**MOBILE COMPUTING PROGRESS FILE**



**Rabia Arif**

**Roll no: BSEF18A039**

**Version Control System**

**Version control systems** are software tools that help software teams manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

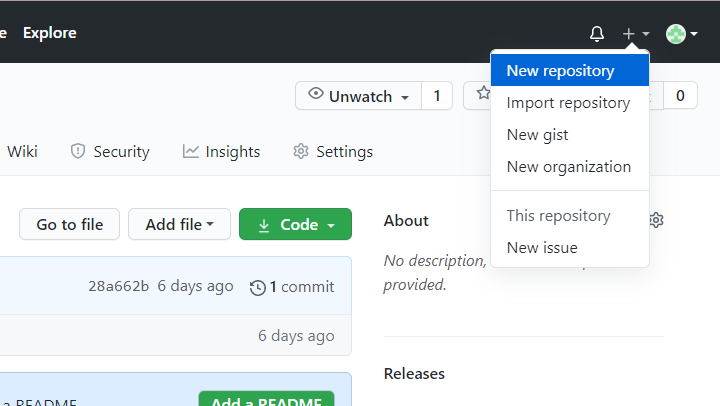
**Git** is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Step-0:

* 1. Sign up to git
  2. Install git

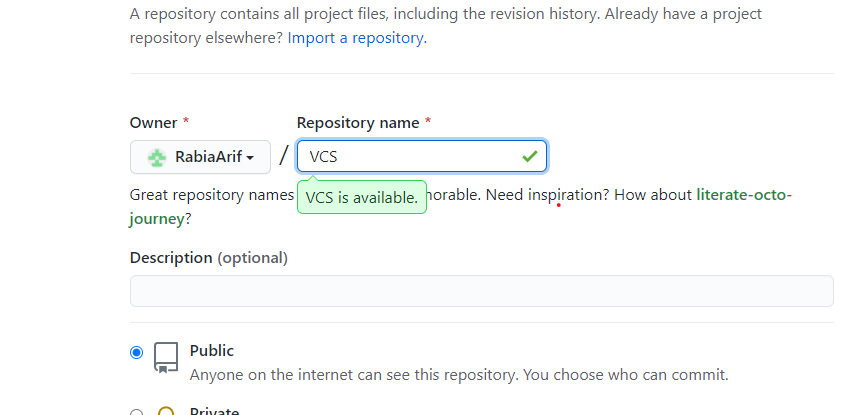
Step-1:

**Create new repository**

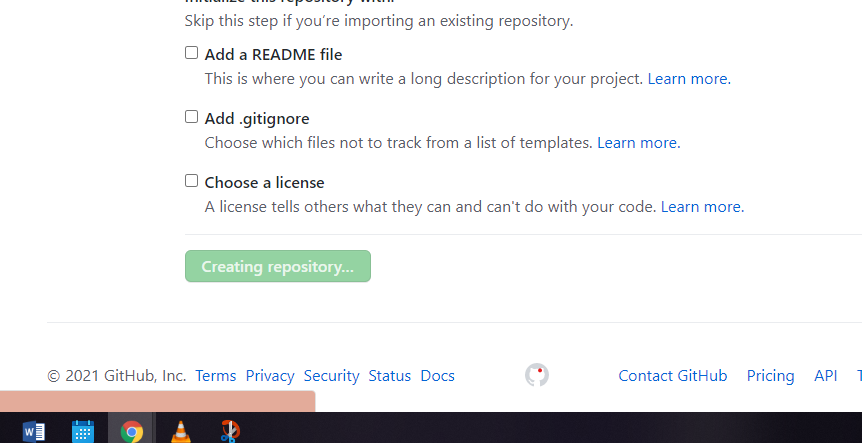


Step-2:

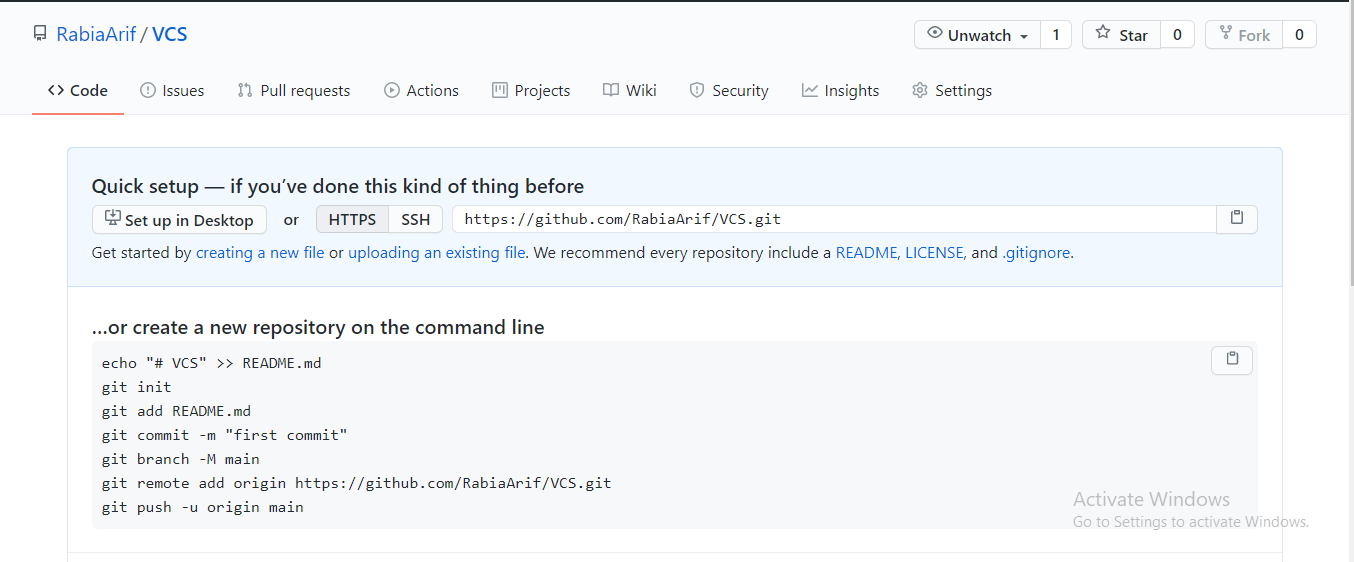
**Set Repository name**



Creating…



Details:

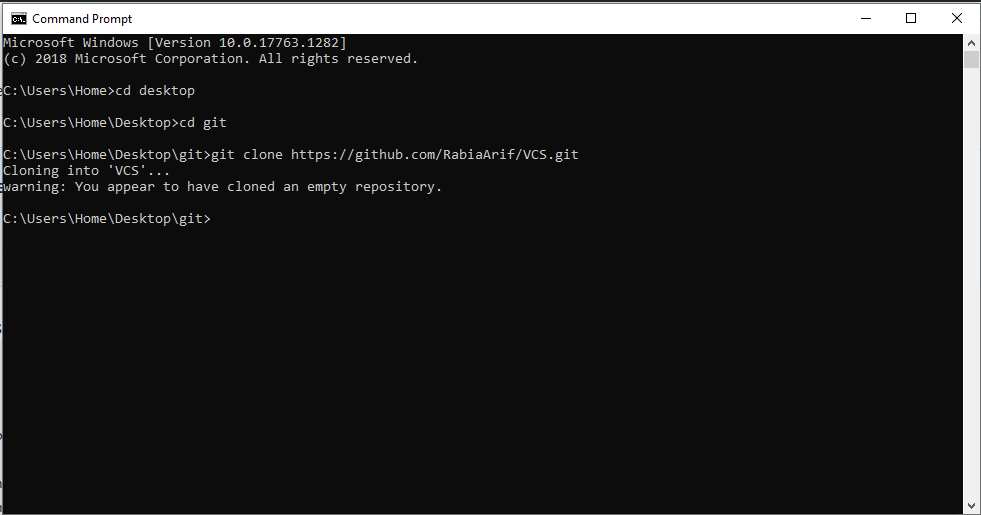


Copy URL

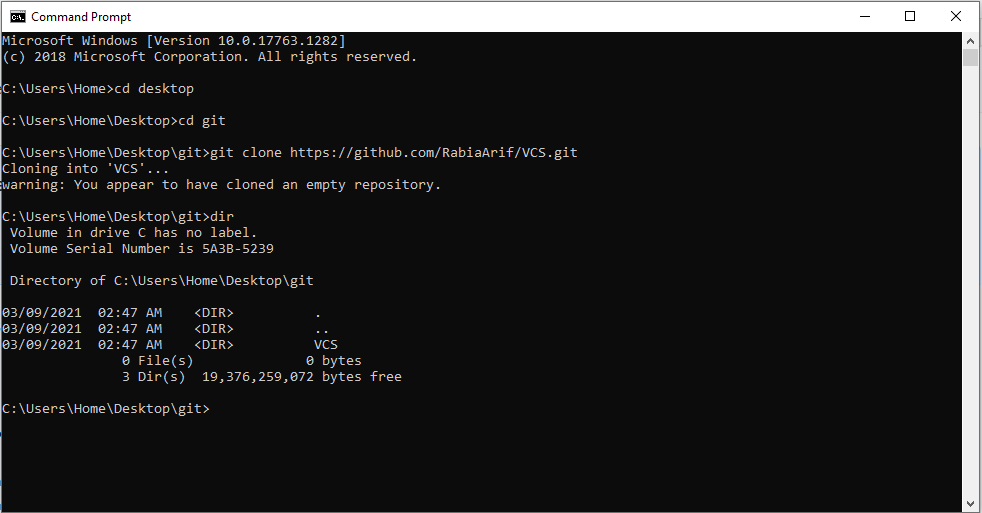
Step-3:

**Cloning locally**

Using command: **git clone** {URL}

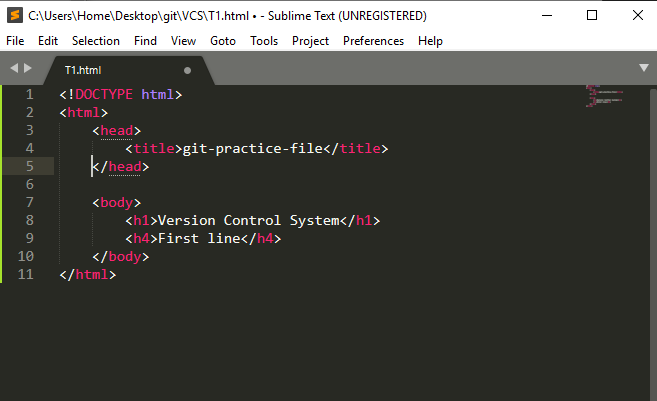


Repository cloned locally



Step-4:

**Creating file in local repository** (C:\...\git\VCS)

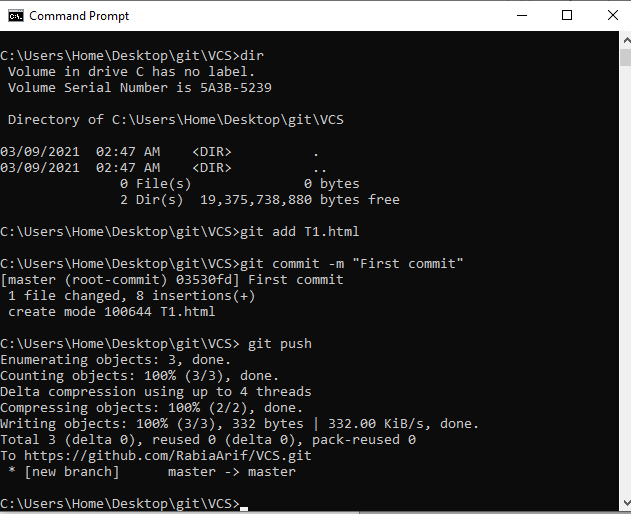


Step-5:

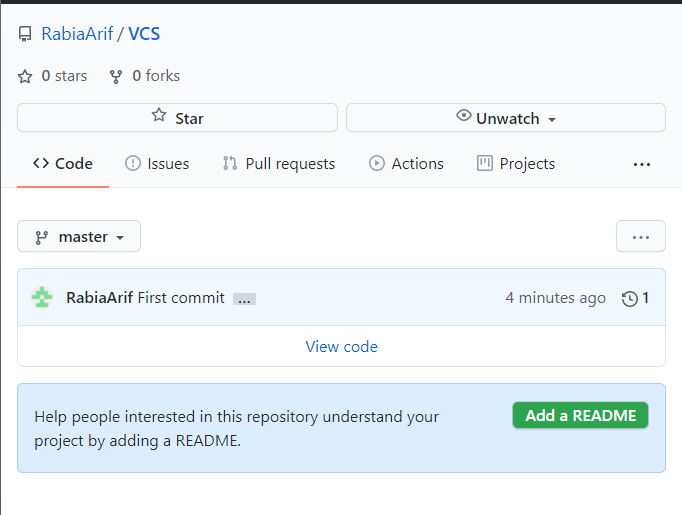
**Add locally created file to online repository**

Using commands

1. **git add** {fileName}
2. **git commit –m** {“message”}
3. **git push**

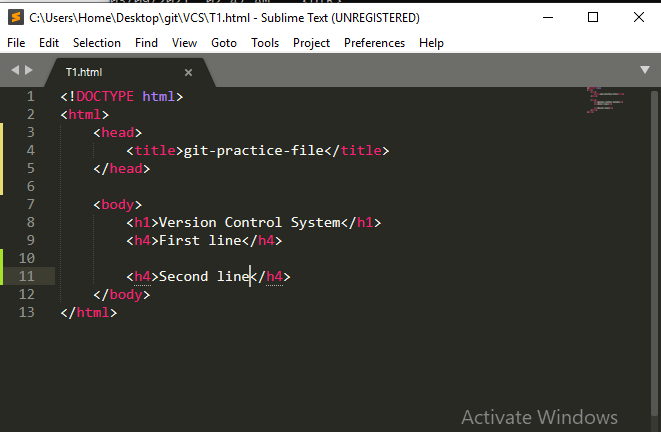


Now data in repository and data available locally is **synchronized**.



