RUNNING GO ON ANDROID

Running indirectly…

The Go Mobile project is marked as “experimental” and I guess it will take a couple of years until we’ll get some stable version.

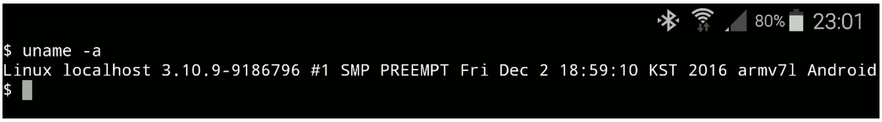
The way I’m going to show you how to run Go programs on Android is a bit tricky as it actually uses Termux which is an Android terminal emulator and Linux environment app. Technically we could install and run there many other languages, but let’s see how it’s getting on with Go.

What makes **Termux** really interesting is the API we can use to access the Android API. For instance we can check our battery status, take a photo or check our device GPS location.

Before we continue, install these two apps on your Android phone:

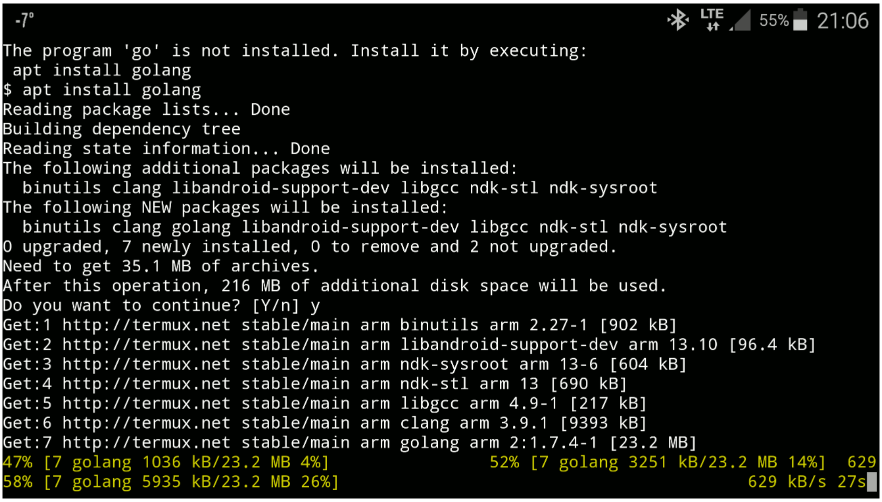
- Termux
- Termux API

That’s what I see when typed **uname -a**



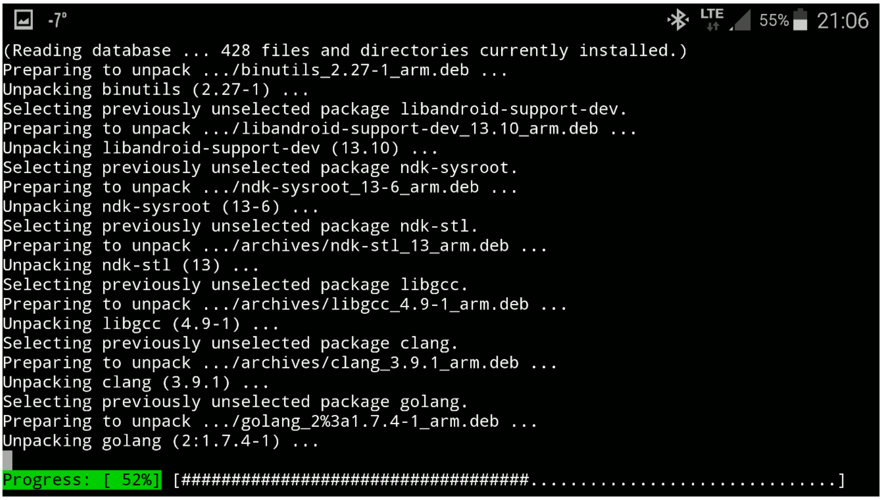
```
$ uname -a
Linux localhost 3.10.9-9186796 #1 SMP PREEMPT Fri Dec 2 18:59:10 KST 2016 armv7l Android
$
```

