**SFDX**

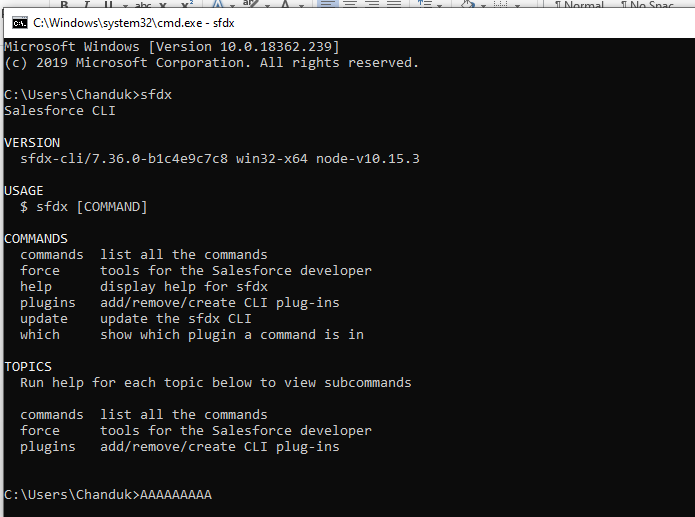
* Download Salesforce command line Interface.

Link: <https://developer.salesforce.com/tools/sfdxcli>

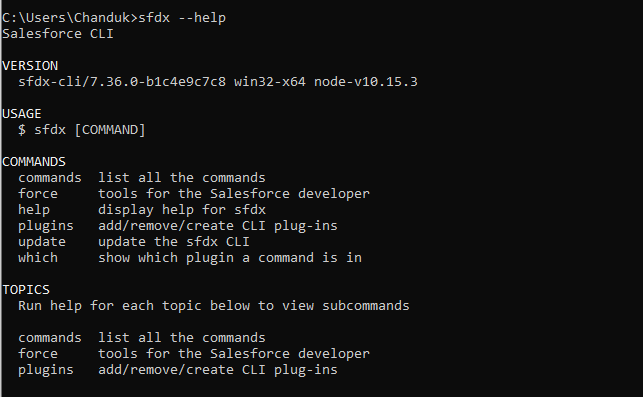
* Next, Download VS Code Editor

Link: <https://code.visualstudio.com/download>

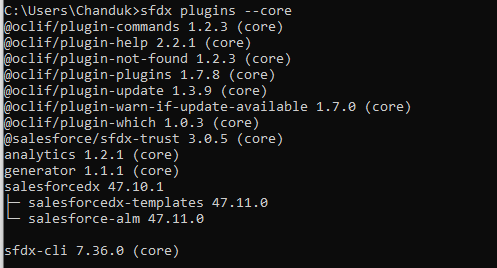
* The Salesforce Developer Experience Command Line Interface (SFDX CLI) is essentially a command line tool that makes it much easier and more pleasant to develop for the Salesforce Platform.
* Open the command prompt (windows) and type cmd then open.
* Then check the sfdx successfully installed or not.
* Then Getting like this



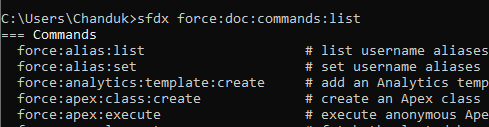
* If you want to help about CLI Commands then type
* sfdx –help ,Then getting like this



* Verify your plugin version,
* sfdx plugins –core,



* Run this command to return a list of the command families in the force topic:
* Command: sfdx force –help
* This command returns all the force commands:
* If You want to display all commands write this command
* Command: sfdx force:doc:commands:list
* Then display the all commands in your command prompt



Help:

Run help for each topic below to view subcommands

Force: alias manage username aliases

Force: apex work with Apex code

Force: auth authorize an org for use with the Salesforce CLI

Force: config configure the Salesforce CLI

Force: data manipulate records in your org

Force: doc display help for force commands

Force: lightning create Aura components and Lightning web components, and test Aura components

