Creation of aar file for the android development of ipfs.

This dcument will be focusing on the construction of the aar files that are required by the https://github.com/ipfs-shipyard/gomobile-ipfs library and for implementing the custom ipfs commands that we are going to implement in the future.

In Brief , most of the work will be done in linux systems as windows was struggling with multiple issues and low support on the topic.

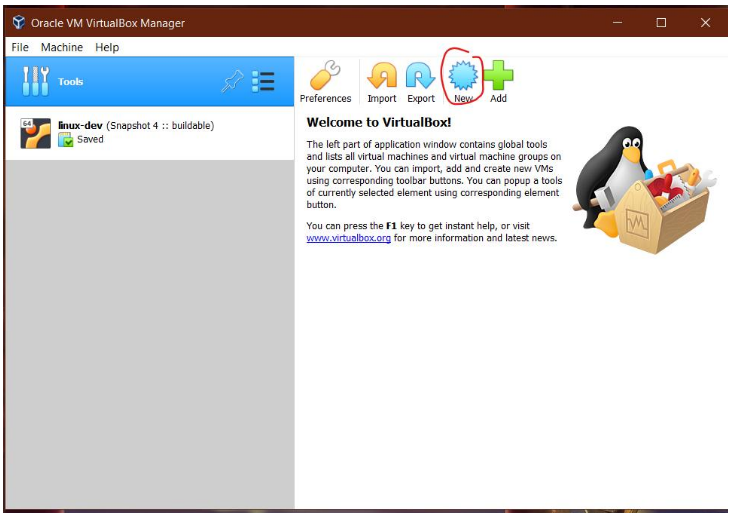
In my case I used virtual box as a supplement to work things out.
And this Guide will also include the steps related to that.

<hr>

Setting Up Virtual Box

The first step will be installation of Virtual Box. Well assuming you have downloaded one you need to have these additional things to make it run similar to my environment.
- Ubuntu OS iso file.. { preferably the current LTS version}
- Virtual Box Extension Pack
- and Internet

Create a New instance.



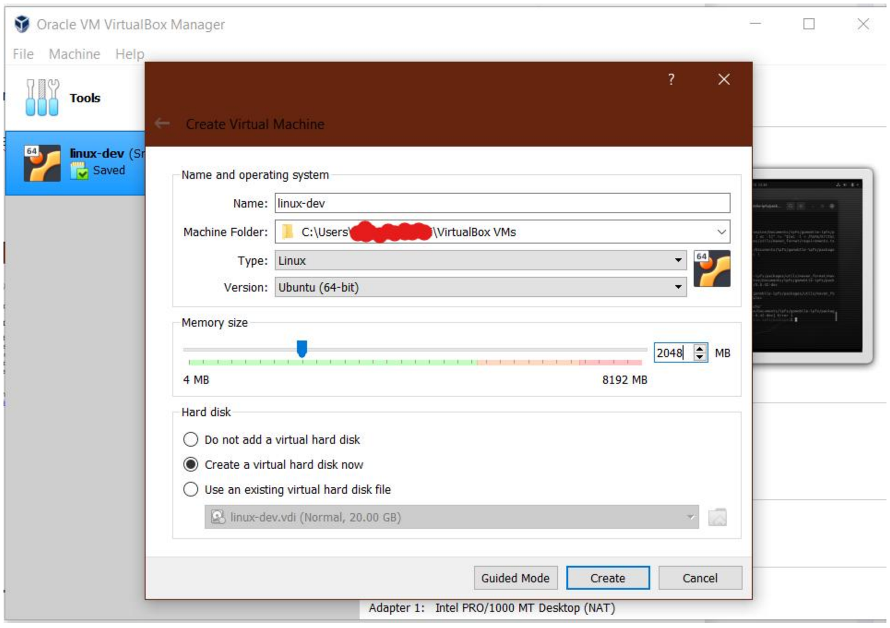
linux-dev (Snapshot 4 :: buildable)
Saved

Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Give it an appropriate name, and put it in a location you can remember, and rest of the settings as follows.