Step-6:

**Adding new content in local file**

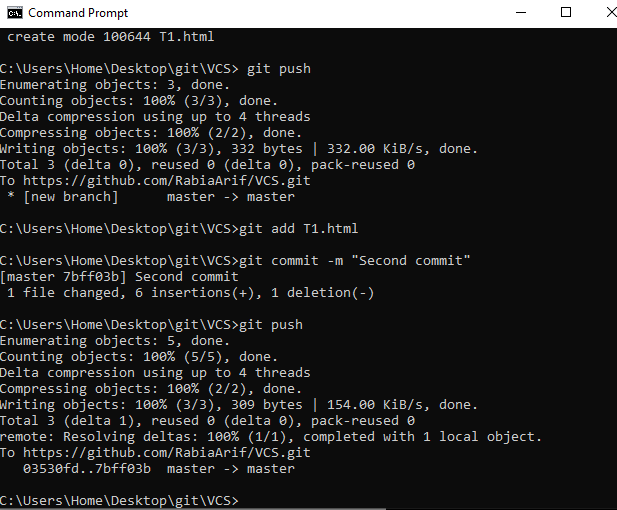


Step-7:

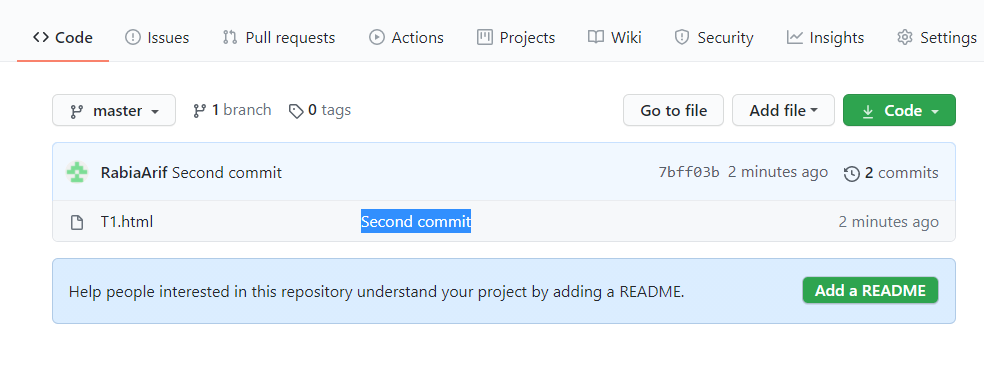
**Push addition in repository**

Using above 3 command

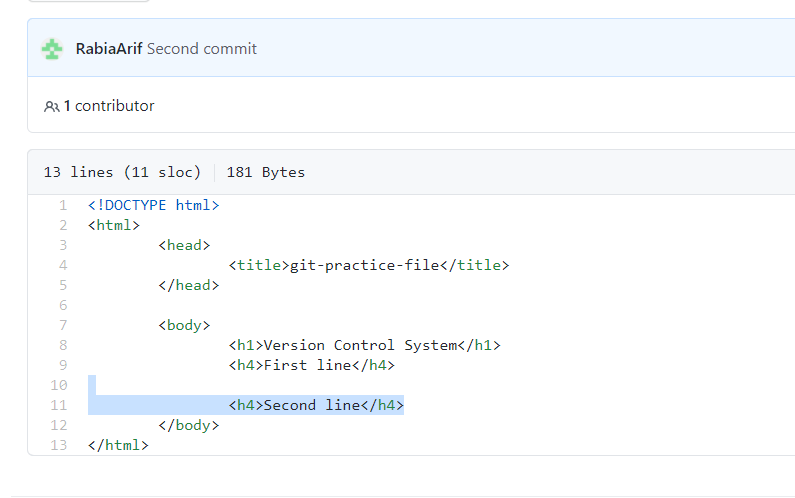
1. **git add** {fileName}
2. **git commit –m** {“message”}
3. **git push**



Change visible in online repository

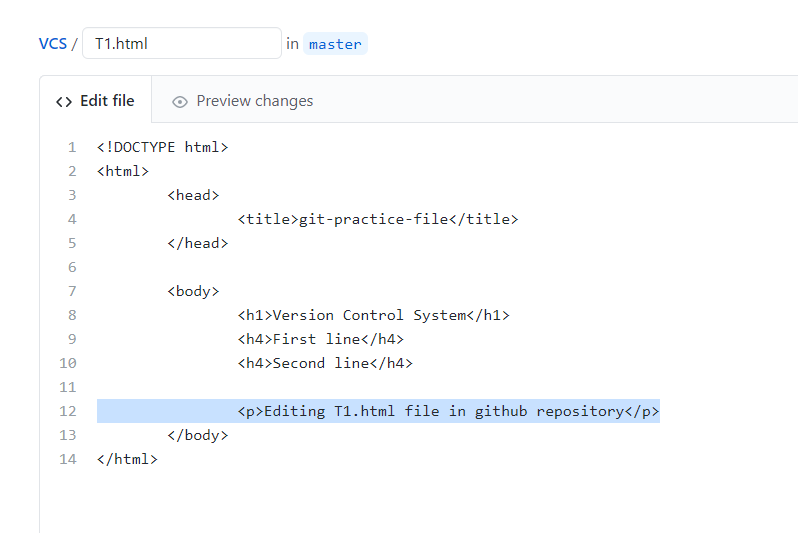


Data in online and local repository … Synchronized

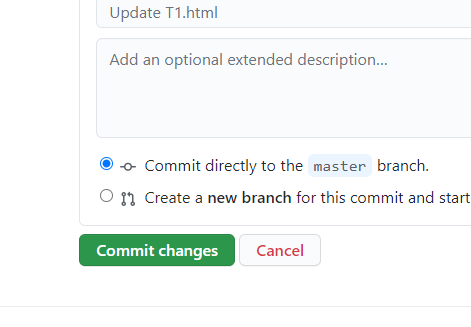


Step-8:

**Online editing a file (in github repository)**



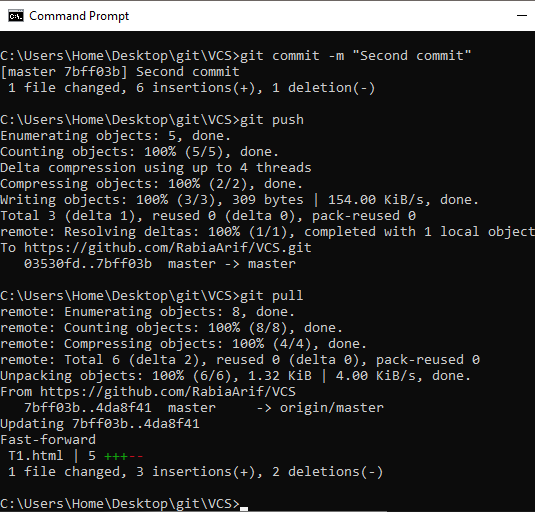
Commit changes



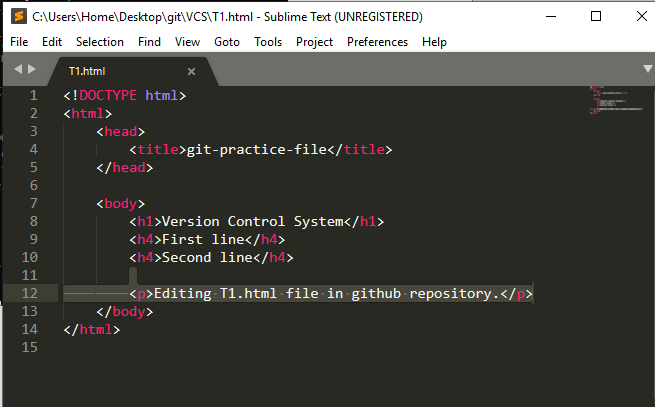
Step-9:

**Getting changes locally**

Using command **git pull**



Synchronized, change also visible in local file



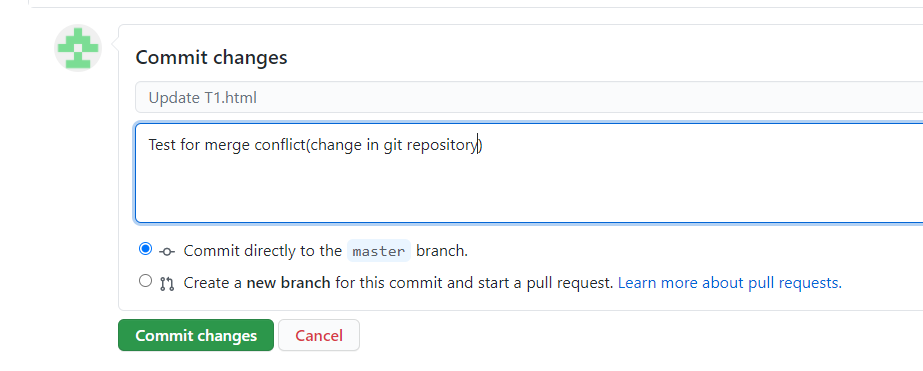
**Merge Conflict**

Version control systems are all about managing contributions between multiple distributed authors (usually developers). Sometimes multiple developers may try to edit the same content. If developer **A** tries to edit code that developer **B** is editing, a conflict may occur. To alleviate the occurrence of conflicts, developers will work in separate isolated branches. The Git merge command’s primary responsibility is to combine separate branches and resolve any conflicting edits.

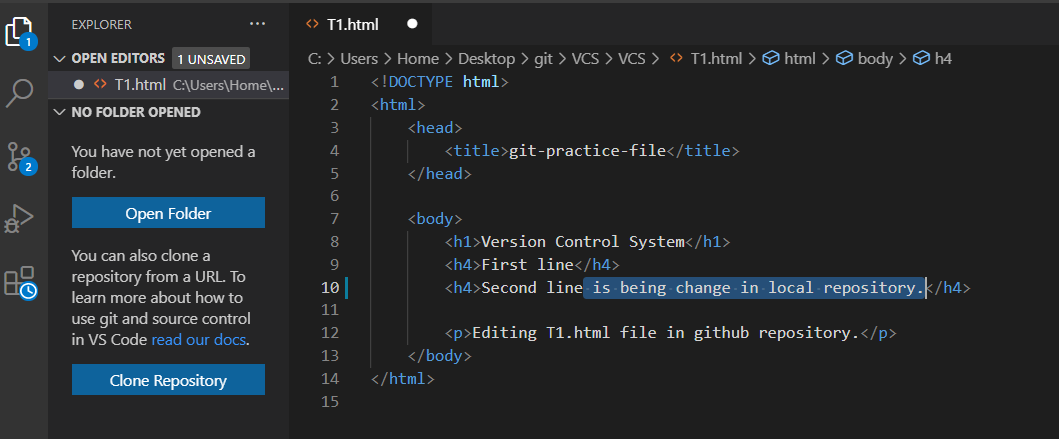
**Merge conflicts** can happen when merging a branch, rebasing a branch, or cherry picking a commit. If Git detects a conflict, it will highlight the conflicted area and ask which code you wish to keep. Once you tell Git which code you want, you can save the file and proceed with the merge, rebase, or cherry pick.

Changing same line of code in git central repository and local repository can case merge conflict.