Force: limits view your org’s limits

Force: mdapi retrieve and deploy metadata using Metadata API

Force: org manage your orgs

Force: package develop and install packages

Force: package1 develop first-generation managed and unmanaged packages

Force: project set up a Salesforce DX project

Force: schema view standard and custom objects

Force: source sync your project with your orgs

Force: user perform user-related admin tasks

Force: visualforce create and edit Visualforce files

------------------------------------------------------------------------------------------------

Force: alias: list # list username aliases for the Salesforce CLI

Force: alias: set # set username aliases for the Salesforce CLI

Force: apex: class: create # create an Apex class

Force: apex: execute # execute anonymous Apex code

Force: apex: log: get # fetch the last debug log

Force: apex: log: list # list debug logs

Force: apex: log: tail # start debug logging and display logs

Force: apex: test: report # display test results

Force: apex: test: run # invoke Apex tests

Force: apex: trigger: create # create an Apex trigger

Force: auth: device: login # authorize an org using a device code

Force: auth: jwt: grant # authorize an org using the JWT flow

Force: auth: list # list auth connection information

Force: auth: logout # log out from authorized orgs

Force: auth: sfdxurl: store # authorize an org using an SFDX auth URL

Force: auth: web: login # authorize an org using the web login flow

Force: config: get # get config var values for given names

Force: config: list # list config vars for the Salesforce CLI

Force: config: set # set config vars for the Salesforce CLI

Force: data: bulk: delete # bulk delete records from a csv file

Force: data: bulk: status # view the status of a bulk data load job or batch

Force: data: bulk: upsert # bulk upsert records from a CSV file

Force: data: record: create # create a record

Force: data: record: delete # delete a record

Force: data: record: get # view a record

Force: data: record: update # update a record

Force: data: soql: query # execute a SOQL query

Force: data: tree: export # export data from an org into sObject tree format for force: data: tree: import consumption

Force: data: tree: import # import data into an org using SObject Tree Save API

Force: doc: commands: display # display help for force commands

Force: doc: commands: list # list the force commands

Force: lightning: app: create # create a Lightning app

Force: lightning: component: create # create a bundle for an Aura component or a Lightning web component

Force: lightning: event: create # create a Lightning event

Force: lightning: interface: create # create a Lightning interface

Force: lightning: lint # analyze (lint) Aura component code

Force: lightning: test: create # create a Lightning test

Force: lightning: test: install # install Lightning Testing Service unmanaged package in your org

Force: lightning: test: run # invoke Aura component tests

Force: limits: API: display # display current org’s limits

Force: mdapi: convert # convert metadata from the Metadata API format into the source format

Force: mdapi: deploy # deploy metadata to an org using Metadata API

Force: mdapi: deploy: cancel # cancel a metadata deployment

Force: mdapi: deploy: report # check the status of a metadata deployment

Force: mdapi: describe metadata # display the metadata types enabled for your org

Force: mdapi: listmetadata # display properties of metadata components of a specified type

Force: mdapi: retrieve # retrieve metadata from an org using Metadata API

Force:mdapi:retrieve:report # check the status of a metadata retrieval

Force:org:clone # clone a sandbox org

Force:org:create # create a scratch or sandbox org

Force:org:delete # mark a scratch or sandbox org for deletion

Force:org:display # get org description

Force:org:list # list all orgs you’ve created or authenticated to

Force:org:open # open an org in your browser

Force:org:shape:create # create a snapshot of org edition, features, and licenses

Force:org:shape:delete # delete all org shapes for a target org

Force:org:shape:list # list all org shapes you’ve created

Force:org:snapshot:create # snapshot a scratch org

Force:org:snapshot:delete # delete a scratch org snapshot

Force:org:snapshot:get # get details about a scratch org snapshot

Force:org:snapshot:list # list scratch org snapshots

Force:org:status # report sandbox org creation status and authenticate to org

force:package1:version:create # create a first-generation package version in the release org

force:package1:version:create:get # retrieve the status of a package version creation request

force:package1:version:display # display details about a first-generation package version

force:package1:version:list # list package versions for the specified first-generation package or for the org

force:package:create # create a package

force:package:hammertest:list # list the statuses of running and completed ISV Hammer tests

force:package:hammertest:report # display the status or results of a ISV Hammer test