Name and operating system

Name: linux-dev

Machine Folder: C:\Users\[redacted]\VirtualBox VMs

Type: Linux

Version: Ubuntu (64-bit)

Memory size

2048 MB

4 MB 8192 MB

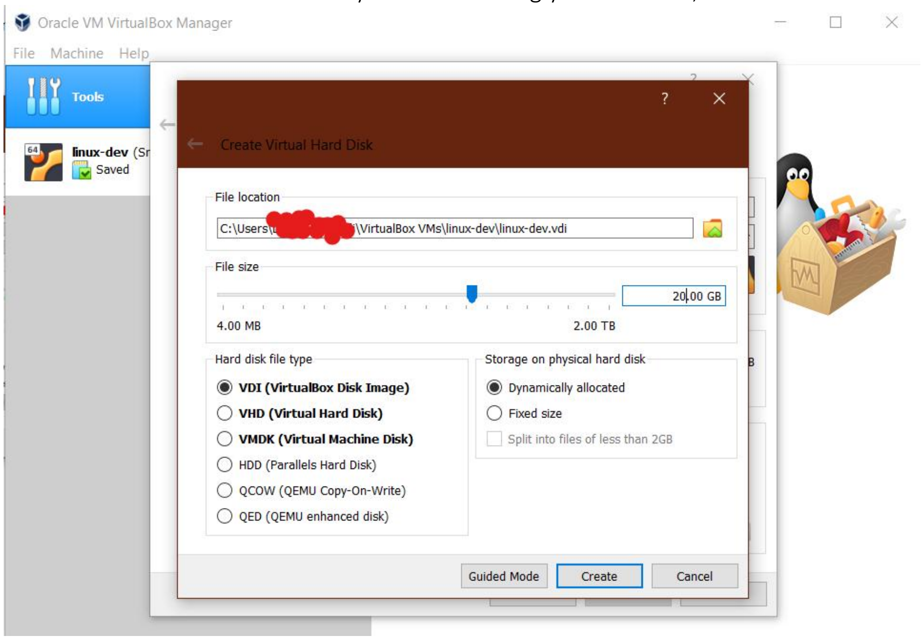
Hard disk

- ○ Do not add a virtual hard disk
- ● Create a virtual hard disk now
- ○ Use an existing virtual hard disk file

linux-dev.vdi (Normal, 20.00 GB)

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

Put at least 20GB of HDD memory for it to not bug you in future , then hit create


