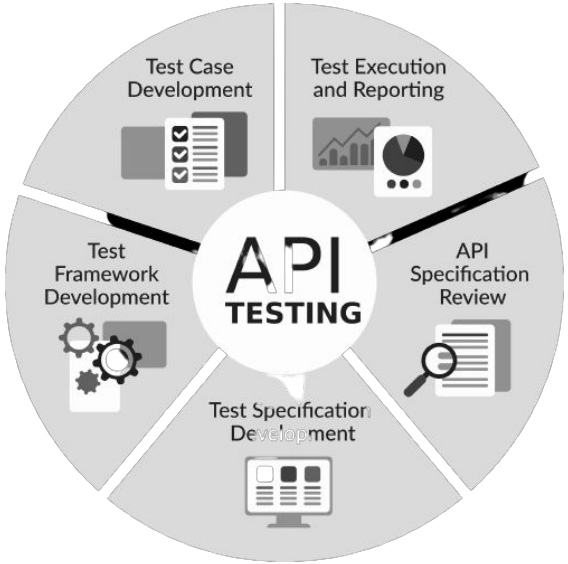
Rest Assured Basics



Pramod Dutta
Lead SDET.

Coding + Rest Assured

Agenda

- Post Request using Rest Assured
- Payload Manage.
- Patch Request Using Rest Assured
- Delete , Put
- One Integration Scenario.

Agenda

Rest Assured Concepts

- Assert Response Time Of Request.
- Payload Manage
  - String
  - Map
  - Object Class
- Payload as Object & Array
  - Class Jackson APIs - Object Mapper
  - Payload as JSON Object
  - Payload as JSON Array.
  - ArrayNode

Agenda

Rest Assured Concepts

- POJO
  - Serialization
  - DeSerialization
- GSON
- Json Include
- Json ignore
- JSON Path
- Compare Two JSON Using Jackson
- Json Assert & Assert J and DDT

Restful Booker Full CRUD

- Post Request using Rest Assured
- Patch Request Using Rest Assured
- Delete , Put
- One Integration Scenario

How to Perform POST Request in Rest Assured?

1. Non BDD Style.
2. BDD Style.
3. With Auth.

Use when.post()

How to Manage the payload in POST/PUT/PATCH

1. Using String
2. Using Map
3. Ultimate moving to POJOs. / Using Class

How to Perform PUT Request in Rest Assured?

1. Non BDD Style.
2. BDD Style.

- Make sure we have a :id available for the PUT Request.
- Full Update with JSON Replace.

Use when.put()

How to Perform Patch Request in Rest Assured?

1. Non BDD Style.
2. BDD Style.

- Make sure we have a :id available for the Patch Request.
- It is a Partial Update.

Use when.put()

How to Perform Delete Request in Rest Assured?

1. Non BDD Style.
2. BDD Style.

- Make sure we have a :id available for the Patch Request.
- It is a Delete the resource.

Use when.delete()

How to Check Response Time in RA?