force:package:hammertest:run # run ISV Hammer

force:package:install # install a package in the target org

force:package:install:report # retrieve the status of a package installation request

force:package:installed:list # list the org’s installed packages

force:package:list # list all packages in the Dev Hub org

force:package:uninstall # uninstall a second-generation package from the target org

force:package:uninstall:report # retrieve status of package uninstall request

force:package:update # update package details

force:package:version:create # create a package version

force:package:version:create:list # list package version creation requests

force:package:version:create:report # retrieve details about a package version creation request

force:package:version:list # list all package versions in the Dev Hub org

force:package:version:promote # promote a package version to released

force:package:version:report # retrieve details about a package version in the Dev Hub org

force:package:version:update # update a package version

force:project:create # create a Salesforce DX project

force:project:upgrade # update project config files to the latest format

force:schema:sobject:describe # describe an object

force:schema:sobject:list # list all objects of a specified category

force:source:convert # convert source into Metadata API format

force:source:delete # delete source from your project and from a non-source-tracked org

force:source:deploy # deploy source to an org

force:source:deploy:cancel # cancel a source deployment

force:source:deploy:report # check the status of a metadata deployment

force:source:open # edit a Lightning Page with Lightning App Builder

force:source:pull # pull source from the scratch org to the project

force:source:push # push source to a scratch org from the project

force:source:retrieve # retrieve source from an org

force:source:status # list local changes and/or changes in a scratch org

force:user:create # create a user for a scratch org

force:user:display # displays information about a user of a scratch org

force:user:list # lists all users of a scratch org

force:user:password:generate # generate a password for scratch org users

force:user:permset:assign # assign a permission set to one or more users of an org

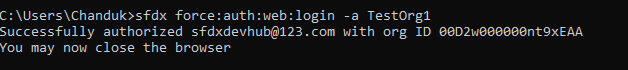
force:visualforce:component:create # create a Visualforce component

force:visualforce:page:create # create a Visualforce page

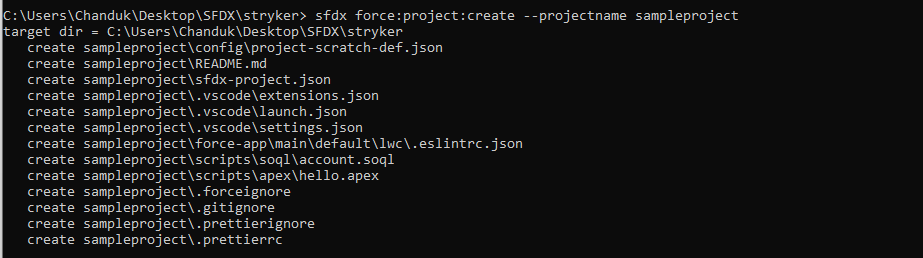
* Open the Devhub org like sandbox or production.
* Command: sfdx force:auth:web:login -r <https://test.salesforce.com>



* And we can directly login our sandbox like this
* Command: sfdx force:auth:web:login -a TestOrg1 🡪(Testorg1 is sandbox user name).
* After Login we are getting like,



* First we should create a project or connect authorized devhub (sandbox/production).
* Command: sfdx force:project:create --projectname sampleproject
* Then Immediately created the Project



* In the project includes the all metadata.
* Set the project path.

**Scratch Org:**

A scratch org is a dedicated short-term Salesforce environment for development and configuration. Scratch org is drive developer productivity and collaboration during the development process, and facilitate continuous integration.