Create Virtual Hard Disk

File location

C:\Users\[redacted]\VirtualBox VMs\linux-dev\linux-dev.vdi

File size

20.00 GB

4.00 MB — 2.00 TB

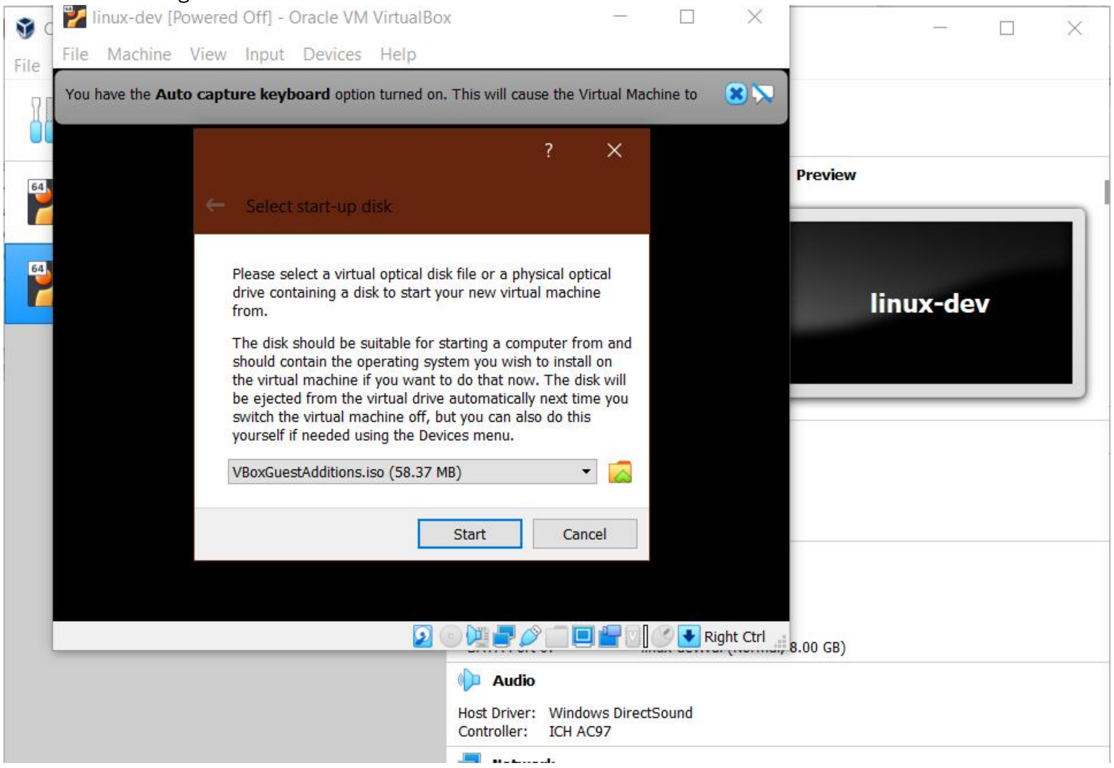
Hard disk file type

- [✓ VDI (VirtualBox Disk Image)]
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)
- HDD (Parallels Hard Disk)
- QCOW (QEMU Copy-On-Write)
- QED (QEMU enhanced disk)

Storage on physical hard disk

- [✓ Dynamically allocated]
- Fixed size
- Split into files of less than 2GB

Now hit start.
You will now be greeted with this screen


linux-dev [Powered Off] - Oracle VM VirtualBox

You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to

Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

VBoxGuestAdditions.iso (58.37 MB)

Host Driver: Windows DirectSound
Controller: ICH AC97

Click on the folder icon, then ‘add’, select the ubuntu LTS iso file, then choose.

Then complete the installation.
{do minimal installation as we don’t need any additional tools}

With this we have out OS set up

Creating the Environment…

First open up the terminal and run the following commands . This will install the required Libs needed for creation of packages like gcc and make.

```
$ sudo apt update && sudo apt upgrade
```

```
$ sudo apt install build-essential
```

```
$ sudo apt install manpages-dev
```
{ not so required }

With this we have gcc in our system to run c programs.
Verify it by running:

```
$ gcc --version
```

Now its time to set up Android SDK and NDK for our work

```
$ sudo apt install android-sdk
```

Well sdk installation is simple.

Well NDK is a little complex. Go to the following link. And download the ndk package for linux
developer.android.com/ndk/downloads

Unzip in the downloads folder, or the folder it was downloaded in. You will see a folder named 'android-ndk-[version_name]' created. Now open the terminal on the folder location with a right click. And type in the command

```
$ sudo mv android-ndk-[version_name] /usr/lib
```

This will move this folder to the location under '/usr/lib/'. Head to that location to verify the lib, you might also find your 'android-sdk' folder here.

Similarly we will install 'go'.
Go to this link : go.dev/dl/
We need to download the go version 1.17.8 or directly download from this link
https://go.dev/dl/go1.17.8.linux-amd64.tar.gz

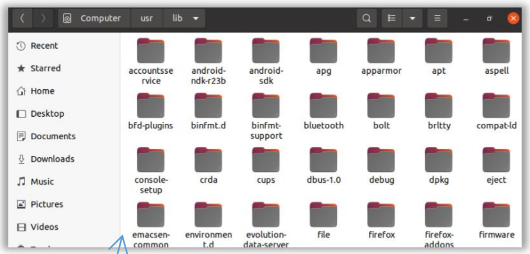
Well this is because half of the libraries we are using in our project are still not made for go 1.18 …
And it also seems that the previous devs also used go 1.17 to create this lib, so we will be doing the same.

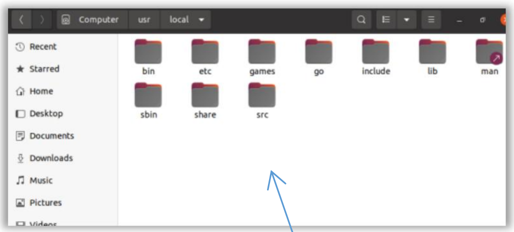
Now again Unzip it in the current folder, it will create a 'go' directory with go-tree inside it.
Now open the terminal in the current folder to run the following command.

```
$ sudo mv go /usr/local
```

This will move this folder to the location under '/usr/local/'.

Make sure that your directories are structured something like this.