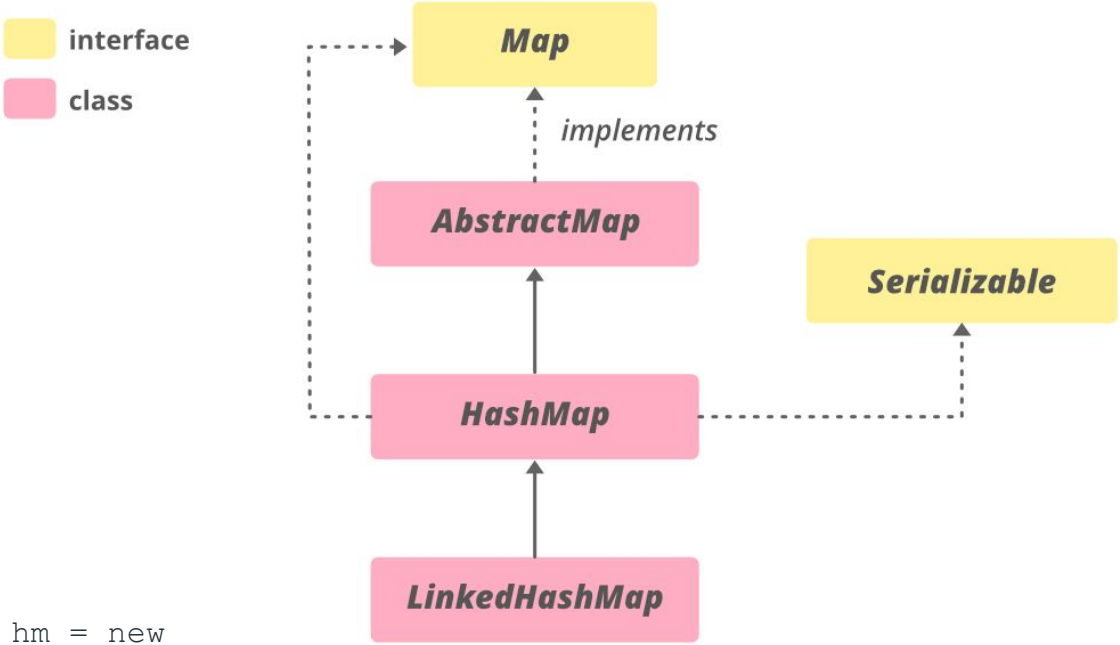
Use request.getTime -

- In Milliseconds it will give.

Hashmap

- HashMap<K, V> is a part of Java's collection since Java 1.2. This class is found in java.util package.
- It stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer).
- HashMap is similar to HashTable, but it is unsynchronized.
- It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values.

Hashmap




```
HashMap<K, V> hm = new
HashMap<K, V>();
```

Manage Payload

- String
- Maps
    - JSON Object
    - JSON Array

GSON

Gson is an open-source Java library to serialize and deserialize Java objects to JSON.

// Map -> InputStream

// JSON -> Object, Object -> JSON

https://mvnrepository.com/artifact/com.google.code.gson/gson

DeSerialization

Serialization

What is POJO ?

- POJO stands for **Plain Old Java Object** and it is an ordinary Java objects not a special kind of.
- The term POJO was coined by Martin Fowler, Rebecca Parsons and Josh MacKenzie in September 2000 when they were talking on many benefits of encoding business logic into regular java objects rather than using Entity Beans or JavaBeans.

POJO : Some rules

- Each variable should be declared as **private** just to restrict direct access.
- Each variable which needs to be accessed outside class may have a **getter or a setter** or both methods. If value of a field is stored after some calculations then we must not have any setter method for that.
- It Should have a default public constructor.
- Can override toString(), hashcode and equals() methods.
- Can contain business logic as required.

POJO : Should not

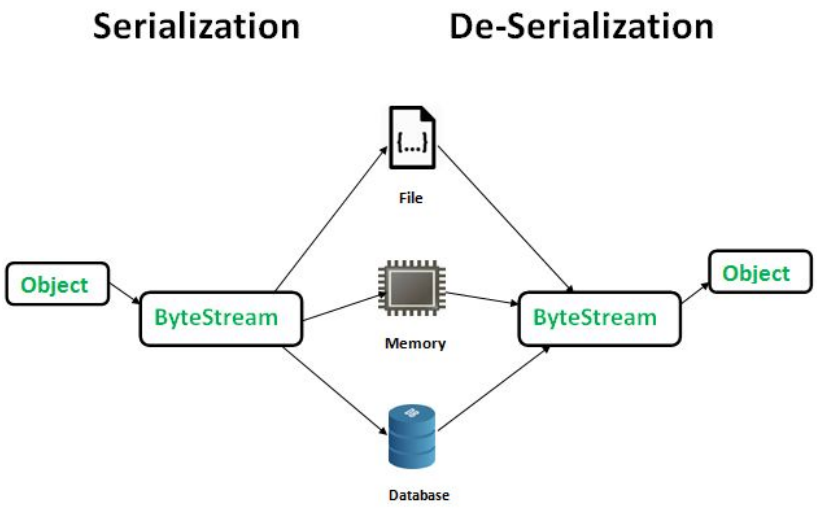
- Extend prespecified classes
- Implement prespecified interfaces
- Contain prespecified annotations