* **Changing same line of code in git repository**



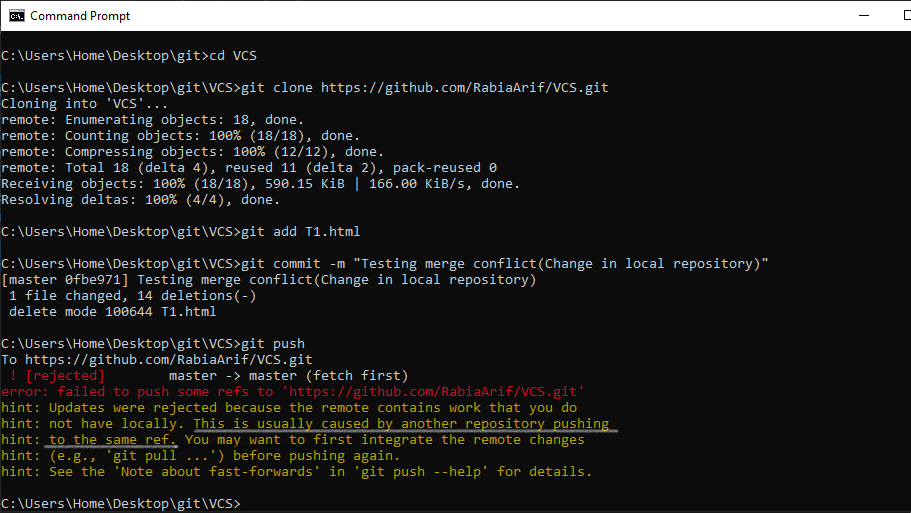
* **Changing same line of code in local repository**



* Using above 3 command

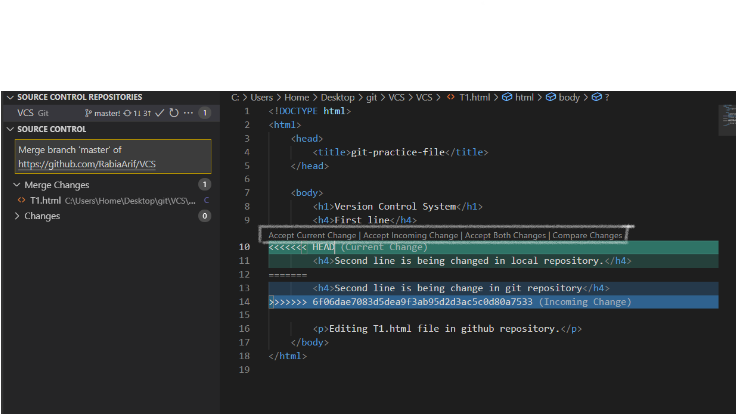
1. **git add** {fileName}
2. **git commit –m** {“message”}
3. **git push**

to save changes



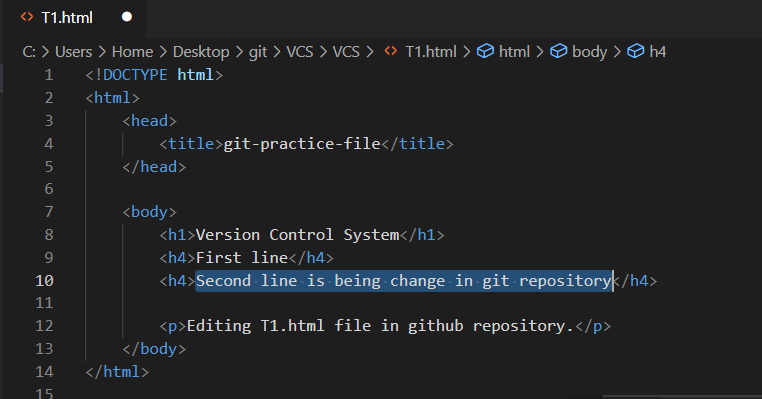
Error occur because we are trying to change same line locally and in central repository simultaneously.

Choose one of four options:

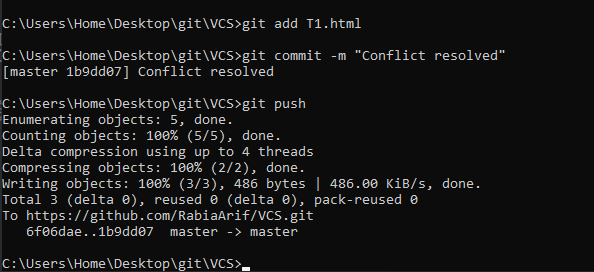


(Here we are choosing) **Accept incoming Change**

Now we have commit that was done in git central repository



After merging conflict has been resolved, push data



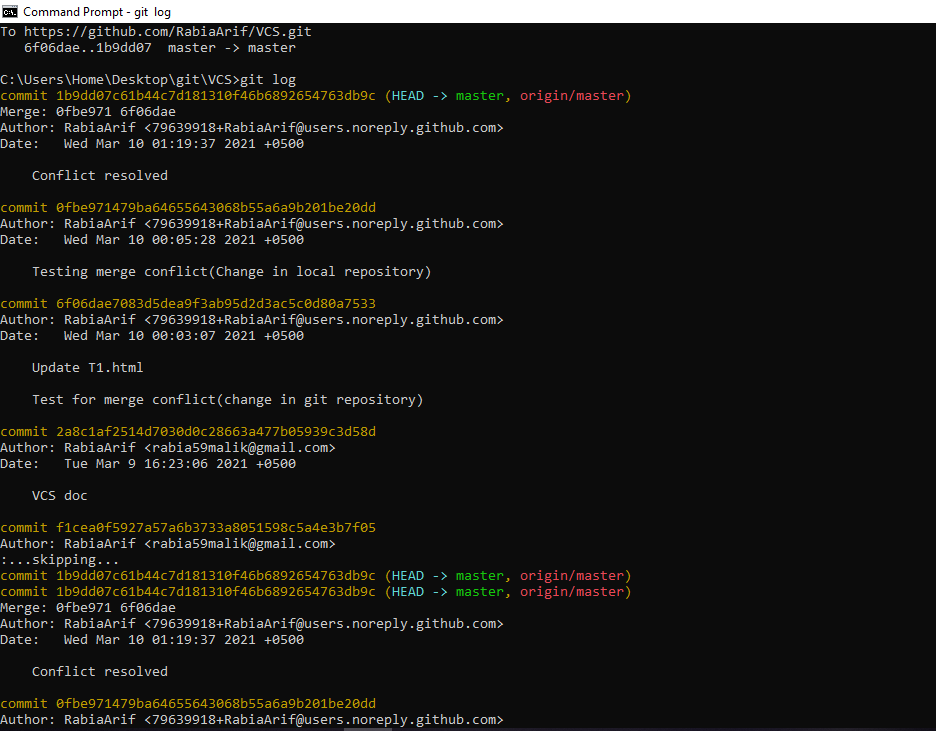
**More commands**

1. **git log:**

The git log command displays all of the commits in a repository’s history.

By default, the command displays each commit’s:

* Secure Hash Algorithm (SHA)
* Author
* Date
* Commit message



1. **Remove file**

Using commands

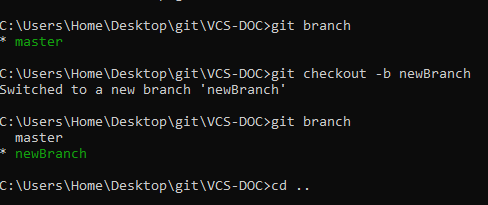
1. **git rm** {fileName}
2. **git commit –m** {“message”}
3. **git push**

**Branching**

**Branching**, in version control and software configuration management, is the duplication of an object under version control. Each object can thereafter be modified separately and in parallel so that the objects become different. In this context the objects are called branches.

Branching allows teams of developers to easily collaborate inside of one central code base. When a developer creates a branch, the version control system creates a copy of the code base at that point in time. Changes to the branch don't affect other developers of the team.

* **git branch**
* **git check**
* **git merge**

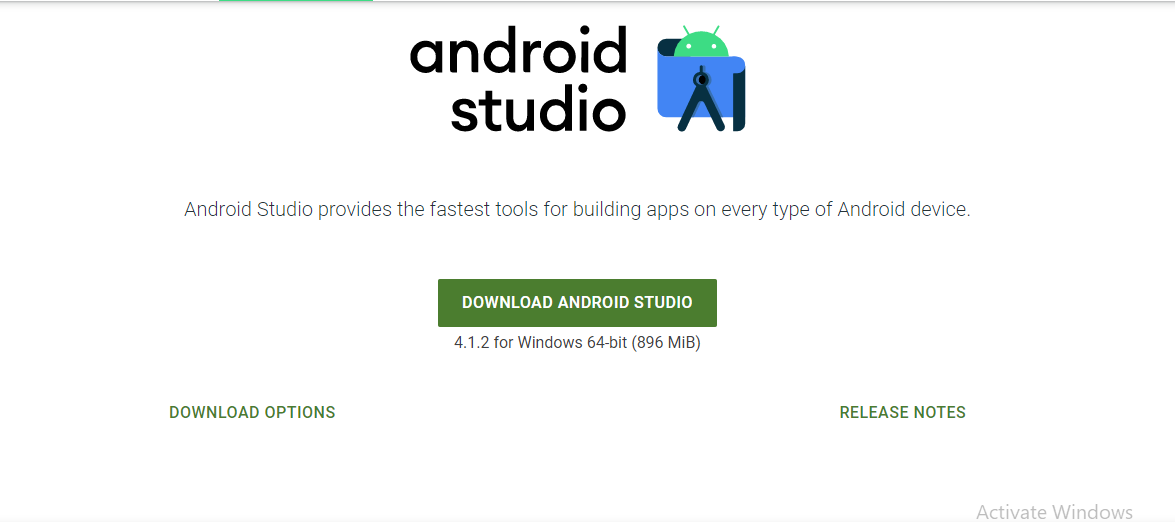


**ANDROID STUDIO**

Step-0:

Install Android Studio

[**https://developer.android.com/studio**](https://developer.android.com/studio)

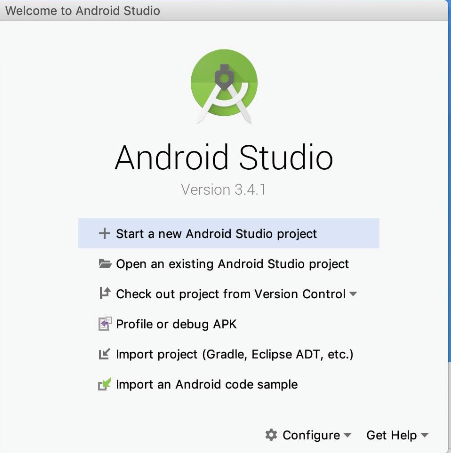


Check out for your System Requirements



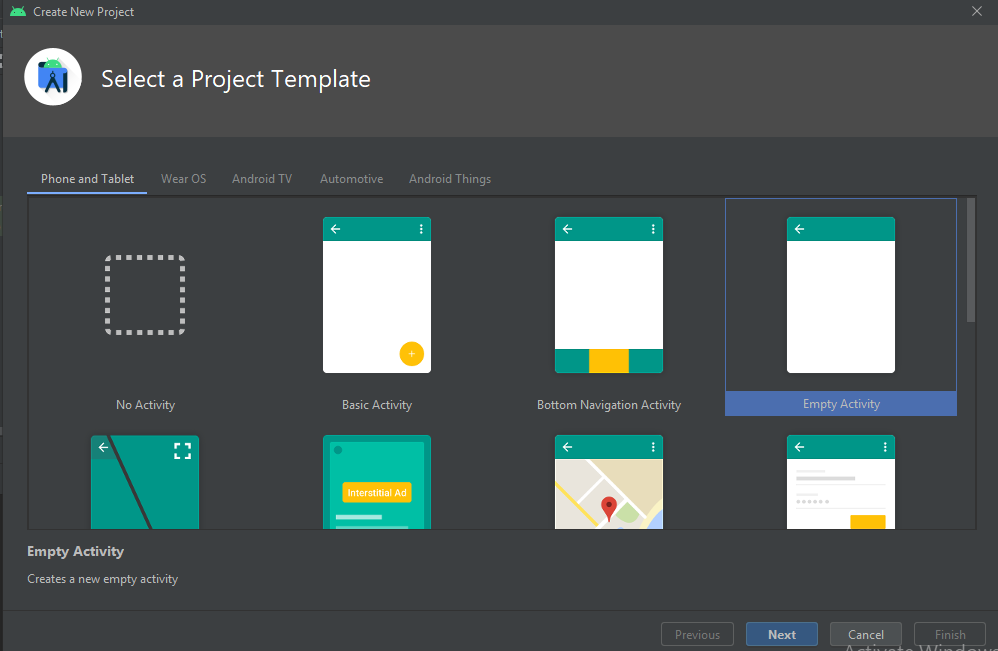
Step-1:

**Create new project**



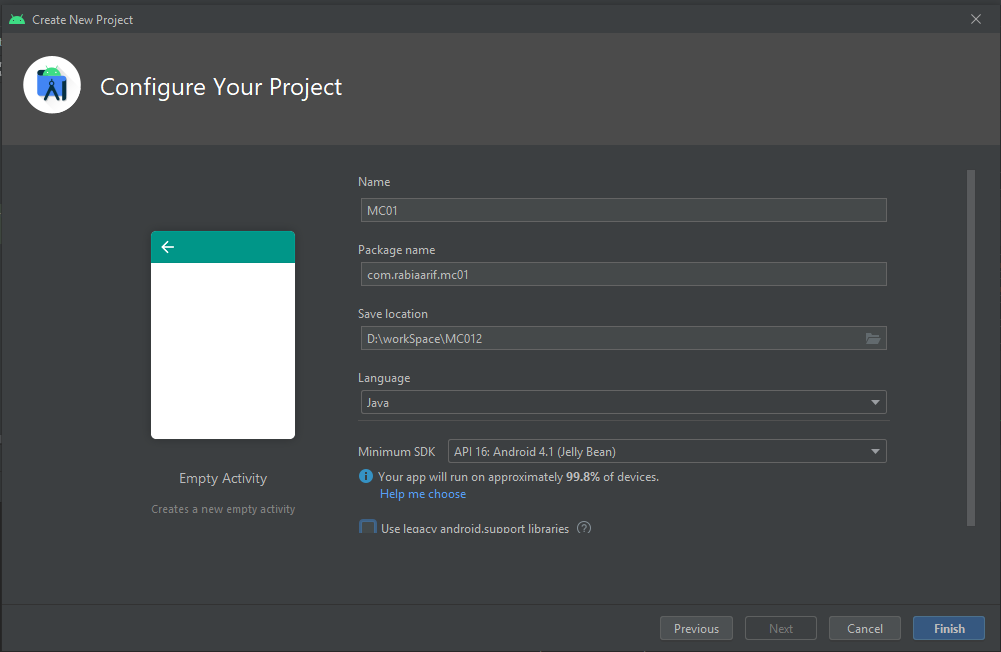
Step-2:

**Select Project Template**

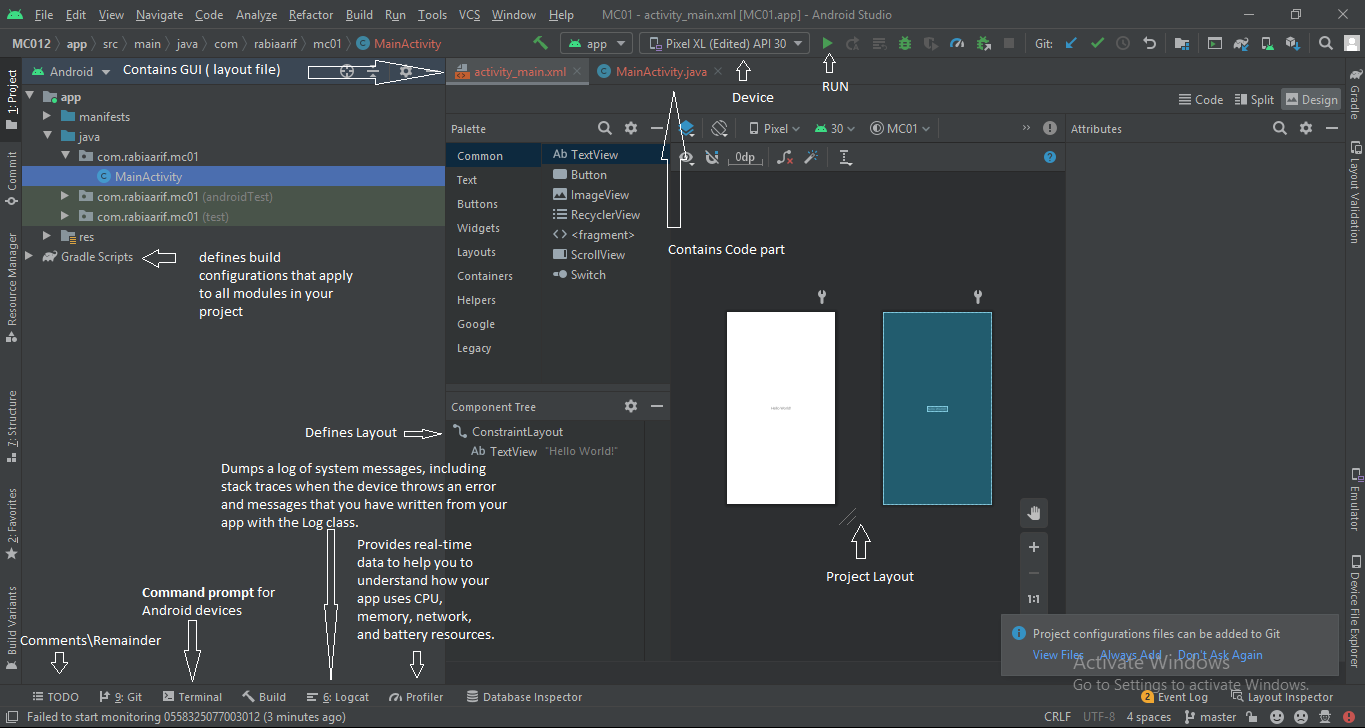


Step-3:

**Configure your Project**



1. Name
2. A unique package name
3. Location
4. Language : JAVA
5. Minimum API level / SDK (min. version of Android OS required to run your app)



Project Layout:

2 types :

* Design
* Blueprint

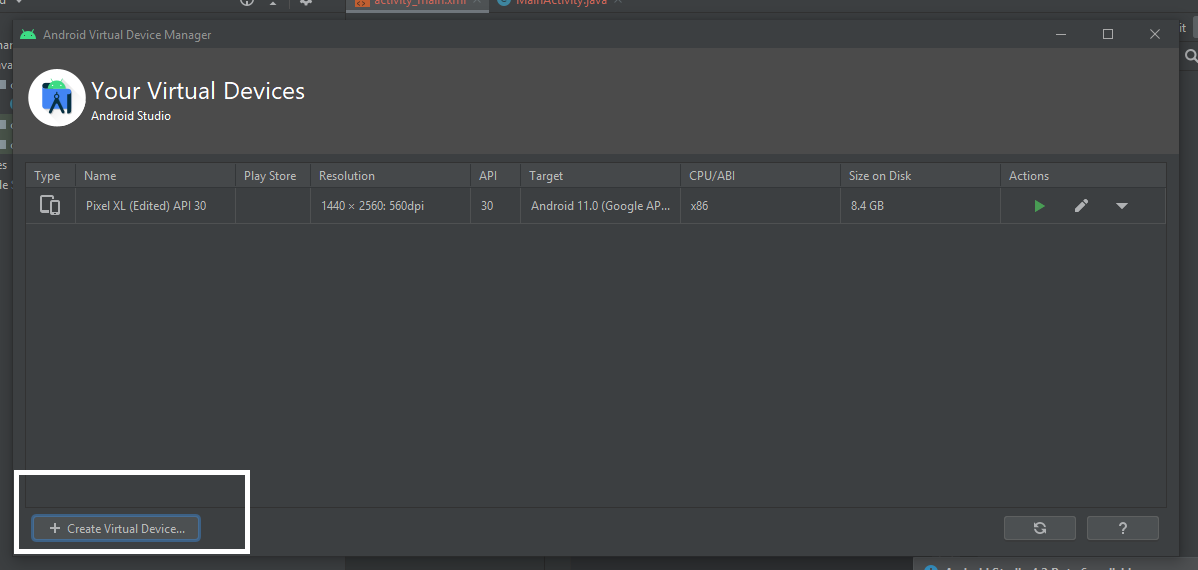
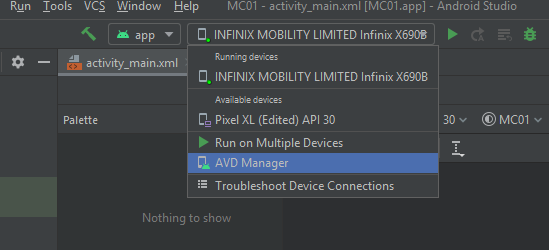
Step-4:

**Select device**

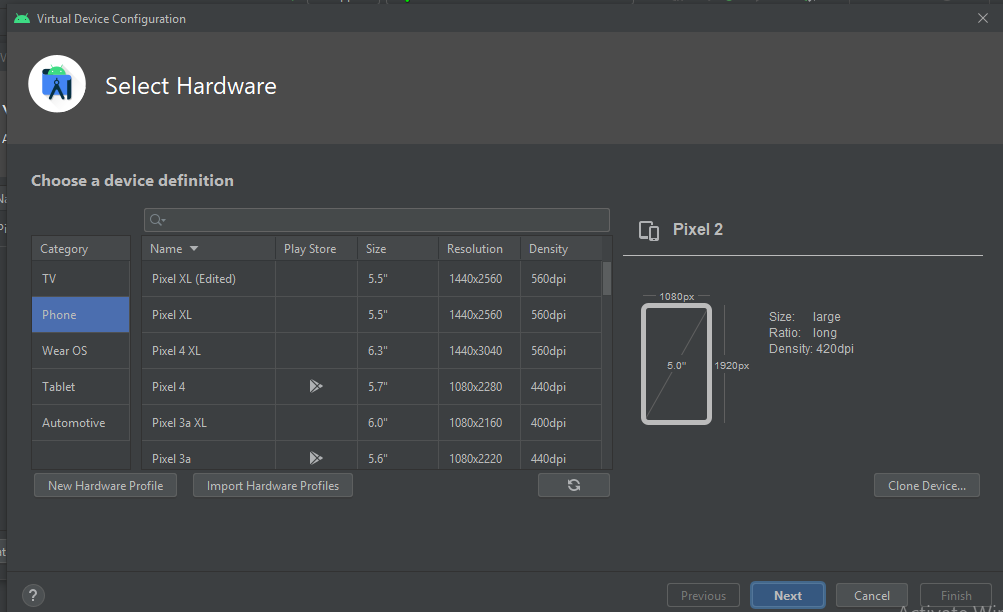
Using AVD Manager

**AVD Manager:**

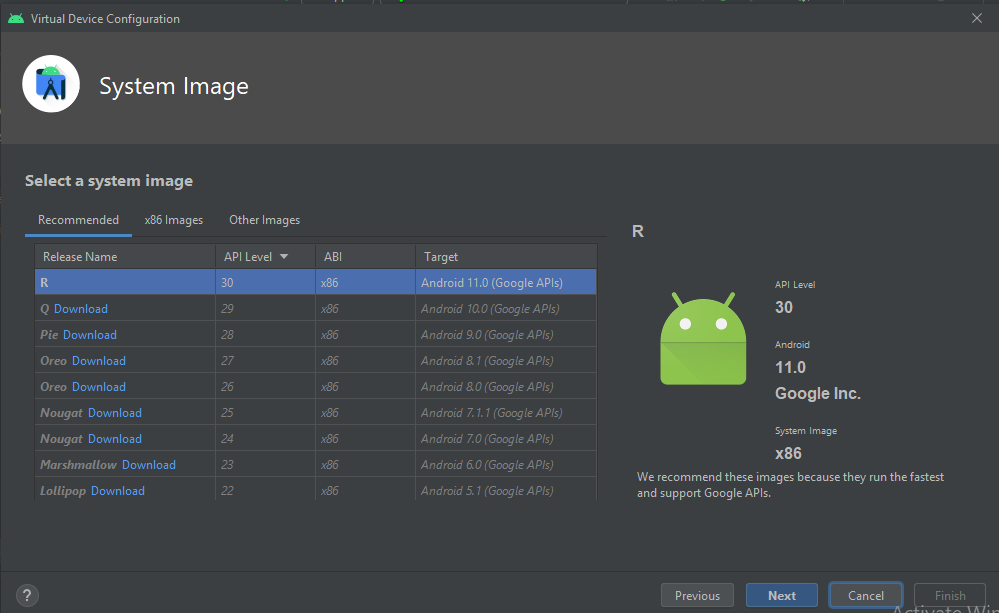
An **Android Virtual Device** (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The **AVD Manager** is an interface you can launch from Android Studio that helps you create and manage AVDs.