In usr/lib

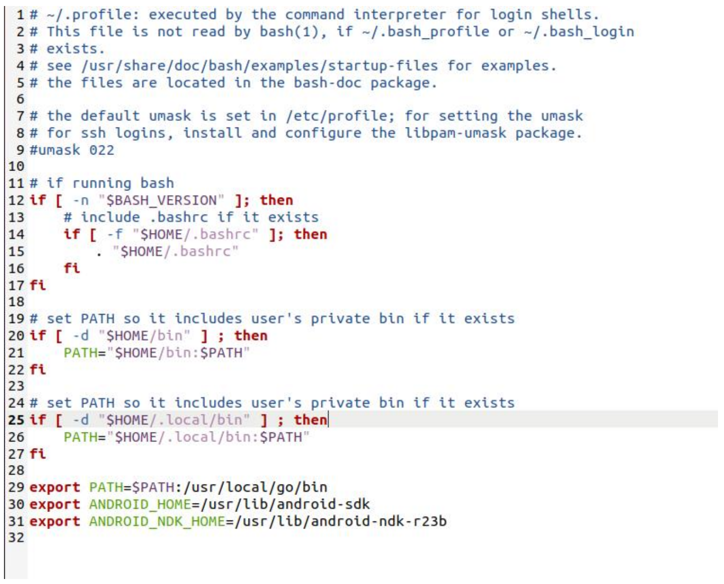
In usr/local

Now open the terminal :
And type the following command.

```
$ gedit ~/.profile
```

It will open a text editor where we can put our path variables.
At the End of File, add the following lines.

```
export PATH=$PATH:/usr/local/go/bin
export ANDROID_HOME=/usr/lib/android-sdk
export ANDROID_NDK_HOME=/usr/lib/android-ndk-[version_name]
```



```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export PATH=$PATH:/usr/local/go/bin
export ANDROID_HOME=/usr/lib/android-sdk
export ANDROID_NDK_HOME=/usr/lib/android-ndk-r23b

```

Save the file then on the same terminal run the following command. { and every time when terminal says unknown command 'go' }

```
$ source ~/.profile
```

With this we have completed our environment.
We will now move on to the next Step.

Configuring the Project.

We need to have two repos in of system present.
Ie;

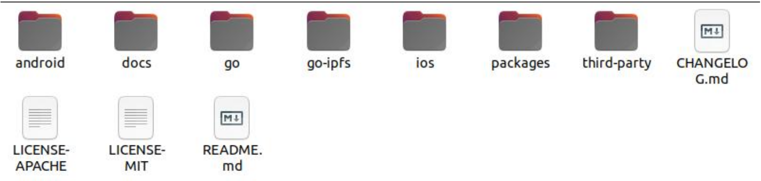
https://github.com/ipfs-shipyard/gomobile-ipfs
https://github.com/ipfs/go-ipfs { the one you modified }

In my case I added a custom command named crazyTest which will return the ID hash and addressed to connect to our ipfs daemon.
PS:: I realized I haven't created a doc of how I did it. Tell me If I need to.

Now move the go-ipfs folder { 'github.com/ipfs/go-ipfs' this one} into the gomobile-ipfs { 'github.com/ipfs-shipyard/gomobile-ipfs' this one }...

Now we will only work with gomobile-ipfs this folder.

The directory may look something like this after we did the above steps.



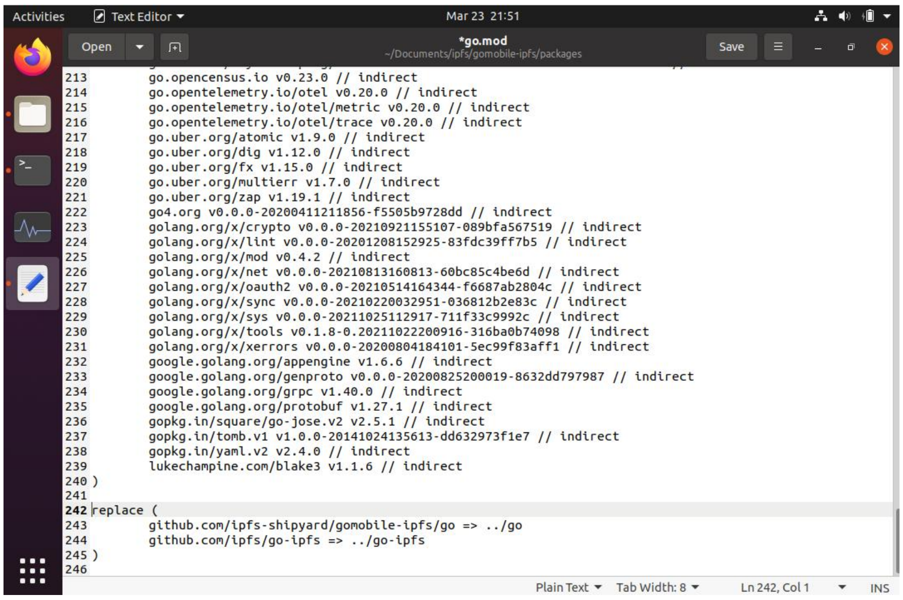
Here go to 'packages' and you will find a makefile and go.mod files...
here we will work with the go.mod file . For that open the terminal and typein

```
$ gedit go.mod
```