Advantages of POJO :-

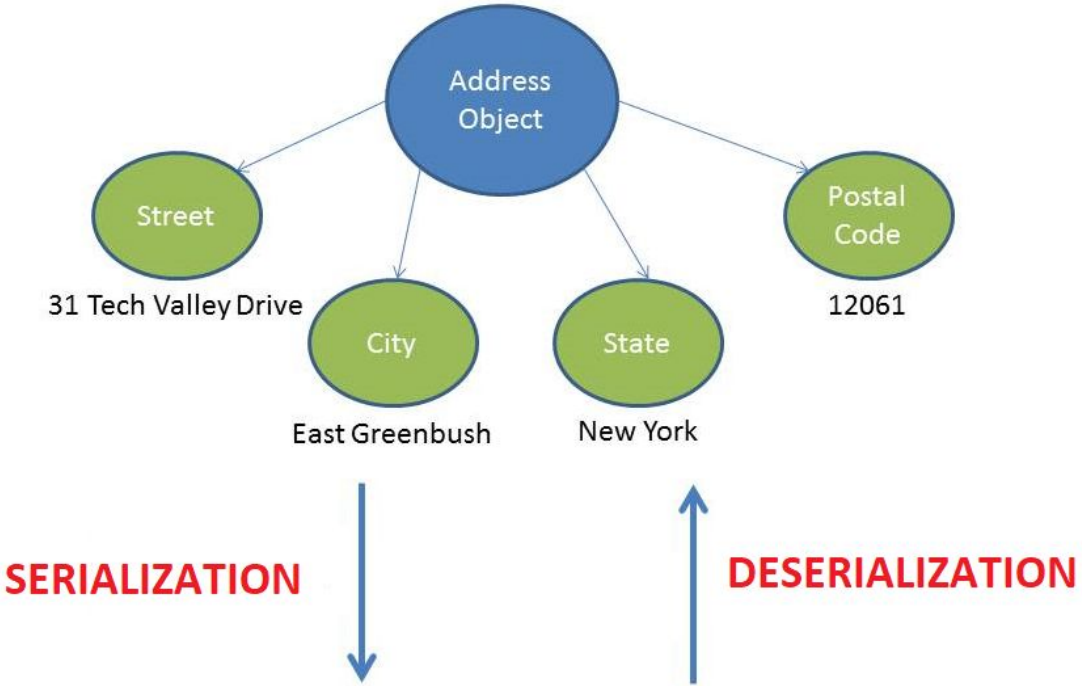
- Increases readability
- Provides type checks
- Can be **serialized** and **deserialized**
- Can be used anywhere with any framework
- Data Manipulation is easier. We can have logic before setting and getting a value.
- Builder pattern can be used with POJO.
- Frequently used to create payloads for API.
- Reusability

Serialization

Serialization is the process of converting an object into a stream of bytes(json) to store the object or transmit it to memory, a database, or a file. I