Installing Go on Termux is as easy as firing up **apt install golang**



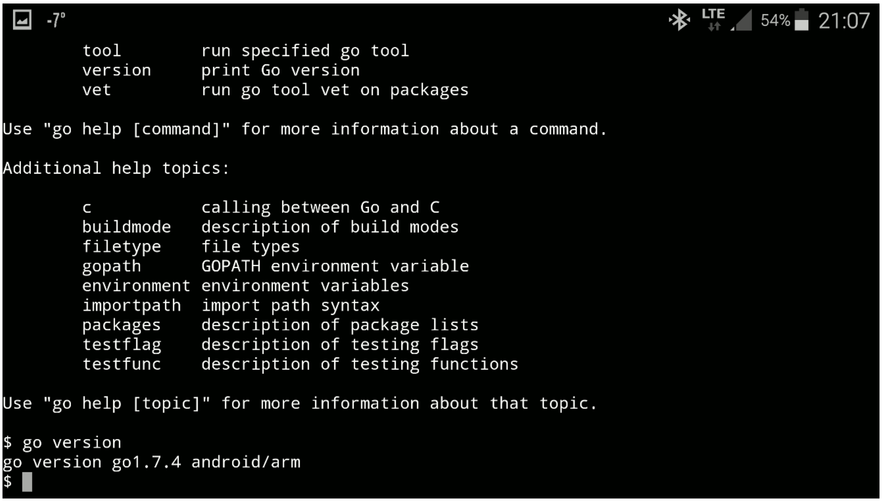
```
The program 'go' is not installed. Install it by executing:
 apt install golang
$ apt install golang
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils clang libandroid-support-dev libgcc ndk-stl ndk-sysroot
The following NEW packages will be installed:
  binutils clang golang libandroid-support-dev libgcc ndk-stl ndk-sysroot
0 upgraded, 7 newly installed, 0 to remove and 2 not upgraded.
Need to get 35.1 MB of archives.
After this operation, 216 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://termux.net stable/main arm binutils arm 2.27-1 [902 kB]
Get:2 http://termux.net stable/main arm libandroid-support-dev arm 13.10 [96.4 kB]
Get:3 http://termux.net stable/main arm ndk-sysroot arm 13-6 [604 kB]
Get:4 http://termux.net stable/main arm ndk-stl arm 13 [690 kB]
Get:5 http://termux.net stable/main arm libgcc arm 4.9-1 [217 kB]
Get:6 http://termux.net stable/main arm clang arm 3.9.1 [9393 kB]
Get:7 http://termux.net stable/main arm golang arm 2:1.7.4-1 [23.2 MB]
47% [7 golang 1036 kB/23.2 MB 4%]                  52% [7 golang 3251 kB/23.2 MB 14%]  629
58% [7 golang 5935 kB/23.2 MB 26%]                                            629 kB/s 27s
```

It takes 216 MB of additional disk space. The process shows exactly what packages are downloaded and installed. You can see the **deb** format is used.



```
(Reading database ... 428 files and directories currently installed.)
Preparing to unpack .../binutils_2.27-1_arm.deb ...
Unpacking binutils (2.27-1) ...
Selecting previously unselected package libandroid-support-dev.
Preparing to unpack .../libandroid-support-dev_13.10_arm.deb ...
Unpacking libandroid-support-dev (13.10) ...
Selecting previously unselected package ndk-sysroot.
Preparing to unpack .../ndk-sysroot_13-6_arm.deb ...
Unpacking ndk-sysroot (13-6) ...
Selecting previously unselected package ndk-stl.
Preparing to unpack .../archives/ndk-stl_13_arm.deb ...
Unpacking ndk-stl (13) ...
Selecting previously unselected package libgcc.
Preparing to unpack .../archives/libgcc_4.9-1_arm.deb ...
Unpacking libgcc (4.9-1) ...
Selecting previously unselected package clang.
Preparing to unpack .../archives/clang_3.9.1_arm.deb ...
Unpacking clang (3.9.1) ...
Selecting previously unselected package golang.
Preparing to unpack .../golang_2%3a1.7.4-1_arm.deb ...
Unpacking golang (2:1.7.4-1) ...

Progress: [ 52%] [###################################.............................]
```