In the editor in the EOF add the following line

```
replace (
        github.com/ipfs-shipyard/gomobile-ipfs/go => ../go
        github.com/ipfs/go-ipfs => ../go-ipfs
)
```

It should look as follows



```
	go.opencensus.io v0.23.0 // indirect
	go.opentelemetry.io/otel v0.20.0 // indirect
	go.opentelemetry.io/otel/metric v0.20.0 // indirect
	go.opentelemetry.io/otel/trace v0.20.0 // indirect
	go.uber.org/atomic v1.9.0 // indirect
	go.uber.org/dig v1.12.0 // indirect
	go.uber.org/fx v1.15.0 // indirect
	go.uber.org/multierr v1.7.0 // indirect
	go.uber.org/zap v1.19.1 // indirect
	go4.org v0.0.0-20200411211856-f5505b9728dd // indirect
	golang.org/x/crypto v0.0.0-20210921155107-089bfa567519 // indirect
	golang.org/x/lint v0.0.0-20201208152925-83fdc39ff7b5 // indirect
	golang.org/x/mod v0.4.2 // indirect
	golang.org/x/net v0.0.0-20210813160813-60bc85c4be6d // indirect
	golang.org/x/oauth2 v0.0.0-20210514164344-f6687ab2804c // indirect
	golang.org/x/sync v0.0.0-20210220032951-036812b2e83c // indirect
	golang.org/x/sys v0.0.0-20211025112917-711f33c9992c // indirect
	golang.org/x/tools v0.1.8-0.20211022200916-316ba0b74098 // indirect
	golang.org/x/xerrors v0.0.0-20200804184101-5ec99f83aff1 // indirect
	google.golang.org/appengine v1.6.6 // indirect
	google.golang.org/genproto v0.0.0-20200825200019-8632dd797987 // indirect
	google.golang.org/grpc v1.40.0 // indirect
	google.golang.org/protobuf v1.27.1 // indirect
	gopkg.in/square/go-jose.v2 v2.5.1 // indirect
	gopkg.in/tomb.v1 v1.0.0-20141024135613-dd632973f1e7 // indirect
	gopkg.in/yaml.v2 v2.4.0 // indirect
	lukechampine.com/blake3 v1.1.6 // indirect
)

replace (
	github.com/ipfs-shipyard/gomobile-ipfs/go => ../go
	github.com/ipfs/go-ipfs => ../go-ipfs
)
```

Save and exit the editor.

Open the terminal and run the command :

```
$ go mod tidy
```

This should now point to our custom repo for the creation of aar file.

Then run the command

```
$ make build_core.android
```

This will now start creating the aar file we require . It starts off by creating a directory named ‘build’ under ‘packages’ .. .

it should go on smoothly here.
But sometimes we need to do one more thing in case we are hit with an error when this command in run.
IN CASE we are hit by errors in the end like ‘gomobile not found’ or ‘gobind not found’ we need to do the following.

> Go to your home directory.
> You will find a ‘go’ folder, open it.
> Go to ‘bin’ and you will find the required files that went missing.
> Copy or move those files to ‘/usr/local/go/bin’

Now retry the `$ make build_core.android`

This time you might reach up to maven publish method , which is written in python . But we are not concerned with that. If we open up the folder of ‘package’ in ‘gomobile-ipfs’ , we will find our core.aar Inside ‘/android/intermediates/core’ …

Precautions/ Cautions:

- the make file will download all these packages independently. And it may take a long time to complete. Personally my best time of completion was 12mins.

- do not give your linux system more than 1 processor. Anything more is crashing the whole system.

- do not try to increase the video memory to more than 32 MB. The VM will take more time to render the screen properly and decreasing the productivity of your system.

- occasionally remove the files that you have downloaded like the .tar files and .zip files.