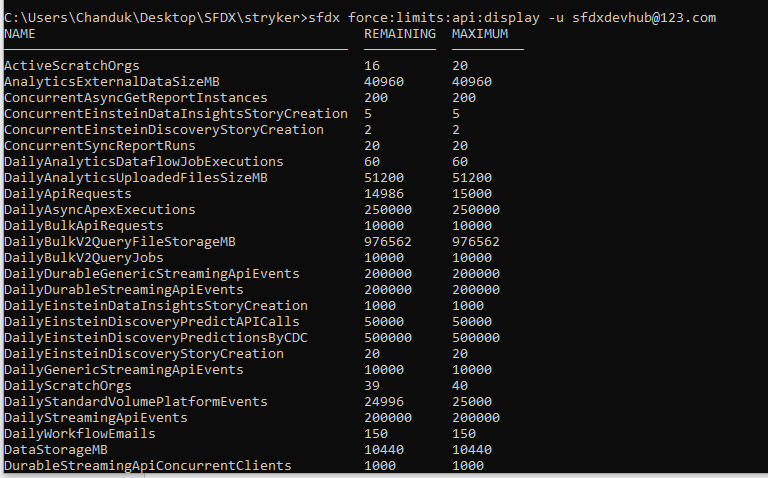
* The Scratch orgs are available in Developer, Enterprise, Group and Professional Editions.
* Scratch orgs drive developer productivity and collaboration during the development process, and facilitate automated testing and continuous integration. You can use the CLI or IDE to open your scratch org in a browser without logging in. You might spin up a new scratch org when you want to:
* To ensure optimal performance, your Dev Hub org edition determines your scratch org allocations. These allocations determine how many scratch orgs you can create daily, and how many can be active at a given point. By default, Salesforce deletes scratch orgs and their associated ActiveScratchOrg records from your Dev Hub org when a scratch org expires. A scratch org expires in 7 days unless you set a duration when you create it.
* Scratch orgs have these storage limits:

200 MB for Data and 50 MB for File.

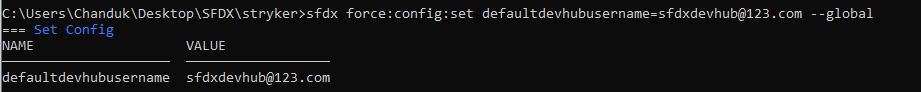
* These Scratch orgs we have some Limitations while creating scratch orgs.

| **Edition** | **Active Scratch Org Allocation** | **Daily Scratch Org Allocation** |
| --- | --- | --- |
| Developer Edition or trial | 3 | 6 |
| Enterprise Edition | 40 | 80 |
| Unlimited Edition | 100 | 200 |
| Performance Edition | 100 | 200 |

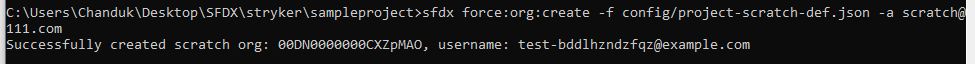
* If you want to view how many scratch orgs you have allocated, and how many you have remaining:
* Command: sfdx force:limits:api:display -u [sfdxdevhub@123.com](mailto:sfdxdevhub@123.com)



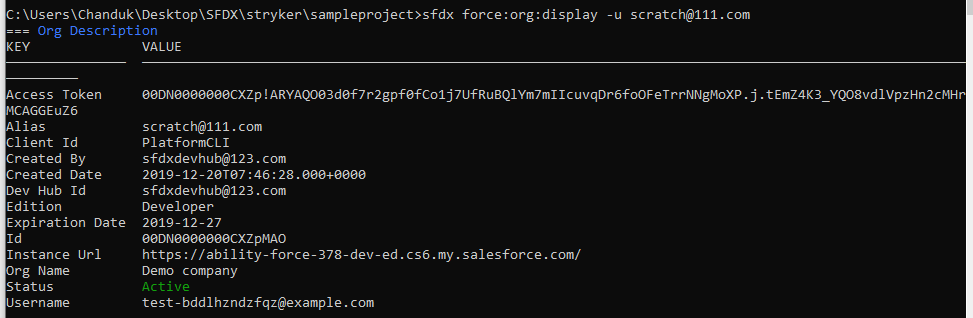
* Before Create the scratch org, a dev hub org username set either with a flag or by default in the config.
* Command:
* sfdx force:config:set defaultdevhubusername=sfdxdevhub@123.com –global



* Then create scratch org
* Command: sfdx force:org:create -f config/project-scratch-def.json -a [scratch@111.com](mailto:scratch@111.com)