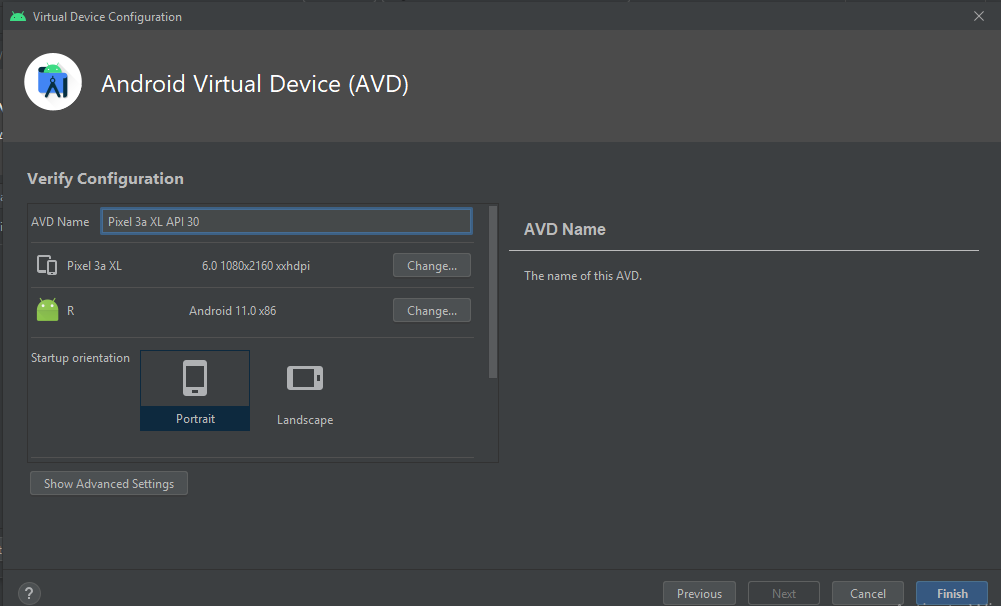
Choose any device 🡪



Next🡪

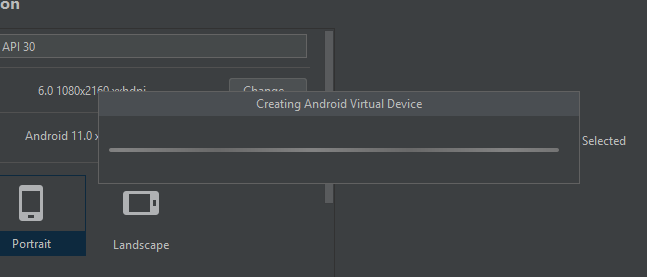


Next🡪

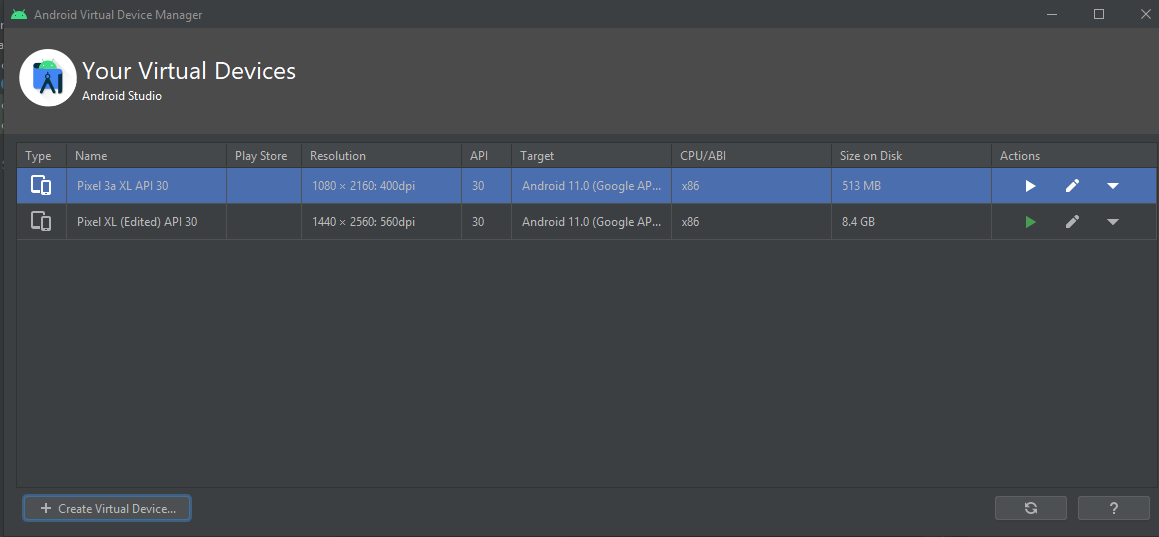


Finish

Downloading/Creating new virtual device

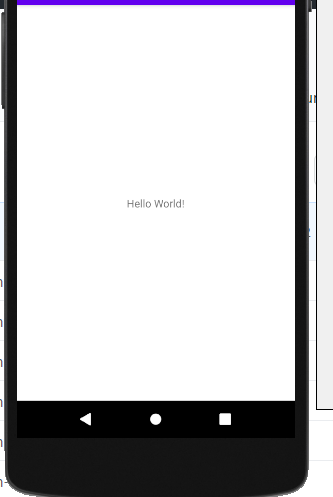


New device available in AVD Manager



**Run**

**Emulator:**



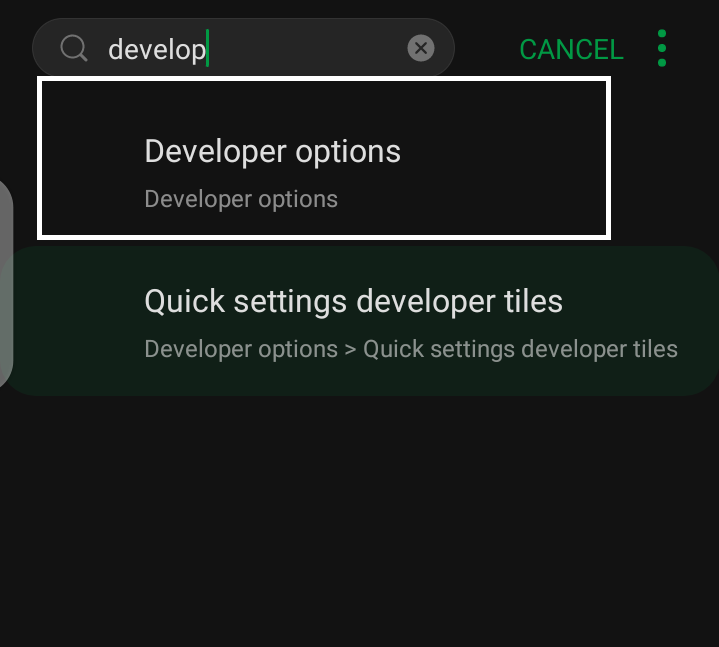
**To run your app on your mobile**

Step-0:

Connect mobile with pc using data-cable

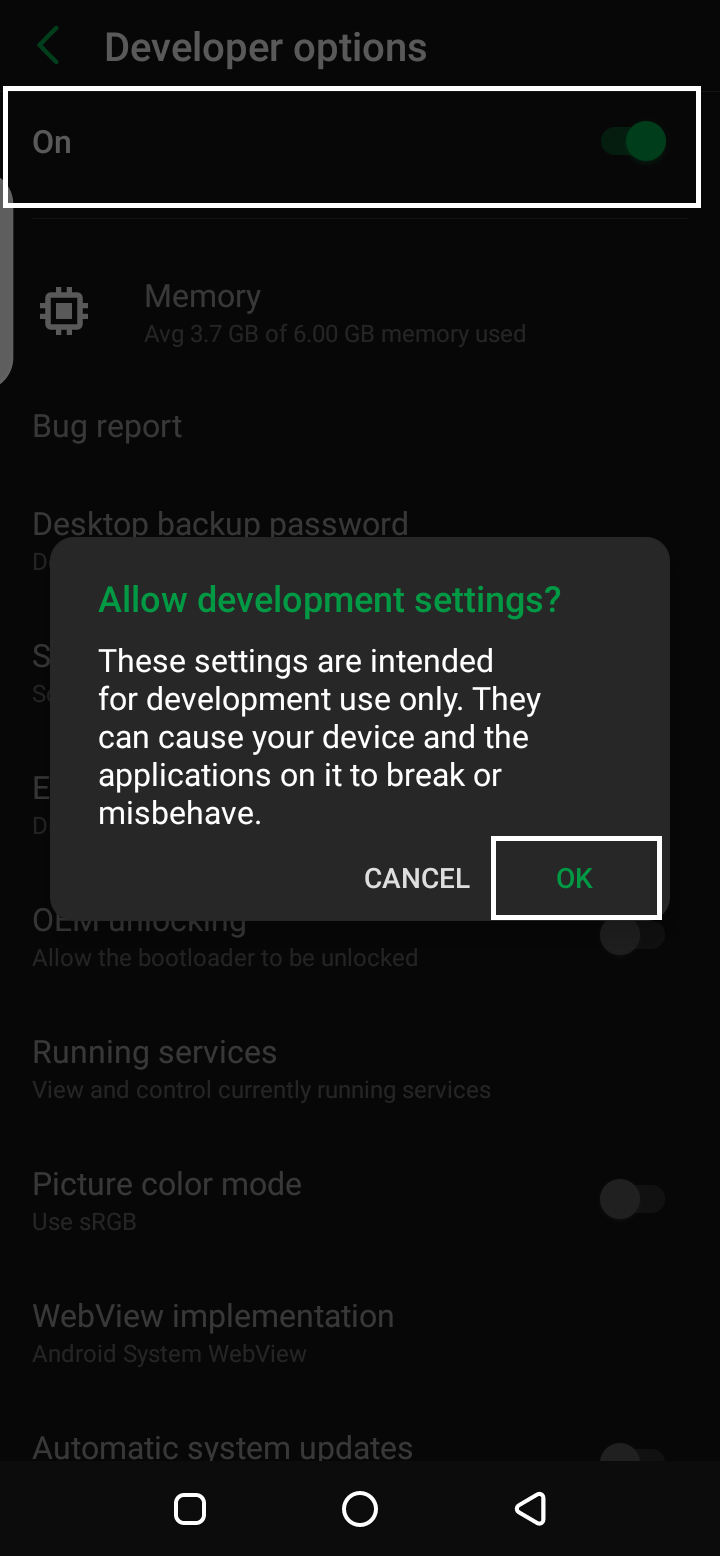
Step-1:

**Search for Developer Options in Settings**



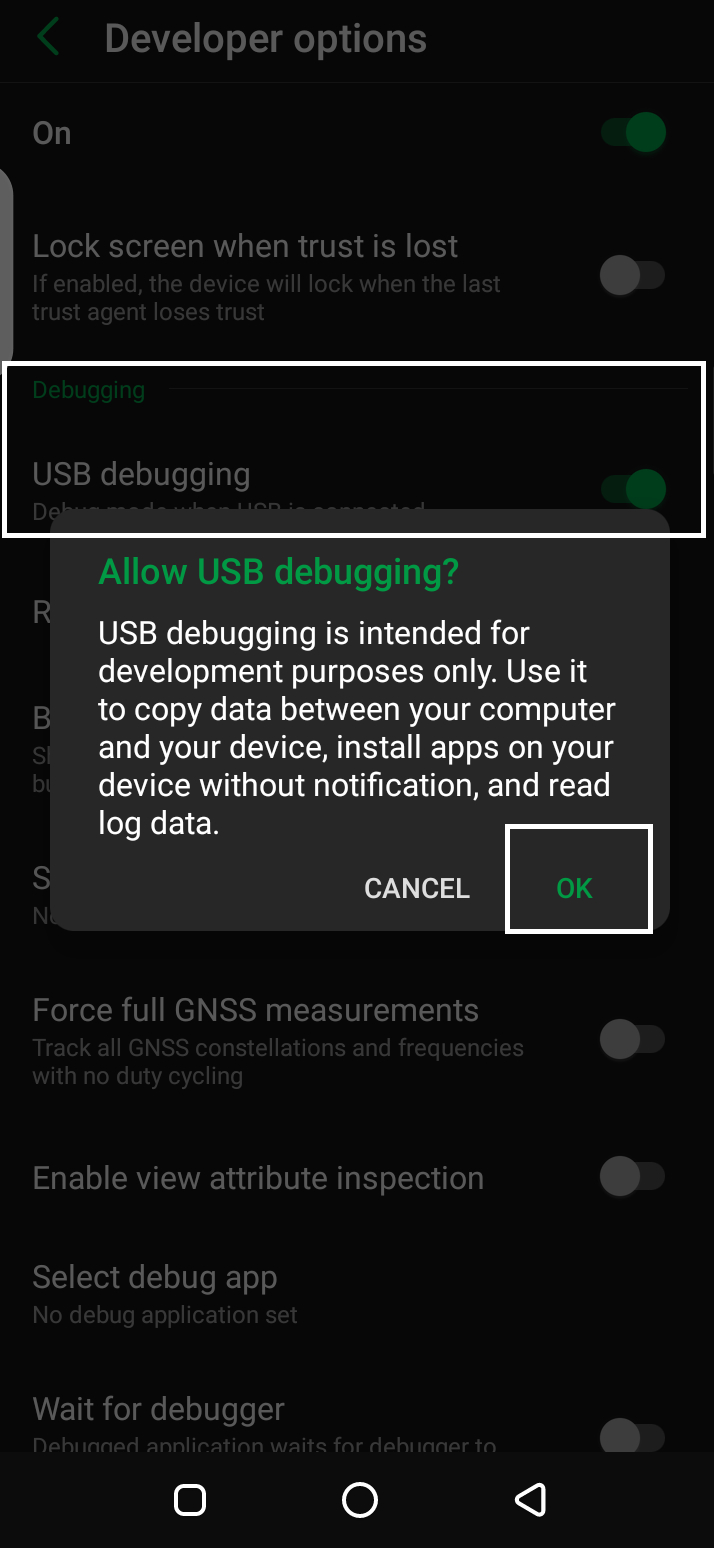
Step-2:

**Turn on developer options**

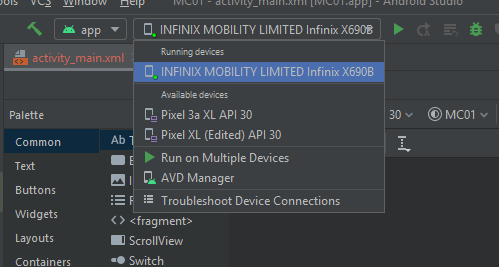


Step-3:

**Allow USB debugging**

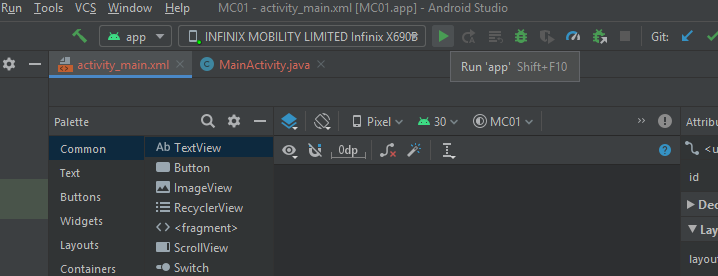


Device available in AVD Manager

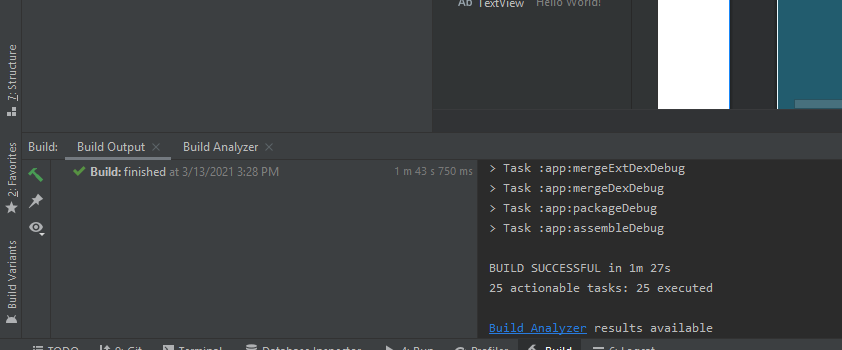


Step-5:

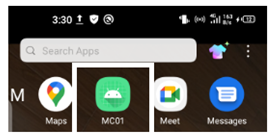
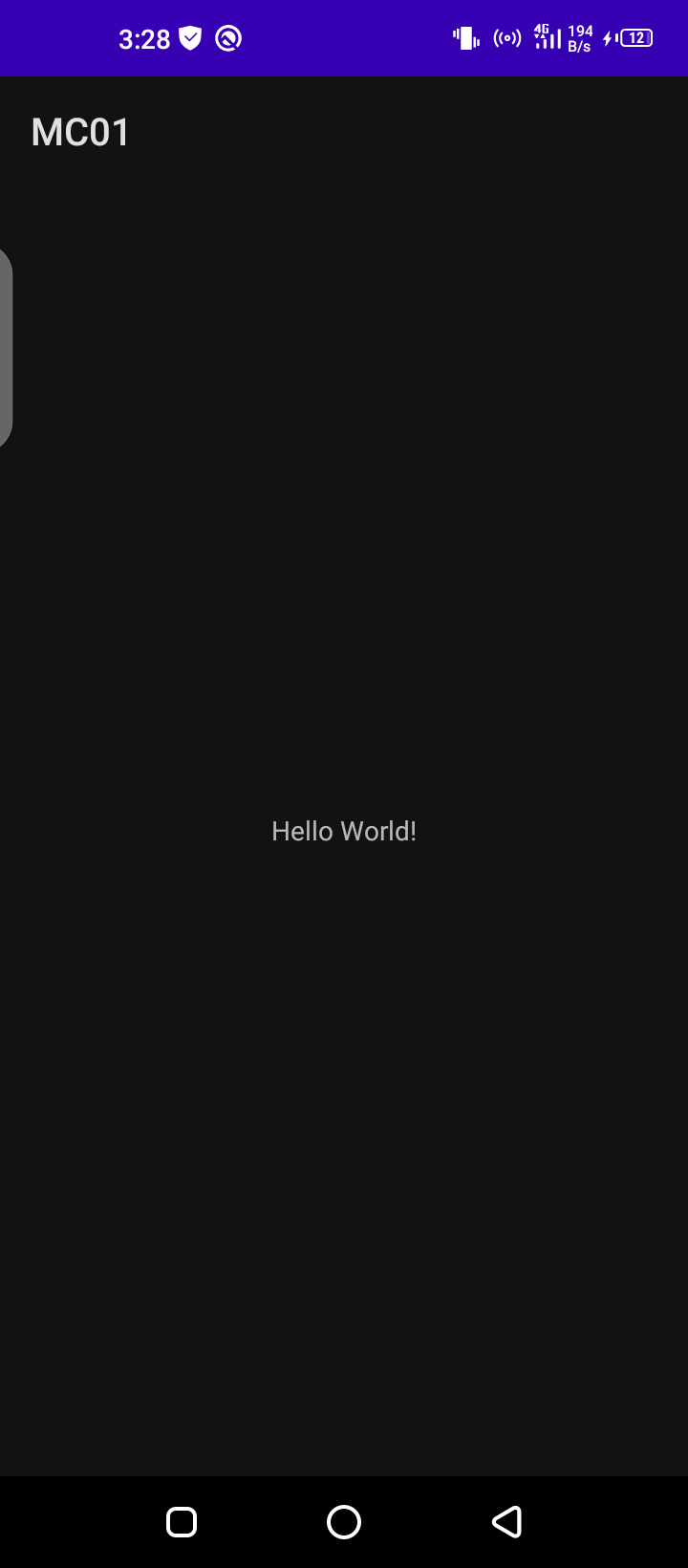
**Run**



**Build output:**



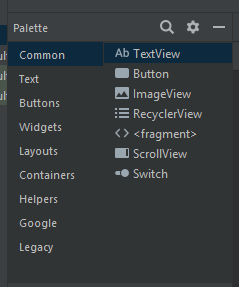
App running on phone



**Multiple\_Screen\_Task**

( <https://github.com/RabiaArif/Multiple_Screen_Task.git> )

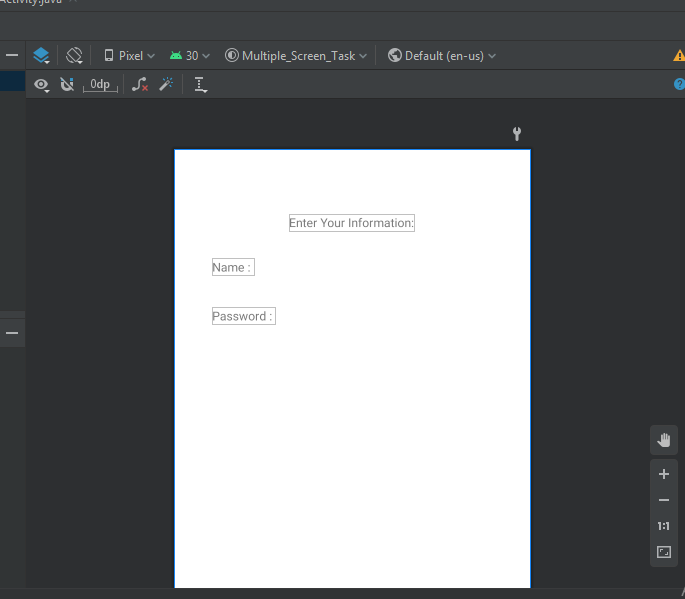
Palette



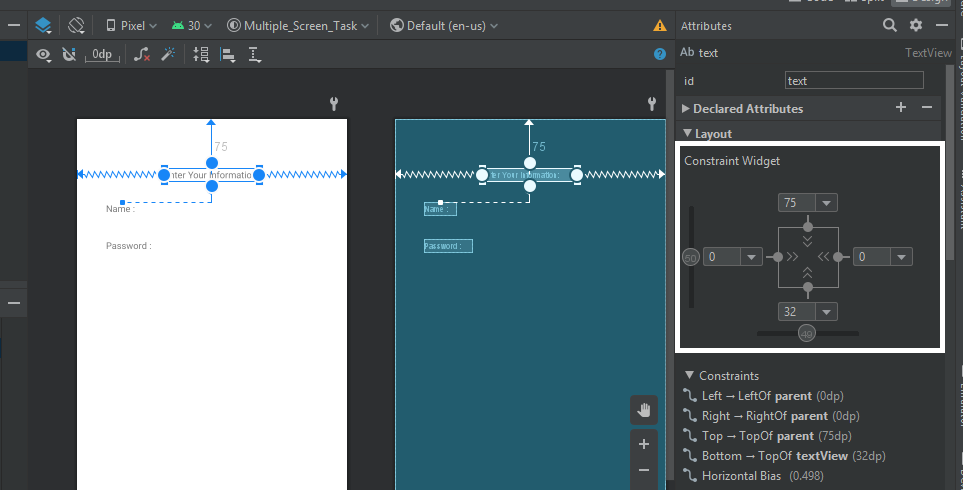
Step-1: **Add and drop**

We can set text/Button/Image and much more by simply **Add and drop**

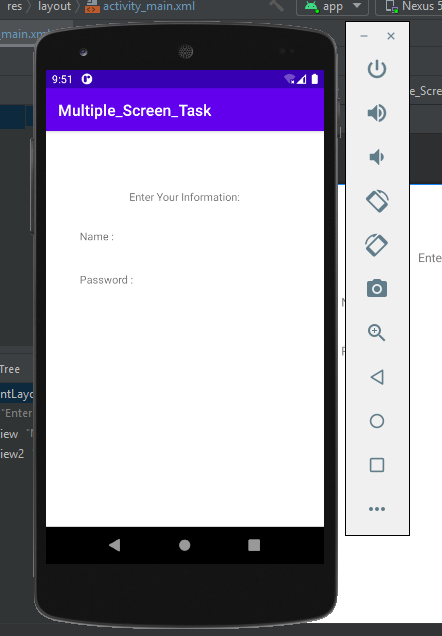
Initially adding three textViews



Step-2: **Set Constraints for each item**

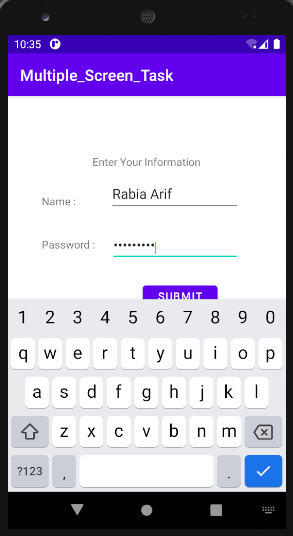
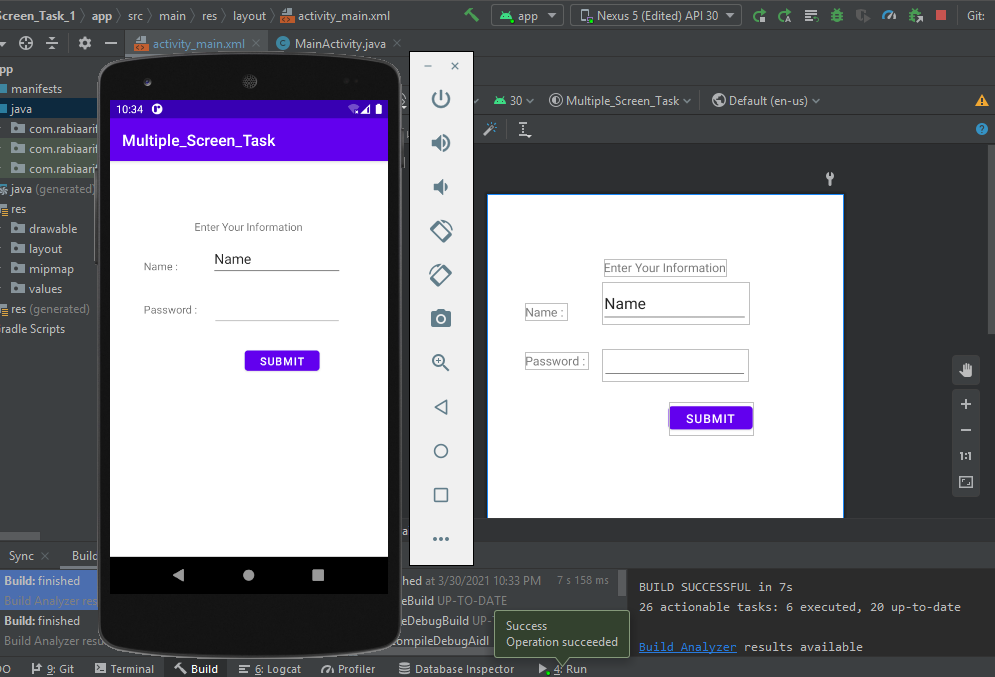