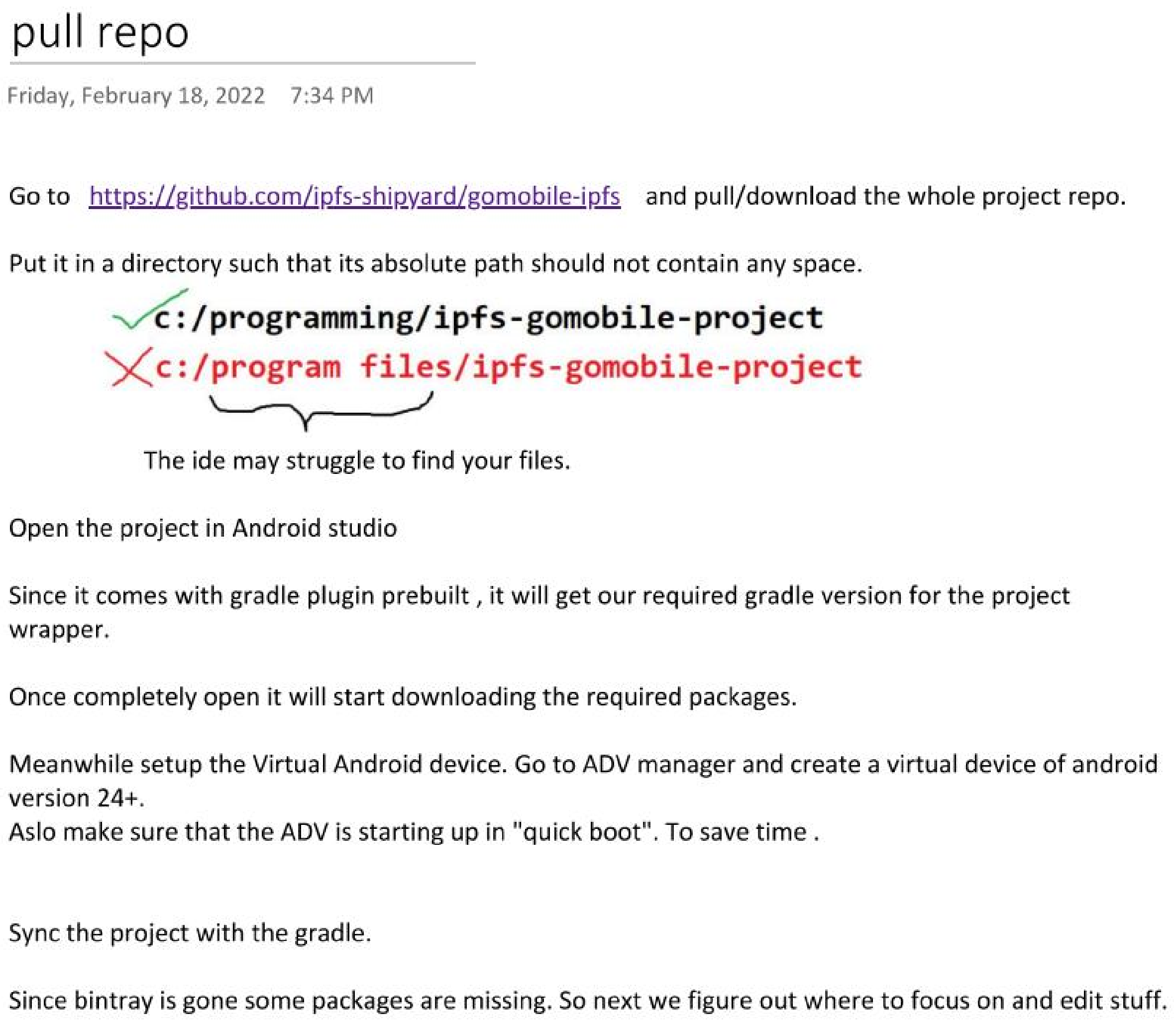
- each attempt of construction of .aar file will bloat up the system space and cache. Remove them , as every time you run the make command a new cache folder is created and cache files are downloaded again.

- the VM may crash a lot so create a VM snapshot at every successful step completed. Additionally it will save your system from cold-boot.

- Snapshots consume a lot of space so remove any previous snapshot before saving up a new one.

- Keep an eye on the VM statistics like cpu load ram and VM exits to find out when our system is going through an error, or infinite loop and when the system has to be stopped.

With the above Portion we have successfully completed the creation of required Library Files.
Now we will use this file inside our ipfs-shipyard-goipfs project



pull repo

Friday, February 18, 2022 7:34 PM

Go to https://github.com/ipfs-shipyard/gomobile-ipfs and pull/download the whole project repo.

Put it in a directory such that its absolute path should not contain any space.