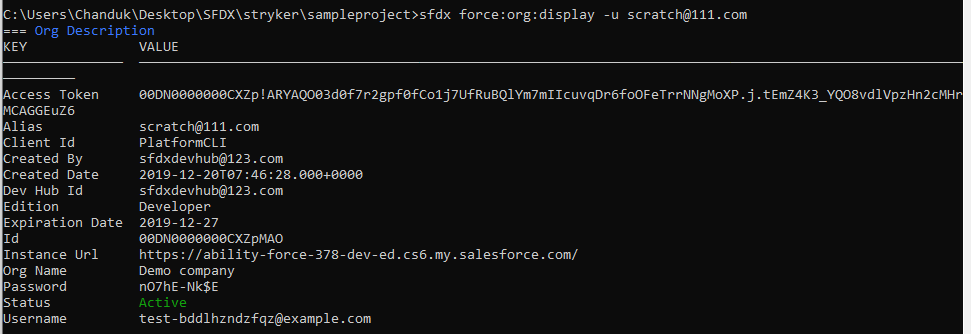
* Display the details of scratch org.
* Command: sfdx force:org:display -u scratch@111.com



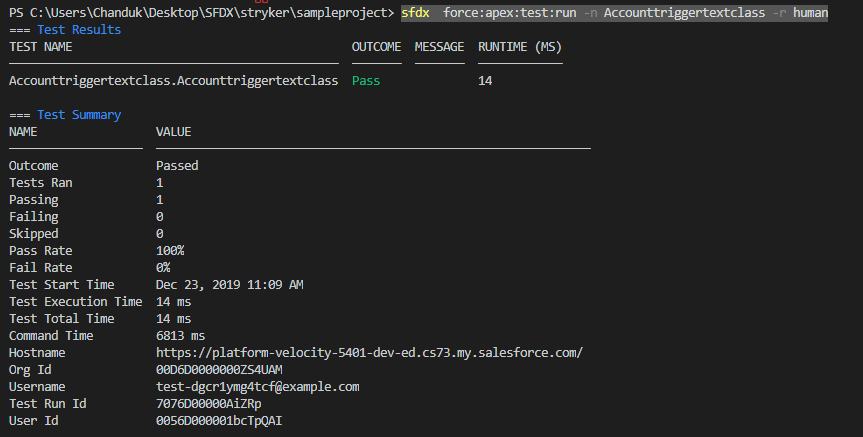
* Then set the password of Scratch org.
* Command: sfdx force:user:password:generate -u [scratch@111.com](mailto:scratch@111.com)



* If we want display the all Details about Scratch org with password
* Command: sfdx force:org:display -u [scratch@111.com](mailto:scratch@111.com)

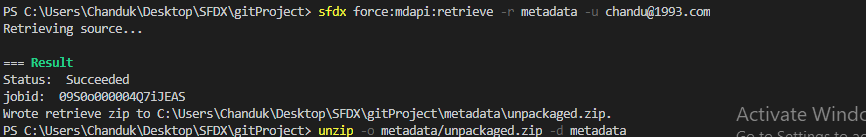


* We should copy and brows the Instance URL Link.
* Then Login Scratch Org with User name and Password.
* If run the test class in command line
* Command: sfdx force:apex:test:run -n Accounttriggertextclass -r human
* Then getting like this

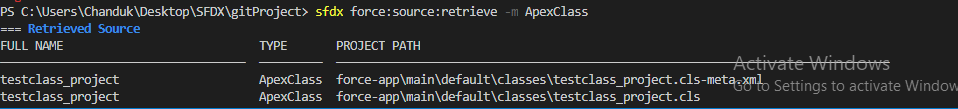


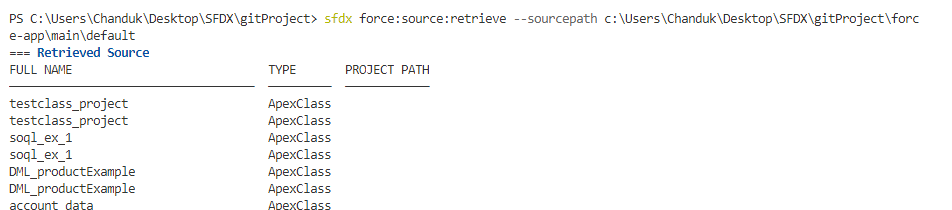
* Retrieve the metadata to DevHub:

Command:sfdx force:mdapi:retrieve –r metadata –u username

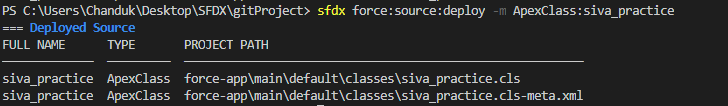


* Retrieve the All Apex classes.