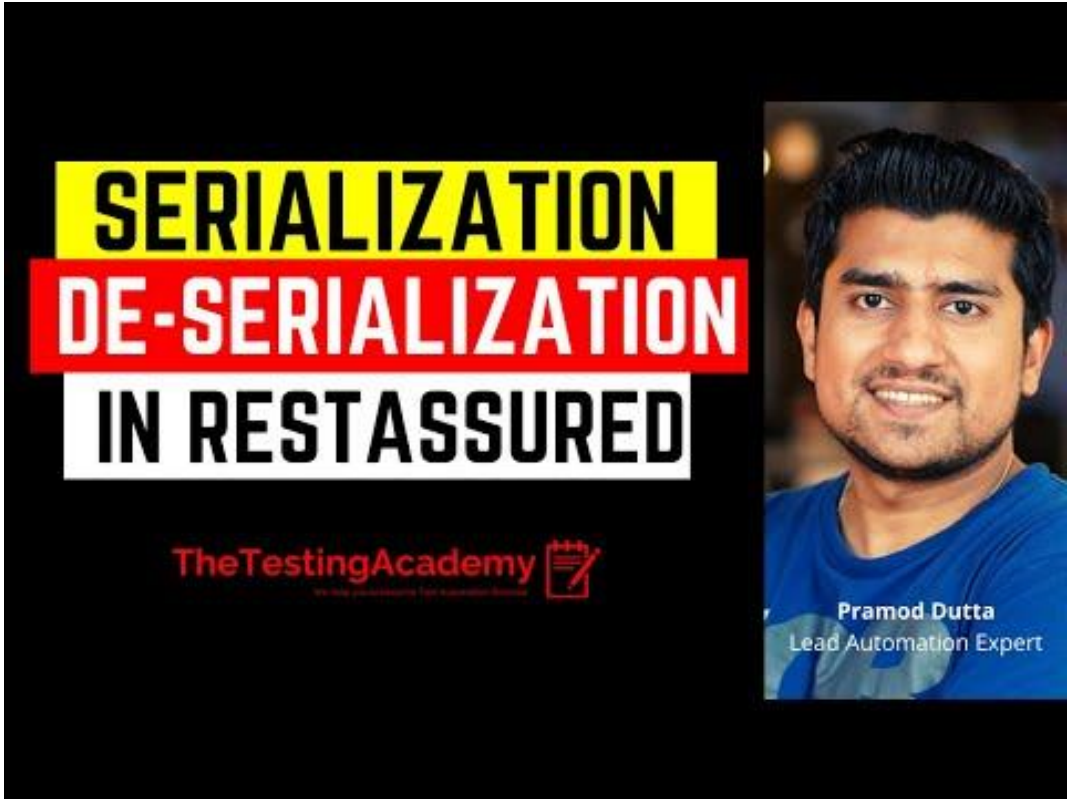


Serialization



<Address><Street>31 Tech Valley Drive</Street><City>East Greenbush</City><State>New York</State><Postal Code>12061</Postal Code></Address>

De-Serialization



https://www.youtube.com/watch?v=muDFyUPOOZo&feature=emb_title

Jackson API

- Jackson API is a high performance JSON processor for Java.
- We can perform serialization, deserialization , reading a JSON file, writing a JSON file and a lot more things using Jackson API.

https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.13.4

ObjectMapper to create a JSON Object or ObjectNode

Manage Payload

Class using Jackson APIs - Payload as JSON Object.

Manage Payload

Class using Jackson APIs - Payload as JSON Array

Manage Payload

Class using Jackson APIs - Payload as ArrayNodes

PassValueBetweenTest

```
2
3   import org.testng.ITestContext;
4   import org.testng.annotations.Test;
5
6   public class passValueBetweenTest {
7
8       @Test
9       public void Method2(ITestContext iTestContext)
10
11          String token = "Hello I am Token";
12          iTestContext.setAttribute("token",token);
13      }
14
15      @Test
16      public void Method2(ITestContext iTestContext){
17          String tokenfromFirstMethso = (String) iTestContext.getAttribute( s: "token");
18
19      }
20
21
22  }
```

JSON Path

JSONPath is **a query language for JSON, similar to XPath for XML**. AlertSite API endpoint monitors let you use JSONPath in assertions to specify the JSON fields that need to be verified.

```
{
  "firstName": "Pramod",
  "lastName": "Dutta"
}
```