**Screen 1:**

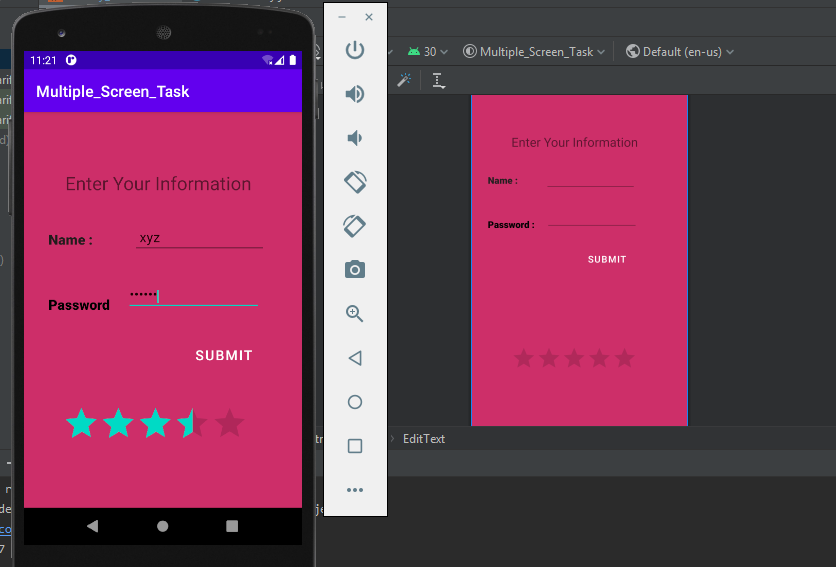


Adding more items like textFields and button

**Screen 2:**



**Screen 3:**



**Layout**

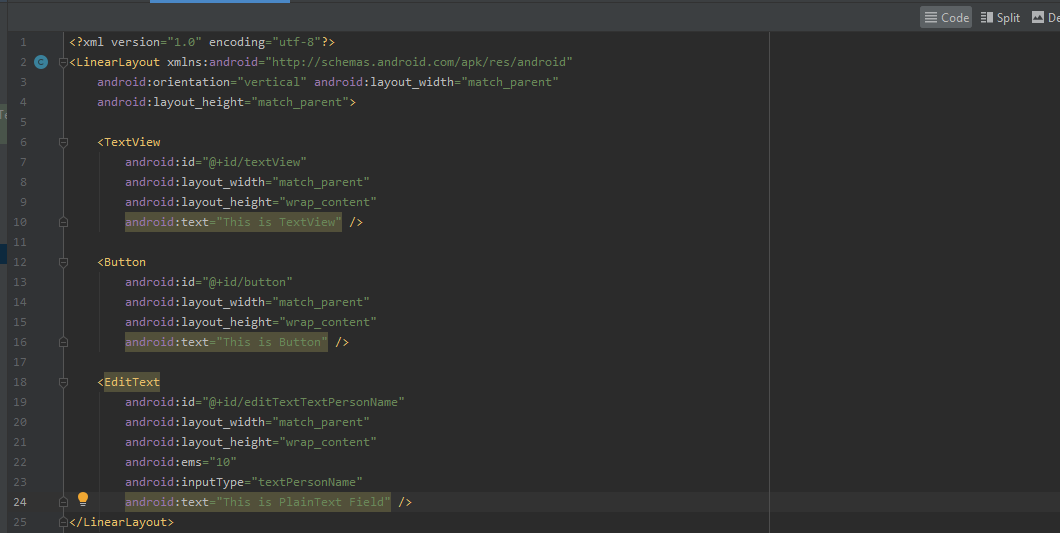
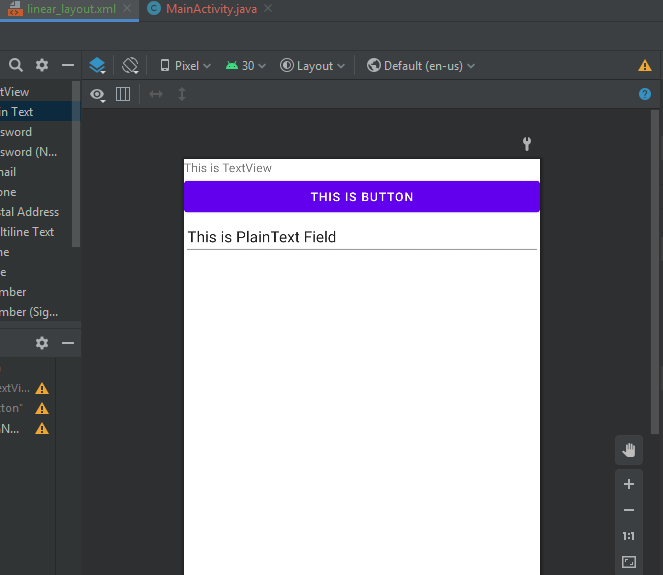
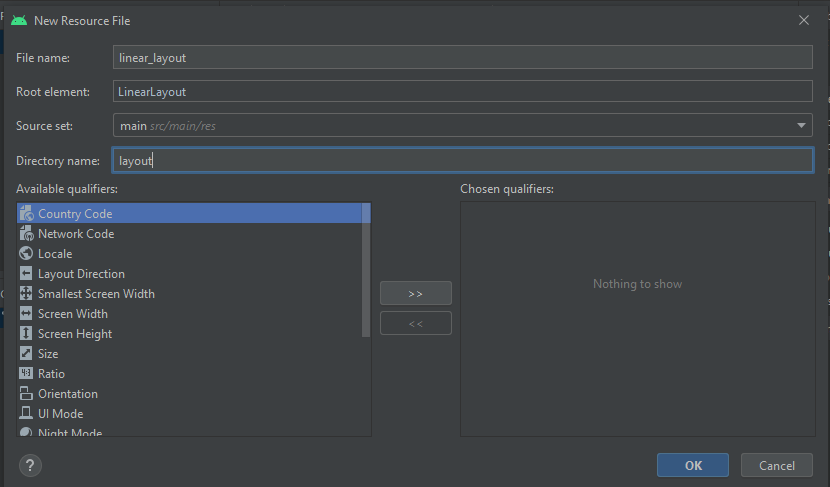
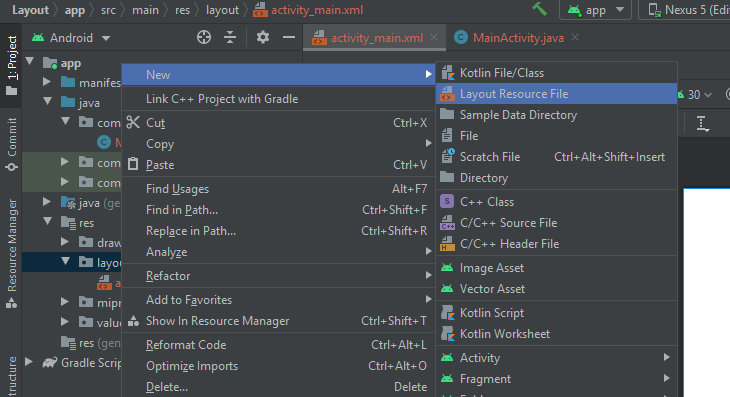
**View:** A view usually draws something the user can see and interact with. The View objects are usually called ‘widgets’ and can be one of many subclasses, such as Button, TextView, etc.

**ViewGroup:** An **invisible** container. The ViewGroup objects are usually called ‘layouts’ can be one of many types that provide a differnet layout structure, such as **LinearLayout** or **ConstraintLayout.**

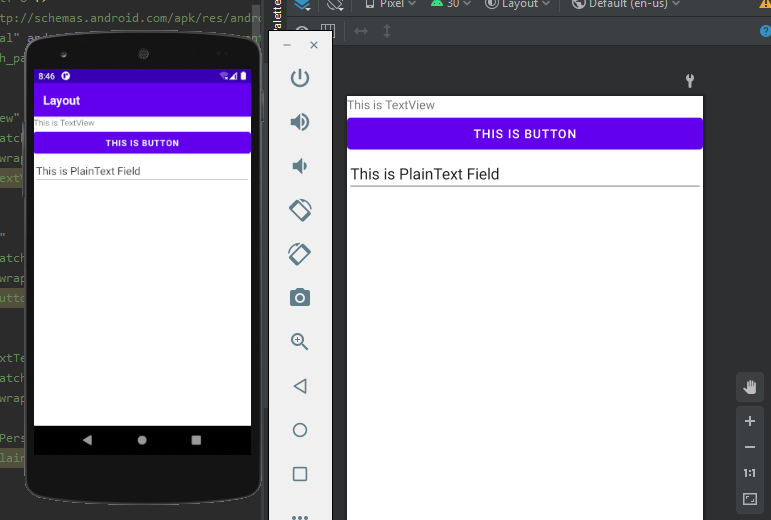
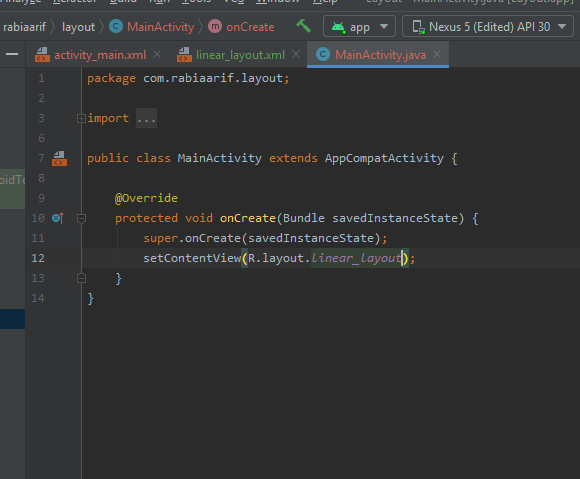
(Incomplete)