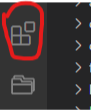
* To retrieve a specific Apex class:
* sfdx force:source:retrieve -m ApexClass:MyApexClass
* To retrieve all custom objects and apex classes
* sfdx force:source:retrieve -m CustomObject,ApexClass
* To retrieve all Apex classes and two specific profiles (one of which has a space in its name).
* sfdx force:source:retrieve -m "ApexClass, Profile:My Profile, Profile: AnotherProfile"
* If you want to deploy all apex classes
* Command :sfdx force:source:deploy -m ApexClass.
* If you want to deploy to particular Apexclass
* Command: sfdx force:source:deploy -m ApexClass:MyApexClass



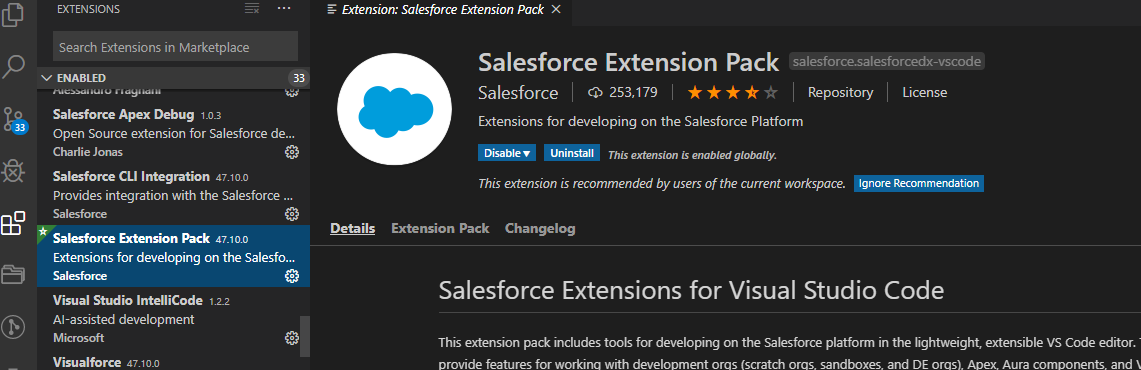
* To deploy all custom objects and Apex classes:
* Command: sfdx force:source:deploy -m CustomObject,ApexClass
* To deploy all Apex classes and two specific profiles (one of which has a space in its name).
* Command:sfdx force:source:deploy -m "ApexClass, Profile:My Profile, Profile: AnotherProfile"
* To deploy all components listed in a manifest.
* sfdx force:source:deploy -x path/to/package.xml
* To run the tests that aren’t in any managed packages as part of a deployment:
* sfdx force:source:deploy -m ApexClass -l RunLocalTests
* To check whether a deployment would succeed (to prepare for Quick Deploy):
* sfdx force:source:deploy -m ApexClass -l RunAllTestsInOrg –c
* To deploy an already validated deployment (Quick Deploy):
* sfdx force:source:deploy -q 0Af9A00000FTM6pSAH
* If you want to cancel the deploy
* force:source:deploy:cancel
* check the status of a metadata deployment
* force:source:deploy:report

**VS Code Editor**

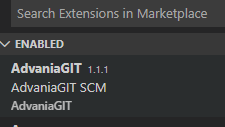
* Install the VS Code editor.
* Open The VS Code Editor and open Market place

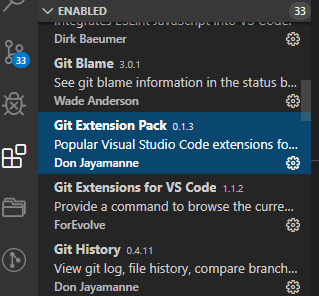


* Type Salesforce extension pack.