Let’s type **go version**



```
        tool        run specified go tool
        version     print Go version
        vet         run go tool vet on packages

Use "go help [command]" for more information about a command.

Additional help topics:

        c           calling between Go and C
        buildmode   description of build modes
        filetype    file types
        gopath      GOPATH environment variable
        environment environment variables
        importpath  import path syntax
        packages    description of package lists
        testflag    description of testing flags
        testfunc    description of testing functions

Use "go help [topic]" for more information about that topic.

$ go version
go version go1.7.4 android/arm
$
```

Nice! isn’t? :) go is installed in:

```
    $ which go
    /data/data/com.termux/filex/usr/bin/go
```

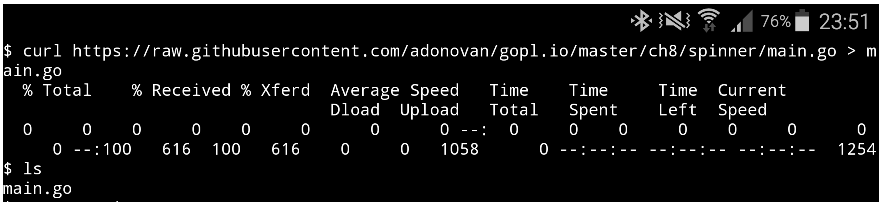
Having **vi** we can type our program directly at the console ;) ..give it a try and in the meantime I will download an example using …curl (we'll use **go get** later on)

Type **apt install curl** and once it's installed, run this:

```
    $ curl https://raw.githubusercontent.com/ad
onovan/gopl.io/master/ch8/spinner/main.go > mai
n.go
```

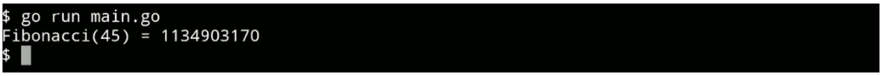
This is an example from "The Go programming Language" book by Alan A. A. Donovan & Brian W. Kernighan.. The code computes the 45th Fibonnaci number.

Saving the curl result into main.go file:



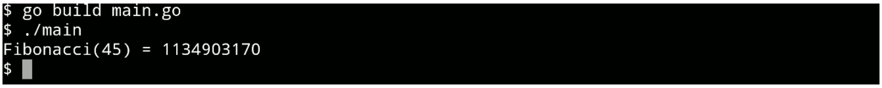
```
$ curl https://raw.githubusercontent.com/adonovan/gopl.io/master/ch8/spinner/main.go > m
ain.go
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:  0     0    0     0    0     0     0
     0 --:100   616  100   616    0     0   1058      0 --:--:-- --:--:-- --:--:--  1254
$ ls
main.go
```

and running it **go run main.go** works!



```
$ go run main.go
Fibonacci(45) = 1134903170
$ 
```

what about compiling? **go build main.go**


```
$ go build main.go
$ ./main
Fibonacci(45) = 1134903170
$
```

works!

If you used go before you know that go supports downloading dependencies/packages by running **go get**. Firstly we need to set up **$GOPATH.**

```
$ echo $HOME
/data/data/com.termux/files/home
$ mkdir $HOME/mygo
$ export GOPATH=$HOME/mygo
$ echo $GOPATH
/data/data/com.termux/files/home/mygo
```

(it’s worth to save GOPATH in your **.bashrc**)