✓ **c:/programming/ipfs-gomobile-project**

✗ **c:/program files/ipfs-gomobile-project**

The ide may struggle to find your files.

Open the project in Android studio

Since it comes with gradle plugin prebuilt , it will get our required gradle version for the project wrapper.

Once completely open it will start downloading the required packages.

Meanwhile setup the Virtual Android device. Go to ADV manager and create a virtual device of android version 24+.
Aslo make sure that the ADV is starting up in "quick boot". To save time .

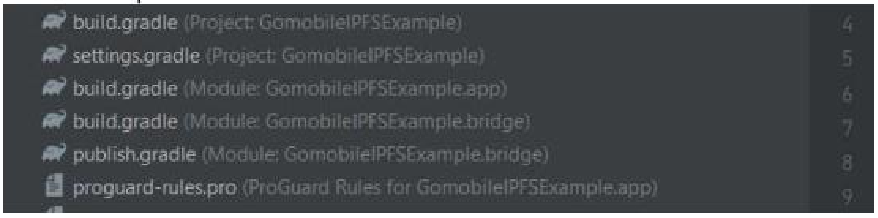
Sync the project with the gradle.

Since bintray is gone some packages are missing. So next we figure out where to focus on and edit stuff.

Working on gradle files

Friday, February 18, 2022 7:56 PM

Most of the pain stake work is done on these files.



build.gradle (Project: GomobileIPFSExample)
settings.gradle (Project: GomobileIPFSExample)
build.gradle (Module: GomobileIPFSExample.app)
build.gradle (Module: GomobileIPFSExample.bridge)
publish.gradle (Module: GomobileIPFSExample.bridge)
proguard-rules.pro (ProGuard Rules for GomobileIPFSExample.app)

First,

build.gradle (Project: GomobileIPFSExample)

Comment out the task :> `task setupAllProjectsRepos`
Since we are going to mention the priority dependencies somewhere else.

Second,

settings.gradle (Project: GomobileIPFSExample)

Add this code above everything else.

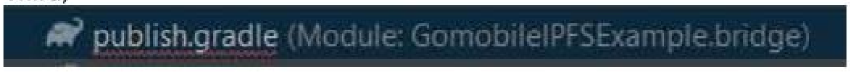
```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.PREFER_SETTINGS)
    repositories {
        google()
        mavenCentral()
        jcenter()
        flatDir {
            dirs'libs'
        }
        // jcenter( { url "https://jcenter.bintray.com/" } )
    }
}
```

this will directly be referred by the whole project as a preffered setting.

```
flatDir {
    dirs'libs'
}
```

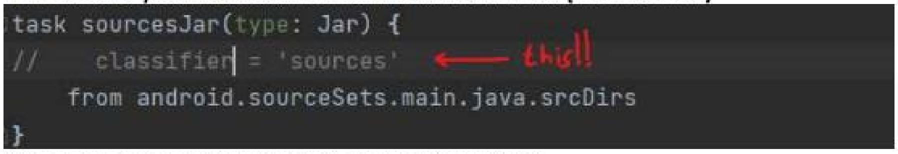
Is a little special as it is a custom folder where core package is kept. Will talk bout that in next page.

Third,



publish.gradle (Module: GomobileIPFSExample.bridge)

Where ever you find classifier comment them out **(*OPTIONAL)**



```
task sourcesJar(type: Jar) {
//     classifier = 'sources'
    from android.sourceSets.main.java.srcDirs
}
```

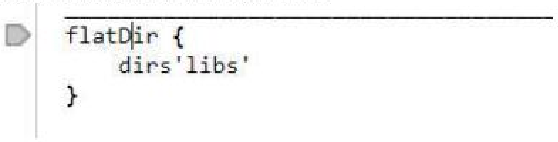
These are not compatible with my version of java.

Lastly, we have something to do with the bridge's **`build.gradle`**
But will inform you about it in the next page. After solving the core problem.

Now we will try to put this archive (aar file) to use …

To do that,

First go to this place : **%PROJECT_DIR%\gomobile-ipfs-master\android**
Create a new folder and name it 'libs'.



```
flatDir {
    dirs'libs'
}
```

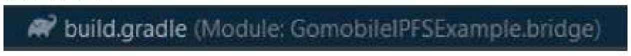
This code in the prevous page in **settings.gradle** refers to that directory.

Next, put that downloaded **.aar** file in that location.