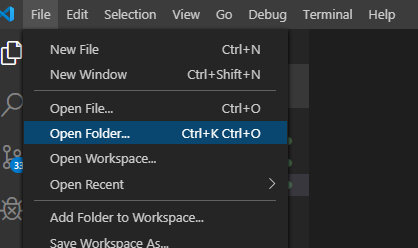


* Install that pack
* And git pack and AdvaniaGIT.





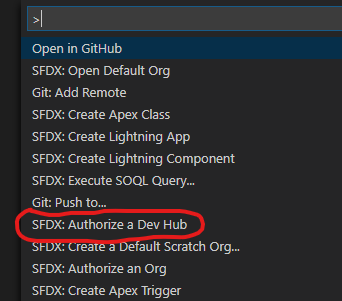
* Click the File Open folder and open project.



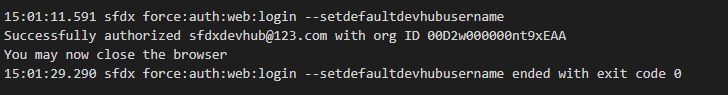
* Then we getting all meta data files like apex Class,Aura,LWC,Git,Objects,Flexipages,Pagelayouts,Permissionsets,Triggers,Tabs,

Applications.

* In VS Code editor we can develop the code and push to scratch org.
* First we have to login devhub (Sandbox).
* Open the command palette(Ctrl+shift+L) Type Authorized Dev Hub.



* Then immediately open the login page and after login getting like this.



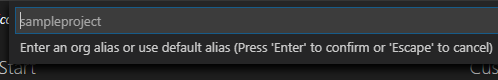
* In the VS Code editor we can create the Scratch org.
* Open the command palette and type create a default scratch org.



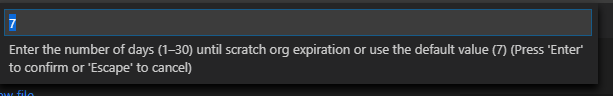
* Select this



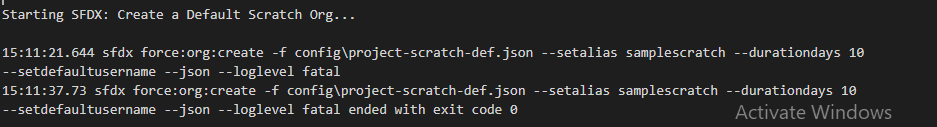
* Enter The Scratch org alias name and press Enter.



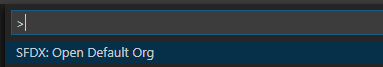
* Then Enter the Scratch org validity and Press Enter
* Scratch org min 7 days and Max 30 validity.



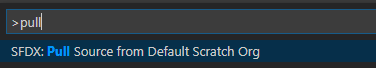
* After creating the scratch org getting like this in output panel.



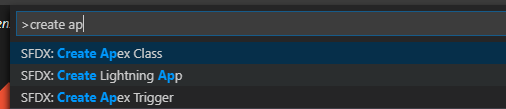
* Then open the scratch org.
* Open command palette and enter open default org.
* Then immediately open the scratch org.



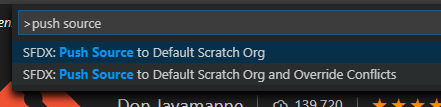
* Then pull source from scratch org.
* Open command palette and type pull source from default scratch org



* Then pulled, whatever the metadata we have in scratch org.
* If whatever the changes is happened in project
* Suppose we create one apex class. Or trigger or lightning application

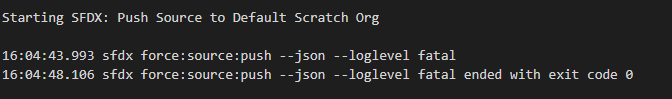


* That code should push to scratch org

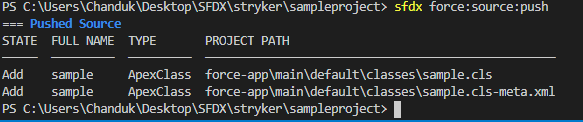


* Then that code pushed to scratch org.