JsonPath for node “firstName” – $.firstName

JsonPath for node “lastName” – $.lastName

```
JsonPath jsonPath = JsonPath.from(jsonString);

        // Since firstName holds a string value use getString() method and provide json path of
firstName

        String firstName = jsonPath.getString("firstName");
        String lastName = jsonPath.getString("lastName");
```

https://jsonpath.com/

https://lzone.de/cheat-sheet/JSONPath

https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.html

JSON Path

Operator	Description
`$`	The root element to query. This starts all path expressions.
`@`	The current node being processed by a filter predicate.
`*`	Wildcard. Available anywhere a name or numeric are required.
`..`	Deep scan. Available anywhere a name is required.
`.<name>`	Dot-notated child
`['<name>' (, '<name>')]`	Bracket-notated child or children
`[<number> (, <number>)]`	Array index or indexes
`[start:end]`	Array slice operator
`[?(<expression>)]`	Filter expression. Expression must evaluate to a boolean

https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.html

JSON Ignore

- JsonIgnore is used at field level to mark a property or list of properties to be ignored.

```java
class Student {
   public int id;
   @JsonIgnore
   public String systemId;
   public int rollNo;
   public String name;

   Student(int id, int rollNo, String systemId, String name){
      this.id = id;
      this.systemId = systemId;
      this.rollNo = rollNo;
      this.name = name;
   }
}
```

```java
import java.io.IOException;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.databind.ObjectMapper;

public class JacksonTester {
   public static void main(String args[]){
      ObjectMapper mapper = new ObjectMapper();
      try{
         Student student = new Student(1,11,"1ab","Mark");
         String jsonString = mapper
            .writerWithDefaultPrettyPrinter()
            .writeValueAsString(student);
         System.out.println(jsonString);
      }
      catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

JSON Include & ignore

Output

```
{
    "id" : 1,
    "rollNo" : 11,
    "name" : "Mark"
}
```

More Jackson Annotations

https://www.tutorialspoint.com/jackson_annotations/index.htm

JSON Include

- @JsonInclude is used at exclude properties having null/empty or default values.

```java
public class JacksonTester {
   public static void main(String args[]){
      ObjectMapper mapper = new ObjectMapper();
      try {
         Student student = new Student(1,null);
         String jsonString = mapper
            .writerWithDefaultPrettyPrinter()
            .writeValueAsString(student);
         System.out.println(jsonString);
      }
      catch (IOException e) {
         e.printStackTrace();
      }
   }
}
@JsonInclude(JsonInclude.Include.NON_NULL)
class Student {
   public int id;
   public String name;

   Student(int id,String name){
      this.id = id;
      this.name = name;
   }
}
```

Output

```
{
   "id" : 1
}
```

JSON Assert

- Sometimes we need to compare two JSONs during API testing.
- We need to compare some parts with static data, ignoring the dynamic data like createdDate and Modified Date.
- JSON Assert Lib helps to assert JSON equality effectively.
- JSONassert class provides overloaded assertEquals() method to compare JSON objects and JSON arrays

Compare two JSONs

```
String responseData1 = "{\n" +
        "  \"firstName\" : \"Pramod\",\n" +
        "  \"lastName\": \"Dutta\",\n" +
        "  \"age\": \"30\"\n" +
        "}\n";

String responseData2 = "{ " +
        "\"firstName\" : \"Pramod\", " +
        "\"lastName\": \"Dutta\", " +
        "\"age\": \"30\", " +
        "\"address\" : \"New Delhi\" }";
```