To tell gradle to directly access this file as an Archive we need to edit the **build.gradle** file in **:bridge** directory . (i.e. **%PROJECT_DIR%\gomobile-ipfs-master\android\bridge**)

We can rename the **.arr** file to whatever name we want for ease of referencing. But it is suggested to leave it as it is. In my case I've renamed it to just "core".

So in the file



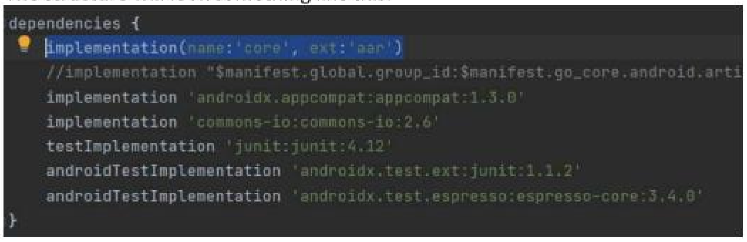
build.gradle (Module: GomobileIPFSExample.bridge)

Go to **dependencies** section.

Comment out this line
implementation "$manifest.global.group_id:$manifest.go_core.android.artifact_id:$rootProject.ext.version"
This is because it may try to download the core pack from bintray. Which will then cause us problems.

Instead add this line.
implementation(name:'core', ext:'aar')
Note: 'core' is the name of the **.aar** file I have downloaded and renamed. if not renamed, the string here should be "core-1.2.1-gateway.aar"

The structure will look something like this.



```
dependencies {
    implementation(name:'core', ext:'aar')
    //implementation "$manifest.global.group_id:$manifest.go_core.android.arti
    implementation 'androidx.appcompat:appcompat:1.3.0'
    implementation 'commons-io:commons-io:2.6'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

This will contain all the required functions and libraries that we will use in our project.

So we don't need to touch the internal Java code unlike we did last time.

Unless we are tapering with it for creation of P2P only network we will leave the internal code just as it is. We can configure our AVD and run our app…

Building the app

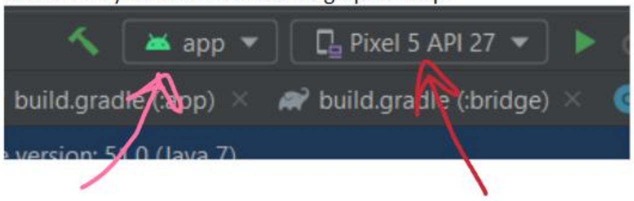
Friday, February 18, 2022 9:32 PM

With This I Think that we have completed all the required steps for building the application

Now we can Sync with Gradle.

Once the Sync is finished we are now ready to build the App.

Make sure you have the following options up

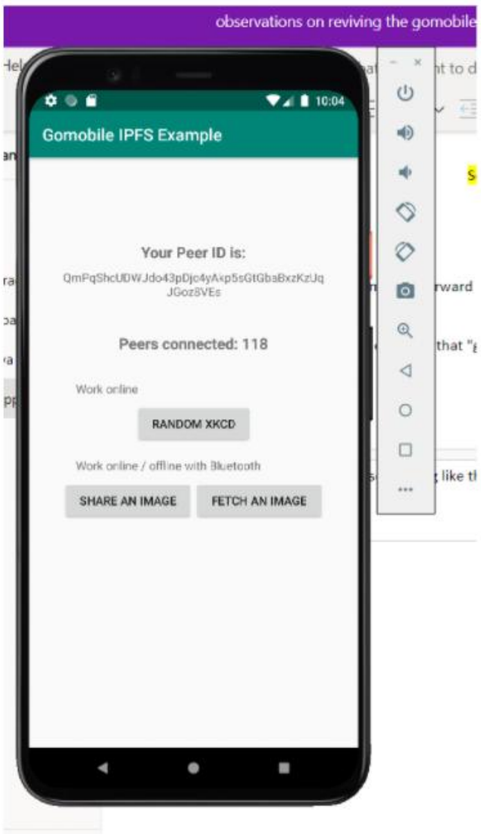


To run the app Select the virtual device

Before moving forward . Clean and Rebuild the project just to be sure...

Well Run the project using that "green Arrow (play)" to build and run the project.

The final output should be something like this...



observations on reviving the gomobile
10:04
Gomobile IPFS Example
Your Peer ID is:
QmPqShcUDWJdo43pDjc4yAkp5sGtGbaBxzKzUq JGoz8VEs
Peers connected: 118
Work online
RANDOM XKCD
Work online / offline with Bluetooth
SHARE AN IMAGE
FETCH AN IMAGE