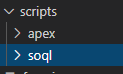
Then getting output like this



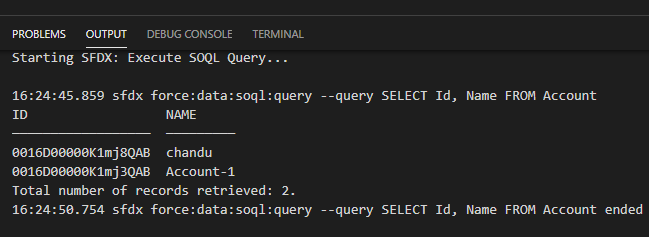
* In the command terminal also same
* Command: sfdx force:source:push



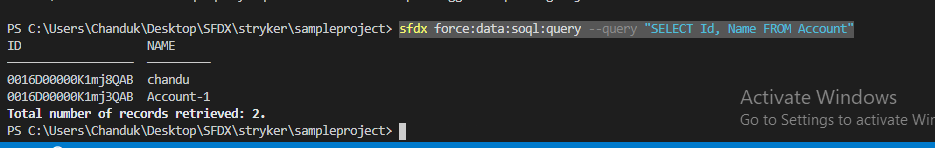
* If you want run the soql query in VS code editor open the soql folder.
* In script folder we have soql folder then open



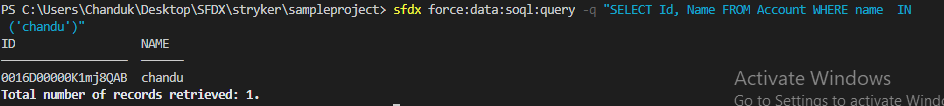
* Write the query and select the query and open the command palette and type Execute SOQL query with currently selected text.
* And select the REST Api execute



* In command terminal also
* Command: sfdx force:data:soql:query --query "SELECT Id, Name FROM Account"

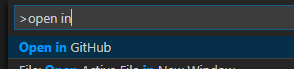


* If write the soql using WHERE clause also like this



**Git Hub:**

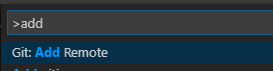
* In command palette type open github.

****

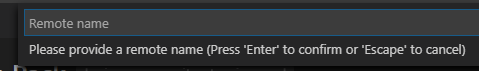
* Then immediately open the github login page.
* Login and open the Git.
* Create the repository.
* Then open command palette and type git clone

****

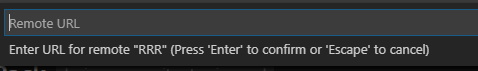
* Git: clone is clone all git access and repository’s.
* Then we add the one repository to this project .why because entire project we should store the metadata and code.
* Then add the remote of the repository.
* Open the command palette and type add remote.



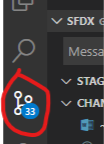
* Then enter the remote name and press enter. Whatever name we may give.



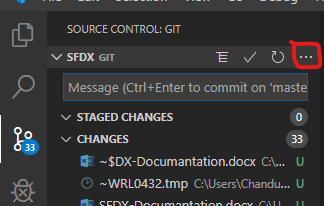
* And enter the Repository URL



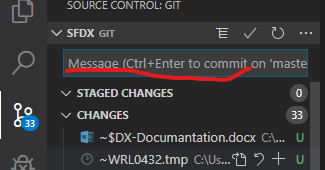
* Then open git icon



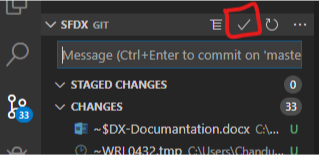
* And click this and connected to your project to Git Hub.



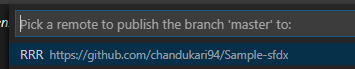
* Whatever the changes is happened in project then display the notification on git icon.
* Then push the code to git hub first have to commit the changes.



* Here type enter the commit message.
* Then click the tick icon and then automatically save the changes.



* Open the command palette and type GIT: push to
* Select the remote of repository.



* Then Save the all code and metadata in repository.