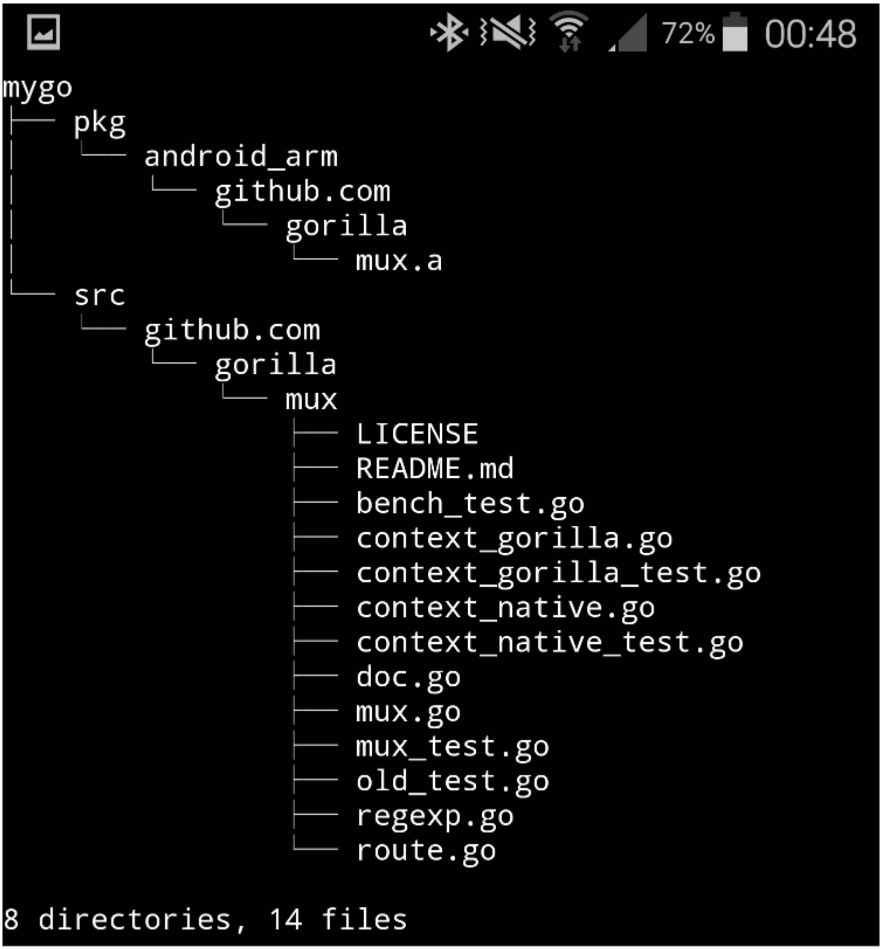
and install **git** by running:

```
$ apt install git
```

Now, run let’s test go get by downloading a popular Go toolkit **gorilla**:

```
$ go get github.com/gorilla/mux
```

tree mygo shows:



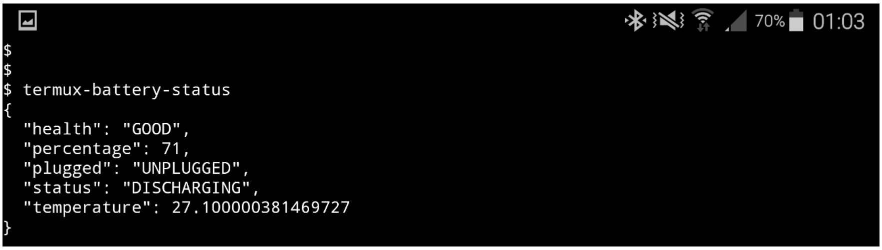
```
mygo
├── pkg
│   └── android_arm
│       └── github.com
│           └── gorilla
│               └── mux.a
└── src
    └── github.com
        └── gorilla
            └── mux
                ├── LICENSE
                ├── README.md
                ├── bench_test.go
                ├── context_gorilla.go
                ├── context_gorilla_test.go
                ├── context_native.go
                ├── context_native_test.go
                ├── doc.go
                ├── mux.go
                ├── mux_test.go
                ├── old_test.go
                ├── regexp.go
                └── route.go

8 directories, 14 files
```

Termux API

So far, so good. The power of Termux lies in the API.

Get familiar with the available API here.