```
JSONAssert.assertEquals(responseData1,responseData2, JSONCompareMode.STRICT);
JSONAssert.assertEquals( message: "LENIENT",responseData1,responseData2, JSONCompareMode.LENIENT);
JSONAssert.assertEquals( message: "NON_EXTENSIBLE",responseData1,responseData2, JSONCompareMode.NON_
JSONAssert.assertEquals( message: "NON_EXTENSIBLE",responseData1,responseData3, JSONCompareMode.NON_
JSONAssert.assertEquals( message: "STRICT_ORDER",responseData1,responseData3, JSONCompareMode.STRICT
JSONAssert.assertEquals( message: "STRICT_ORDER",a1,a2, JSONCompareMode.STRICT_ORDER);
```

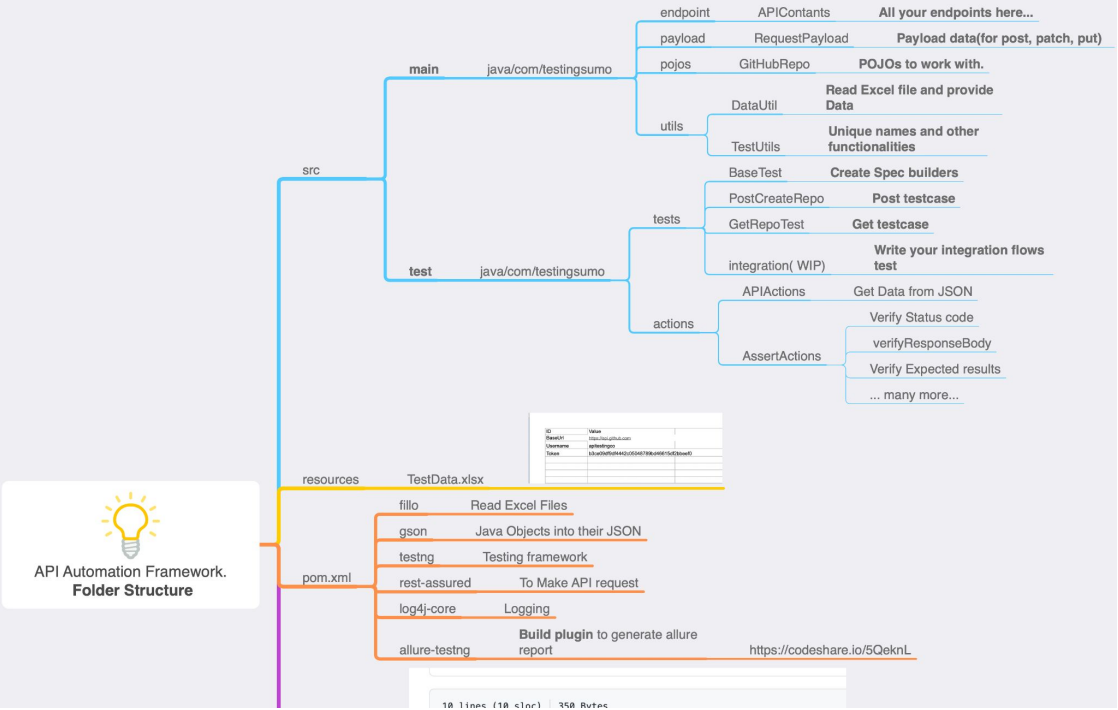
JSON Assert

- JSONCompareMode

	Extensible	Strict Ordering
STRICT	no	yes
LENIENT	yes	no
NON_EXTENSIBLE	no	no
STRICT_ORDER	yes	yes

https://codebeautify.org/jsonviewer/y22993c62

Folder Structure of API Automation



https://sdet.live/rafolder

Demo Project Created!!

- Pom.xml
- Folder structure
- Utils
- Pojo
- Payload management module
- Test cases
- Reporting

https://sdet.live/rafolder

Git Basics

What is Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. So ideally, we can place any file in the computer on version control.

A Version Control System (VCS) allows you to revert files back to a previous state, revert the entire project back to a previous state,

Git is a Distributed Version Control System.

Every user “clones” a copy of a repository (a collection of files) and has the full history of the project on their own hard drive.

