

```
@RunWith (cucumber.class)
@CucumberOptions (
        features = "Feature file Path",
        glue = {"stepDefination pkg Name"}
    )
public class RunnerFile
{ -1,1-
    -1--
}
```

* TestNG RunnerFile

```
@CucumberOptions (
    features = "Feature File Path",
    glue = {"stepDefination pkg Name"}
)
public class RunnerFile extends
        AbstractTestNGCucumberTests {
-1--
-2--
}
```

* Where you used Inheritance in your project.

→ generally we have some common codes like baseURI, Header's, content type. We declared this code in utils class and extend the stepdefination file with utils call so. stepdefination is child and utils is parent class. To Access All the methods from utils (characteristics/ property) we inherits the stepdefination files with utils class.

* property files

→ Contain common Base URI's

- BDD- Behaviour ~~devel~~ driven development
  Given- When- Then- Approach
- cucumber-
  cucumber is a tool or framework that support behaviour driven development (BDD)

Note-
Background & ~~Example~~ Hooks not used Cucumber same time

* Background :
- Background keyword help to Execute common test steps before every scenario Execution
- Instead of defining common steps in scenario outline / scenario every time we are declaring in Background in one time & this steps run every time before execution of each scenario.
- used to define some precondition for all the scenario.

* Unit Testing -
— indivisual unit is testing
- To do unit testing in Java, we have an excellent framework called Junit.
Junit is a unit Testing framework.

* Difference Between Selenium & Cucumber
- Selenium is a tool that used to design Web Automation test
- Cucumber help to design framwork for selenium Automation tests written in BDD Standard.

Beforesuite - Background - scenario - Aftersuite

* Data Tables-

- Data table are generally used to pass the details
- We can declare Data table for test steps only. & data is available for particular teststeps only.

eg:-

Scenario : Home page default login
Given user is on Netbanking login page
When user signup with following details

| Jenny | abcd | John@gmail.com |

Then Home page is populated.

Data Table

eg :- instepdefination

When ("user sign up with following details")
public void ——— (Datatable data)
{
List<List<String>> obj = data.arList();

obj.get(0).get(0)
index

data.cell(0,0);
row, column

two different way to get data from Datatable

}

* Jsonpath -

Jsonpath is a class in RestAssured, which help to parse the JSON Response in suitable way.

* Annotation (Tags) -

Tags are the Annotation, which help to controll the execution the featurefile or scenario.

eg -

① @RegTest
scenario : login

② @RegTest @SmokeTest
scenario : logout

③ @RegTest @SanityTest
Scenario : Create Account

generally tags help to execution scenario by declaring tags we can manage which scenario we have to Execute or Not

@RunWith (cucumber.class)
@ cucumberOptions(

features = "featurefile path",
glue = "{"stepDefination PkgName"},

tags = " @ smokeTest"
→ Execute only smokeTest tag

tags = " @ smokeTest or @ RegTest"
→ Run both tags simultaniously

tags = " not@sanityTest".
→ except thir one, execute Remaining.

tags = "@RegTest" And @ smokeTest"
→ Execute scenario in which both tags present.

Background & Hookr Not Work together

* Hookr -

- Two Hookr
  After & Before

i) After Hookr -

- Execute test script After Execution of scenario.

eg

ii) Before Hookr -

Execute Hookr Method before Execution of scenario.

Example -

```
public class Hookr {
    ↓ Execute this method before of scenario
@Before (" @MobileTest")   which has
public void beforeValidation()   MobileTest
{  -|-                          Tag
   -|-  -|-  }
@After (" @MobileTest") → Execute this
public void afterValidation()  method After
{  -|-                    of scenario which
   -|-  }                 has MobileTest
}                                tags
```

* Download Reportr

gives Report in pretty format.

↓

plugin = { "pretty",
"html : target / cucumber.html",
"json : target / cucumber.json",
"junit : target / cucumber.xml"
}

↑ Folder Name

↑ file Name which Extension

* ~~monochrom~~
* monochrome : true

for get o/p in conrole with detailed Formaf.

* dryrun = true

- To check test is implemented or Not if missed then get snippets for or body in console.