Few Examples of constraint layout

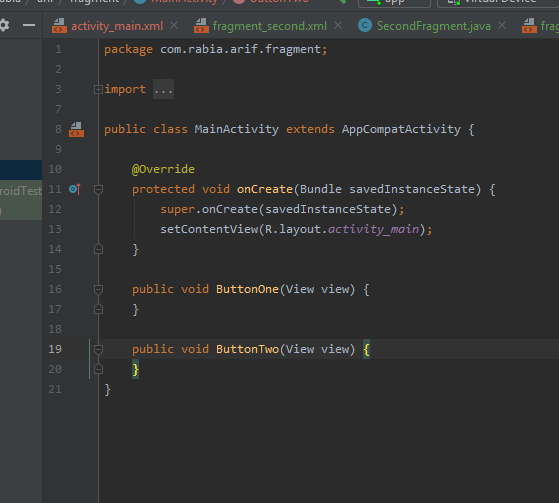
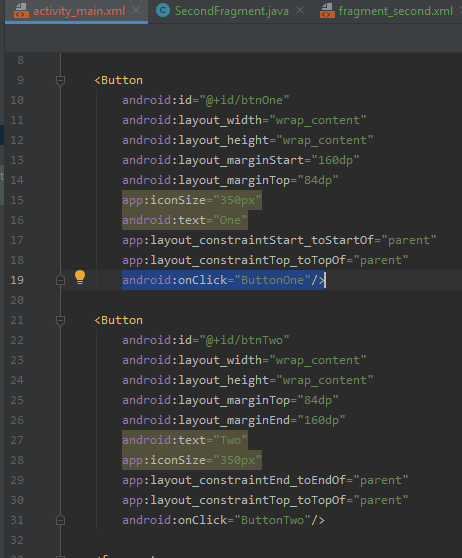
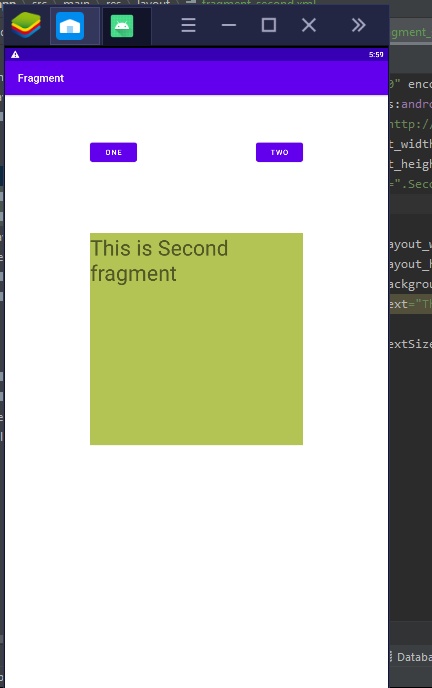
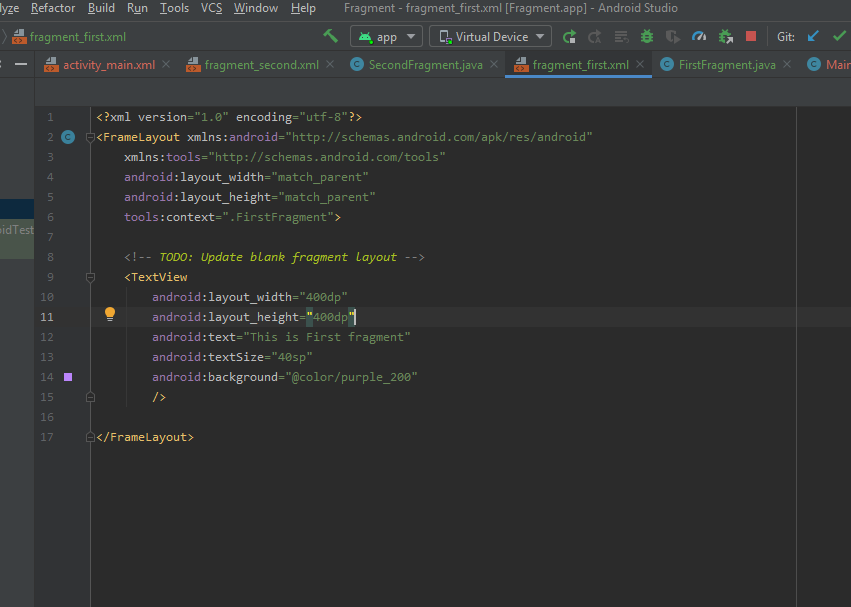
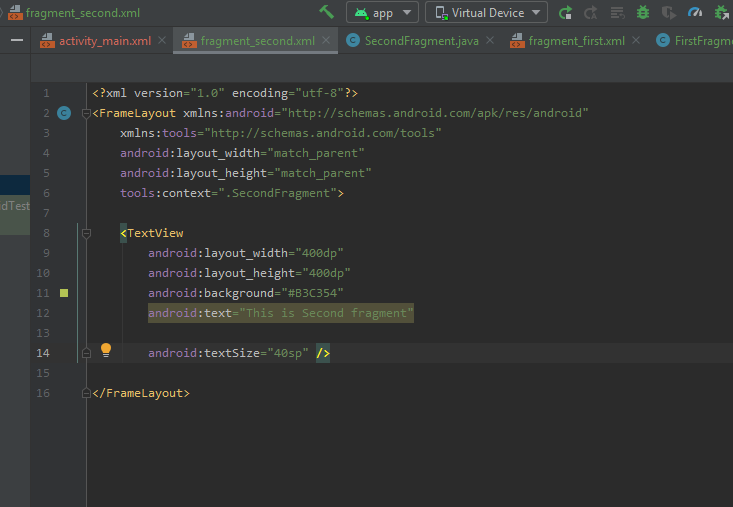
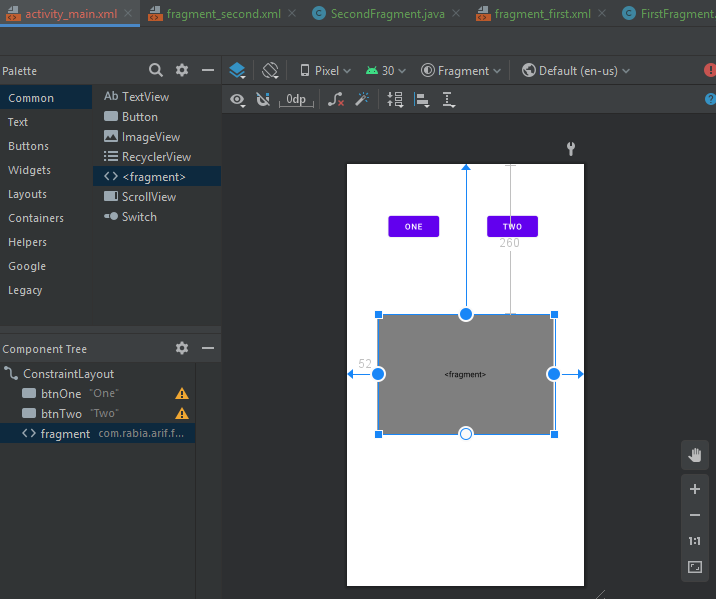
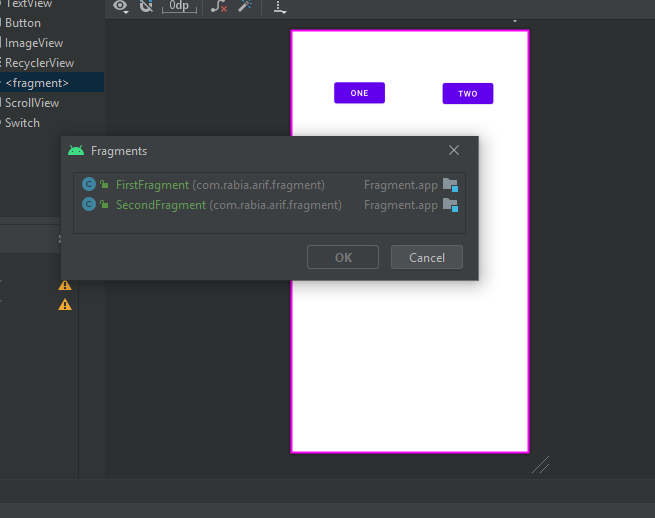
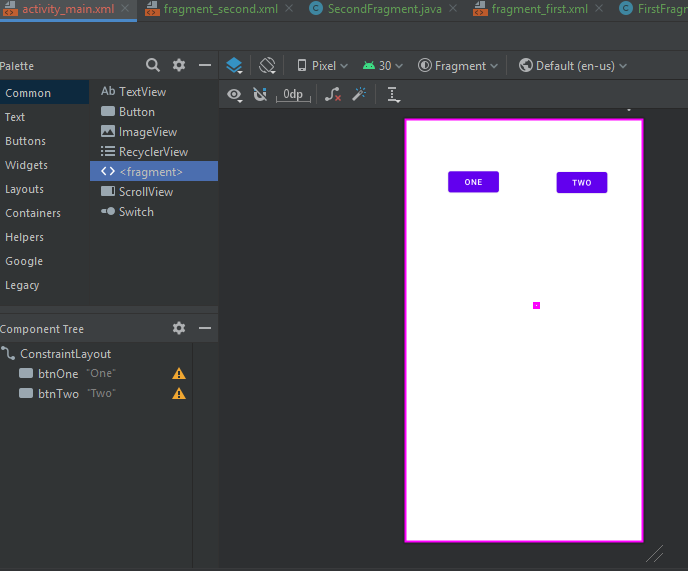
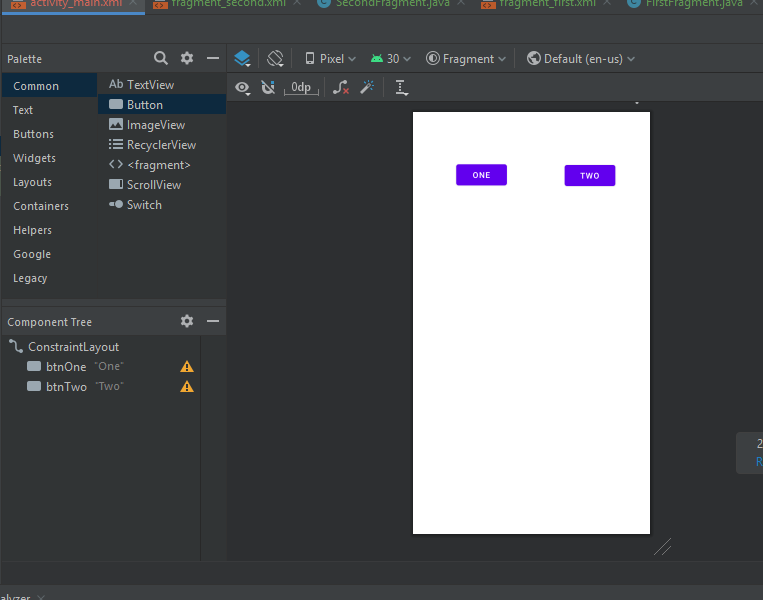
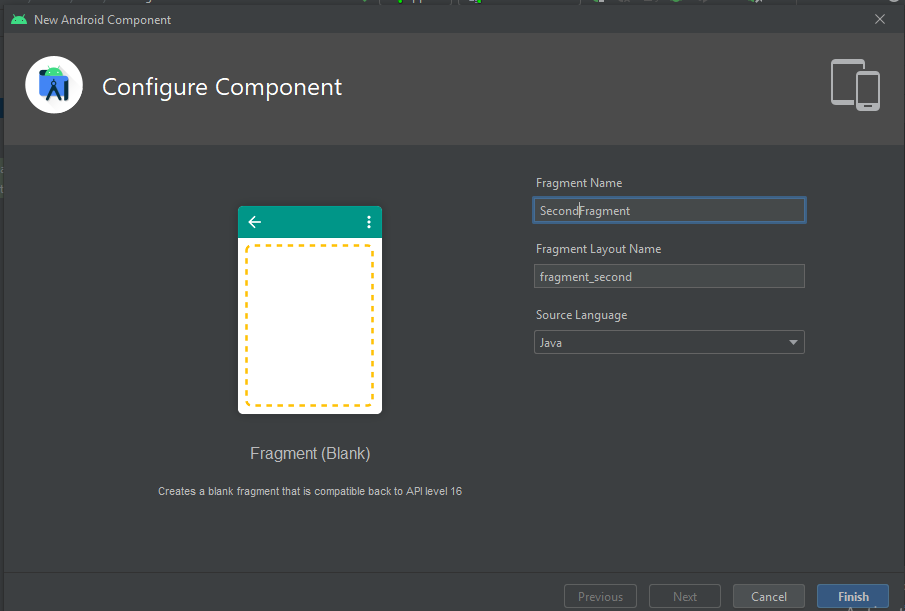
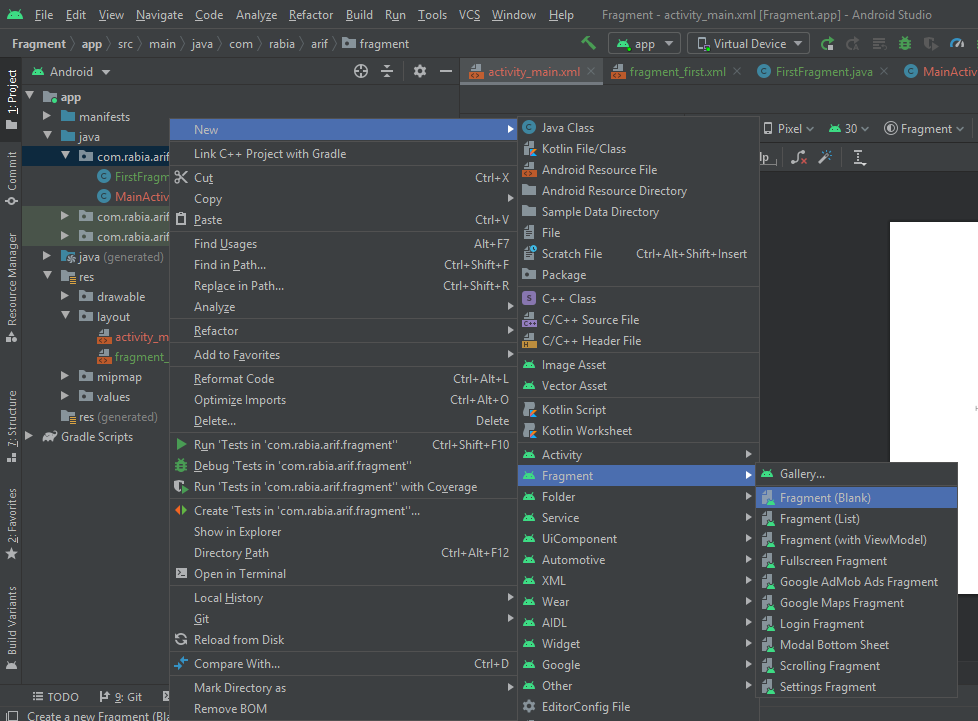
Linear Layout



Change start point of app by changing activity\_main with new layout file i.e. linear\_layout



**Fragment**



FragmentTransaction fragmentTransaction;

Write following code in onClickListener

fragmentTransaction=getSupportFragmentManager().beginTransaction();  
FirstFragment firstFragment = new FirstFragment();  
fragmentTransaction.replace(R.id.*fragment*,firstFragment);  
fragmentTransaction.commit();