```
list1 = [1,2,3]
list1.append(4)
print(list1)
```

#initialFile

```
list1 = [1,2,3]
list1.append(4)
list1.append(5)
print(list1)
```

#addedALine

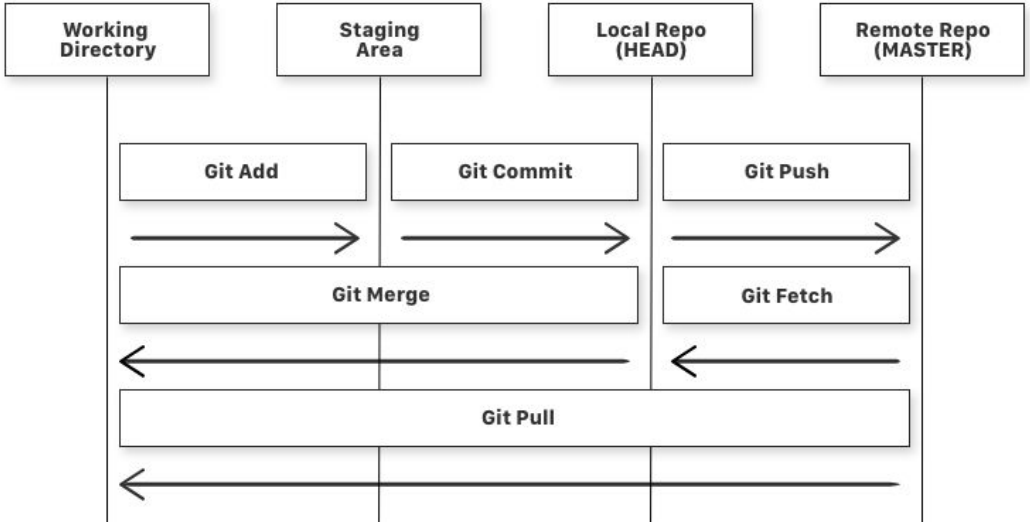
```
list1 = [1,2,3]
list1.pop()
print(list1)
```

#makingChanges

https://sdet.live/rafolder

Git Basics

What is a Repository ?
repo is nothing but a collection of source code.
four fundamental elements in the Git Workflow.
Working Directory, Staging Area, Local Repository and Remote Repository.



https://sdet.live/rafolder

Git Basics

If you consider a file in your Working Directory, it can be in three possible states.

1. **It can be staged.** Which means the files with the updated changes are marked to be committed to the local repository but not yet committed.
2. **It can be modified**. Which means the files with the updated changes are not yet stored in the local repository.
3. **It can be committed**. Which means that the changes you made to your file are safely stored in the local repository.

Git Basics

- `git add` is a command used to add a file that is in the working directory to the staging area.
- `git commit` is a command used to add all files that are staged to the local repository.
- `git push` is a command used to add all committed files in the local repository to the remote repository. So in the remote repository, all files and changes will be visible to anyone with access to the remote repository.
- `git fetch` is a command used to get files from the remote repository to the local repository but not into the working directory.
- `git merge` is a command used to get the files from the local repository into the working directory.
- `git pull` is command used to get files from the remote repository directly into the working directory. It is equivalent to a `git fetch` and a `git merge` .

https://sdet.live/rafolder

Git Practical

```
Step 0: Make a GitHub Account.

Step 1: Make sure you have Git installed on you machine.
If you are on a Mac, fire up the terminal and enter the following
command:

$ git --version

Step 2: Tell Git who you are.
Introduce yourself. Slide in. Seriously, mention your Git username and
email address, since every Git commit will use this information to
identify you as the author.

$ git config --global user.name "YOUR_USERNAME"
$ git config --global user.email "im_satoshi@musk.com"
$ git config --global --list # To check the info you just provided
```

https://sdet.live/rafolder

Git Practical

```
Step 3 - Initialize Git:
And to place it under git, enter:

$ touch README.md    # To create a README file for the repository
$ git init           # Initiates an empty git repository

Step 4 - Add files to the Staging Area for commit:
git add .