* Status code-

* Succesful

200 - ok → data fetching

201 - Created (New data created)

202 - Accepted (Request send but in processing)

203 - Non-Authorative -

204 - NO Content → deleted

* Client Error

400 - Bad Request → client Error

401 - Unauthorized → Required Authentication details

402 - Payment Required

403 - Forbidden → Not Accept to perform Action

404 - Not found → data Not found on server

405 - Method Not Allowed → data Not set

* Server Error

500 - Internal Server Error → server Error

501 - Not Implemented

502 - Bad Gateway

503 - Servic unavailable

504 - Gateway Timeout

505 - HTTP verrion Not supported

* Jenkins.

Jenkins is a tool used for Automation
- It is a opensource server that allow All the developer to build, test and deploy software.
by using Jenkins we can make continious integration of project (jobs) or end-to-end endpoint Automation

* CI/CD pipeliner
continious Integration / continious Delivery

* What is continiousIntegration
- Whenever New code commited to New Repository like GitHub. Gitlab, etc. CI will continiously build tested and merged into shared repository
- Benefits-
① Maintain Report of the project
② Bugs can be found quickly

* continious Delivery.

Build → Deploy → Test → Release.

* Jenkins pipeline -

pipeline can Automated the procor like building, testing. and deploying the Application. Doing manully by UI takes lot's of times and effect which will affect on productivity.

* Continiour Integration -

It is the practice of Automating the Integration of code changes from multiple contributer into a single software project, called as CI.

* continiour delivery -

After succertul Build, Test and deployment our software getting released so this procesr is continiously happed after every Integration called as continiour delivery

* Jenkins pipeline -

- Jenkins pipeline is the process Automation of build, deploy, test and Release by in one process by connecting them.

→ contain multiple Job Executing one by one by sequential Manner.

* How to setup Build pipeline in Jenkins

① Chain the jobs (sequentially execution)
(Add upstream / downstream Job)
→

② Install build pipeline plugin
=> Manage Jenkins => plugin
=> Build pipeline plugin → for creating pipeline

=> Dashboard + icon

=> Build pipeline view
→ give Name
→ Select initial Job
* No. of display Build - Time (How many execution.

=> Delivery pipeline

- Build periodically (Jenkins)

* How to configure project from git

- Create New project in Jenkins & give Name

⇒ select source code Management

None — for desktop project

✓ Git → for git Repo projects

→ add credentials to Git.

→

→ select Branch Name

→ * Add Build steps

→ ~~Invoke top level maven targets~~

:. (for Cucumber Required plugin Cucumber-reports)

- top level Maven targets

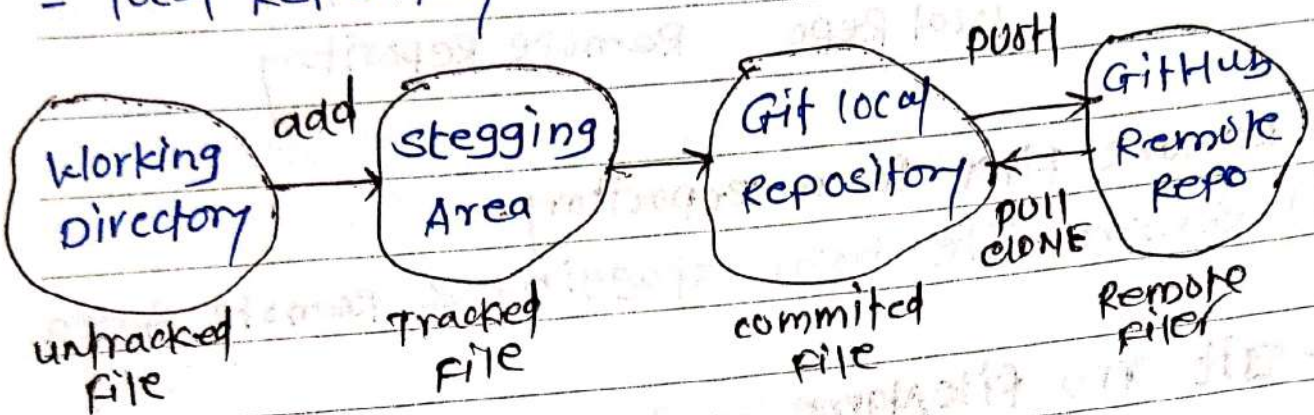
- Add post build Action

→ select Cucumber-reports

→ configure

* Git - Version control system

* GIT - Distributed version control system
- software installed in local system
- local repository



