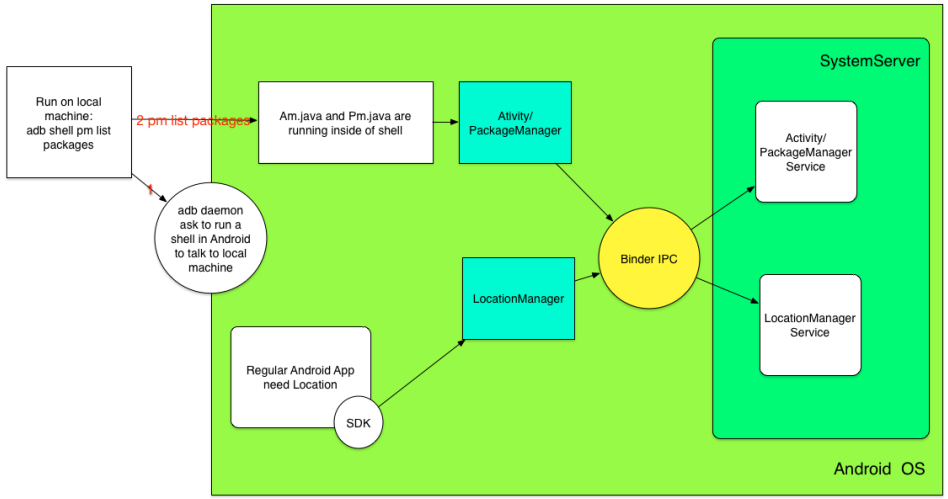
Efficiency and fun from using ADB Shell, Part 3 - am, dumpsys, android system properties

As we see from part 1 and 2, adb shell has tones of interesting commands, let's figure out how this all works under the hood.



When we run our adb shell, adb daemon asks to run shell in Android and then command in this case `pm list packages` passed there. This command handled in class Pm.java and called in PackageManager which via Binder inter-process communication mechanism invoke required function in PackageManagerService. Similar process happening when our app wants for example get a location from LocationManager. System services such as LocationManagerService and ActivityManagerService loads at boot time via SystemServer We can see that adb and app communicate quite similar, and there is no magic.

Now we will see how to get info about running service, check if service is running, how to start components and other useful commands related to am, dumpsys.

Dumpsys

Dumpsys is very powerful Android tool which runs on device and can dump information about system services plus it provides a possibility to communicate/set property if it was defined by service. In the end, we can get as much power and cool information as the system service provide.

The basic usage could be to check does the service running at all:

```
adb shell dumpsys activity services our.package.name.OurService
</pre>
```

So what does it mean? First let's see list of available system services:

```

adb shell dumpsys -l
Currently running services:
  DockObserver
  SurfaceFlinger
  accessibility
  account
  activity
  alarm
  android.security.keystore
  appops
  appwidget
  assetatlas
  audio
  backup
  battery
    ...
</pre>
```

We will get pretty long list of services depending of our device and version of android. Then to see what each service could do for us run:

```
adb shell dumpsys activity -h
Activity manager dump options:
  [-a] [-c] [-p package] [-h] [cmd] ...
  cmd may be one of:
    a[ctivities]: activity stack state
    r[recents]: recent activities state
    b[roadcasts] [PACKAGE_NAME] [history [-s]]: broadcast state
    i[ntents] [PACKAGE_NAME]: pending intent state
    p[rocesses] [PACKAGE_NAME]: process state
    o[om]: out of memory management
    perm[issions]: URI permission grant state
    prov[iders] [COMP_SPEC ...]: content provider state
    provider [COMP_SPEC]: provider client-side state
```

```
    s[ervices] [COMP_SPEC ...]: service state
    as[sociations]: tracked app associations
    service [COMP_SPEC]: service client-side state
    package [PACKAGE_NAME]: all state related to given package
    all: dump all activities
    top: dump the top activity
    write: write all pending state to storage
    track-associations: enable association tracking
    untrack-associations: disable and clear association tracking
  cmd may also be a COMP_SPEC to dump activities.
  COMP_SPEC may be a component name (com.foo/.myApp),
    a partial substring in a component name, a
    hex object identifier.
  -a: include all available server state.
  -c: include client state.
  -p: limit output to given package.
</pre>
```

We looking for information about service, `s[ervices] [COMP_SPEC ...]: service state` what we need. Finally putting it all together we getting info