Before we commit let’s see what files are staged:
$ git status # Lists all new or modified files to be committed

Step 5 - Commit Changes you made to your Git Repo:
Now to commit files you added to your git repo:

$ git commit -m "First commit"
```

https://sdet.live/rafolder

Git Practical

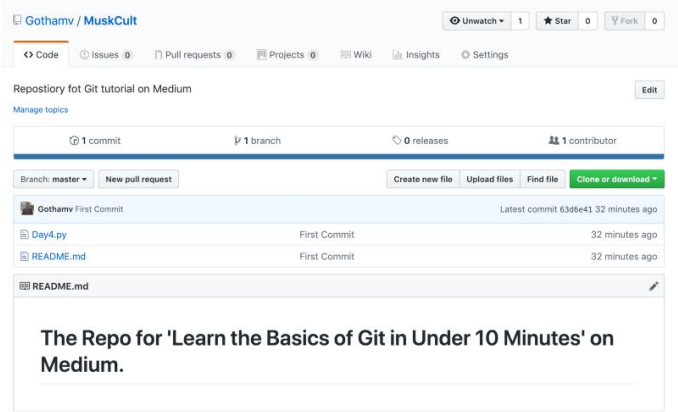
Uncommit Changes you just made to your Git Repo:
Now suppose you just made some error in your code or placed an unwanted file inside the repository, you can unstage the files you just added using:

```
$ git reset HEAD~1
# Remove the most recent commit
# Commit again!
```

Add a remote origin and Push:
Now each time you make changes in your files and save it, it won't be automatically updated on GitHub. All the changes we made in the file are updated in the local repository. Now to update the changes to the master:

```
$ git remote add origin remote_repository_URL
# sets the new remote
```

```
git push -u origin master # pushes changes to origin
```

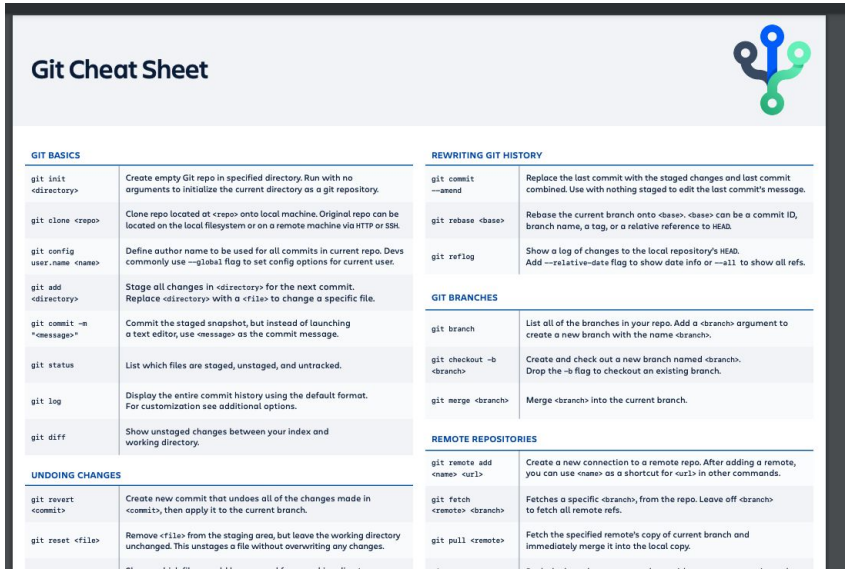


See the Changes you made to your file:
Once you start making changes on your files and you save them, the file won't match the last version that was committed to git. To see the changes you just made:

$ git diff # To show the files changes not yet staged

https://sdet.live/rafolder

Download Git Cheat Sheet



Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

https://sdet.live/rafolder

Thanks, for attending Class

I hope you liked it.

Fin.