- first do git bash
- git init - initialize git empty repository
- git status - (showing unTracked files)
(Not added in stegging Area)
- git add . * } Add files in stegging Area
- git add -a }
- git status - showing files
- git add a.txt - } Add file in stegging Area
fileName }

- git commit -m 'commit message'
(commit file in local Repository)

- git log - showing All the logs of commits

* See the difference between Local / Remote Repository.

git diff master origin / master

↑ local Repo ↑ Remote Repository ↕ branch

* Remove files from Repository.

(i) Remove file from stagging & Remote Area

- git rm fileName - Remove only one file

- git rm -r - Remove All the files

(ii) Remove files from stagging Area.

- git rm --cached fileName

(iii) delete from Working area Not stegging Area

rm fileName

* Git Reset command.

① Remove changes from stagging Area

② Remove commits from Repository level

① git reset fileName.
- Remove change from stegging Area

② git reset <moder> <commit Id>

Whatever commit Id we mention after that commit Id commit remove

• moder

① reset with mixed mode (defaut mode)
- discard changer from local & stagging area

git reset --mixed commitID

② Reset with --soft option
- same as a mixed moder but changes removed from local Repo and available in Working and stagging Area.

git reset --soft commitID

③ Reset with --hard option
git reset --hard commitId
→ Remove from All places
→ impossible to Revert back changes

* Branching

Master - main branch

① git branch
- give All the brach from raposiory

② git branch branchName
- creating New branch having Name

③ git checkout branchName
switch to Another branch

* → indicate curret branch.

* Merging
— create New branch & make changer in it & then pull this merge thir branch into master. (or pull Request)
— ① switch to master branch

git checkout master → master Branch
git merge feature → child branch

merging feature branch in master branch

* GitHub

— Git is a distributed Version controll system that is used to store the sourcecode in software development to track its changes.

I) * procesr for Adding project in git - (by gitBash)

= First create Repository on Git
(having same name of project Name)

=> Go to project location (Go To Folder)

=> Right click — Git Bash

=> git status → Check statur
git init → ~~Check stat~~ initialize

→ git add . * — Add Everything
git status
→ if filter Not adding then use
git . or git add or git .. *

II) git commit -m "Aug 16 commit" → Any Name

III) git remote add origin —URL

git push -u origin ~~main~~ ~~master~~ main
master

git - installed in local system
git-Hub - Remote Repository

* Git -
- software installed in local Machine

=> How to add in gitHub Repo (PUSH)
- go to Working directory and do git bash

- git init → create Empty Repository in local machine
- git remote add origin URL
→ Attached local & Remote Repo connect / Add
- git status - → Statur of Repository

- git add -A
  git add .*
  git add .
  git add
  } used to Add in this filer in staggin Area.

- git status → get status

* 0

* one timer command before commit

- git config --global user.name "manish phatay"
- git config --global user.email "man@gmail.com"

- git commit -m "First commit"
  Any Name

- git push -u origin master
  → Adding or pushing filer to local git Repo to Remote gitHub.

* How to pull form github Repo - (PULL)

→ git pull origin master & fetch same Add file from github Repo to local git Repository

- Again after modification we can push in github Repository

- git Add -A
- git status
- git commit -m "second commit"
- git push -u origin master

* Clone branch

- git clone 'url' → clone the Remote Repository in local Repository

* Git conflict - (Merge conflict)

let consider two different people working on same ~~go~~ git repo (module) when one person make changer & merge code in Repository then ~~and~~ at the time of 2nd person merging code in git then 1st person done changer Not in 2nd person local repo then that time git conflict occurs, ~~d~~ to Handle this we have to again pull the New changer in our local Repo & then again we have to push our code with New changer in git Repo.

→ We are solving merge conflic manually by pulling all changes in local Repository