One of the useful system service commands is `meminfo` which gives us info about memory management of app.

```
adb shell dumpsys meminfo com.viber.voip
Applications Memory Usage (kB):
Uptime: 49789130 Realtime: 64735184

** MEMINFO in pid 18506 [com.viber.voip] **
                   Pss  Private  Private  Swapped     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc
                ------   ------   ------   ------   ------   ------
  Native Heap    11598    11568        0        0    39168    36497
  Dalvik Heap    16736    16668        0        0    44293    33676
 Dalvik Other     1896     1896        0        0
        Stack      580      580        0        0
       Ashmem     1102     1100        0        0
     Gfx dev      5502     5120        0        0
   Other dev         8        0        8        0
    .so mmap      6358      624     4420        0
   .apk mmap       803        0      272        0
   .ttf mmap       142        0       80        0
   .dex mmap     14044        8    14036        0
   .oat mmap      2978        0     1112        0
   .art mmap      1943     1488       12        0
  Other mmap        99       12       32        0
  EGL mtrack     41280    41280        0        0
```

```
      Unknown      640      640        0        0
        TOTAL   105709    80984    19972        0    83461    70173

 App Summary
                       Pss(KB)
                        ------
           Java Heap:    18168
         Native Heap:    11568
                Code:    20552
               Stack:      580
            Graphics:    46400
       Private Other:     3688
              System:     4753

               TOTAL:   105709       TOTAL SWAP (KB):        0

 Objects
               Views:      321         ViewRootImpl:        1
         AppContexts:        5           Activities:        1
              Assets:        4        AssetManagers:        2
       Local Binders:       58        Proxy Binders:       32
       Parcel memory:       19         Parcel count:       78
    Death Recipients:        0      OpenSSL Sockets:        1

 SQL
         MEMORY_USED:        0
  PAGECACHE_OVERFLOW:        0          MALLOC_SIZE:       62
</pre>
```

Looks like very big area to play with :) Btw if we want to know where in [Android source code](https://android.googlesource.com/platform/framew

```
AndroidSource/src/services/core/java/com/android/server/am/ActivityMa
 14719                 packages = true;
 14720             } else if ("-h".equals(opt)) {
 14721:                pw.println("meminfo dump options: [-a] [-d] [
 14722                 pw.println("  -a: include all available infor
 14723                 pw.println("  -d: include dalvik details.");
</pre>
```

That applicable for any system service let's say if anything possible to do with BatteryService:

```
adb shell dumpsys battery -h
Dump current battery state, or:
```

```
  set [ac|usb|wireless|status|level|invalid]
  unplug
  reset
</pre></code>
Thanks for inspiration Roman Mazur we can do a lot of cool [tricks](h

## Am

There is a lot of cases when we need to test does our application har

am start -a "android.intent.action.VIEW" -d "http://our.website.com"
am start -a "android.intent.action.SEND" --es "android.intent.extra.T
//or see geo
adb shell am start -a android.intent.action.VIEW -d "geo:64.873799766
</pre>

Some of the Intent commands we can specify. We can find more in help.


 specifications include these flags and arguments:
    [-a ] [-d ] [-t ]
    [-c  [-c ] ...]
    [-e|--es   ...]
    [--esn  ...]
    [--ez   ...]
    [--ei   ...]
    [--el   ...]
    [--ef   ...]
    [--eu   ...]
</pre></code>

We can find more actions in class [Intent](https://developer.android.
Am gives us possibility to launch our own components Activities/Servi


adb shell am start -n "our.application.id/our.package.name.OurActivit
</pre>

Starting service is quite similar also we could specify additional in

adb shell am startservice -n "ar_g.blog.am/ar_g.blog.am.OurService" -
//and in service just handle intent as we wish
@Override public int onStartCommand(Intent intent, int flags, int sta
    if (intent != null) {
      String name = intent.getStringExtra("action");
      if (name.equals("kill")) {
```

```
        stopSelf();
      } else {
        throw new UnsupportedOperationException("Action not supported
      }
    }
    return super.onStartCommand(intent, flags, startId);
  }
</pre>

Sending broadcast we just use our broadcast action. Even better examp


adb shell am broadcast -a "our.specified.action"
</pre>

Or even better **restart** Android:


adb shell am broadcast -a android.intent.action.BOOT_COMPLETED
</pre>

## Android system properties

Very often during development process occur the necessity to change c


adb shell getprop
</pre>

My favourite one for debugging UI performance, [here](https://androi

For example to see does the frame rate of our app is acceptable:

adb shell setprop debug.hwui.profile visual_bars
//to set default
adb shell setprop debug.hwui.profile false
</pre>

## Other command line tools

We can find other command line tools [here](https://android.googlesou

//disable/enable wifi
adb shell "svc wifi disable"
</pre>
```

```
## Conclusion

That is not all of the tips which could make developing and testing e
```

Written on January 5, 2016