For instance, run **termux-battery-status**



```
$
$
$ termux-battery-status
{
  "health": "GOOD",
  "percentage": 71,
  "plugged": "UNPLUGGED",
  "status": "DISCHARGING",
  "temperature": 27.100000381469727
}
```

The API's result is returned as JSON! This is absolutely brilliant!

If you want to upload and run your own Go programs under Termux you should find a way to get your code there.

Run this command to access storage folders on your mobile phone:

```
$ termux-setup-storage
```

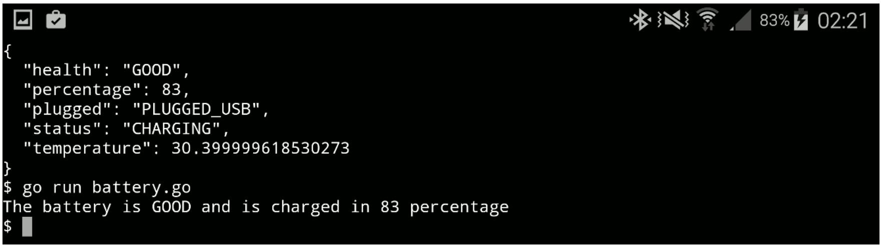
Grand permissions to acccess folders.

Now run **ls** to see available folders. You should see more folders available. You can use usb to access files on your phone but I won't describe it here (you know how to do it, don't you? ;))

Ok, let's build a go program that is returning the current status of the phone's battery:

```go
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "os/exec"
)

var Battery struct {
    Health      string
    Percentage  int
    Status      string
    Temperature float64
}

func main() {
    cmd := exec.Command("termux-battery-status"
)

    //StdoutPipe returns a pipe that will be co
nnected to
    //the command's standard output when the co
mmand starts
    stdout, err := cmd.StdoutPipe()
    if err != nil {
        log.Fatal(err)
    }
    if err := cmd.Start(); err != nil {
        log.Fatal(err)
    }
    if err := json.NewDecoder(stdout).Decode(&B
attery); err != nil {
        log.Fatal(err)
    }

    //Wait waits for the command to exit
    //It must have been started by Start
    if err := cmd.Wait(); err != nil {
        log.Fatal(err)
    }
    fmt.Printf("The battery is %s and is charge
d in %d percentage\n",
        Battery.Health, Battery.Percentage)
}
```

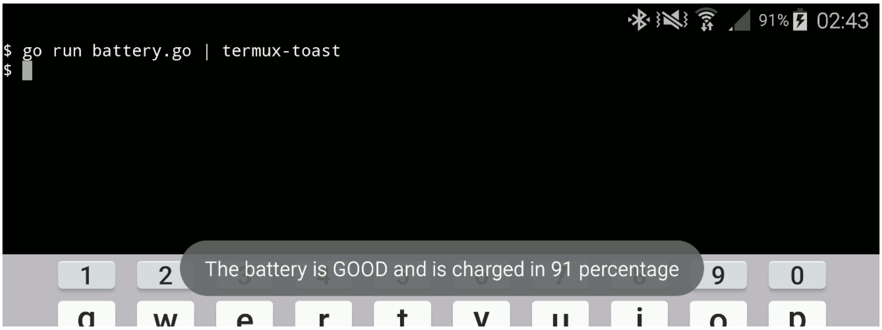
…and running it shows:



```
{
  "health": "GOOD",
  "percentage": 83,
  "plugged": "PLUGGED_USB",
  "status": "CHARGING",
  "temperature": 30.399999618530273
}
$ go run battery.go
The battery is GOOD and is charged in 83 percentage
$
```

I put the code to GitHub, so you can curl it from there :) :
https://github.com/rafalgolarz/termux-go

How about displaying the result in a Toast (a transient popup)?



```
$ go run battery.go | termux-toast
$
```

The battery is GOOD and is charged in 91 percentage

More ideas

The API allows sending SMS’, taking photos, speak text with a system text-to-speech (TTS) engine, and much more. I can imagine a background process (**$ ./yourgoprogram &**) that takes pictures periodically and sends them to a contact of your list. This would act as a security camera.