# P6

Ian Davis

U00723728

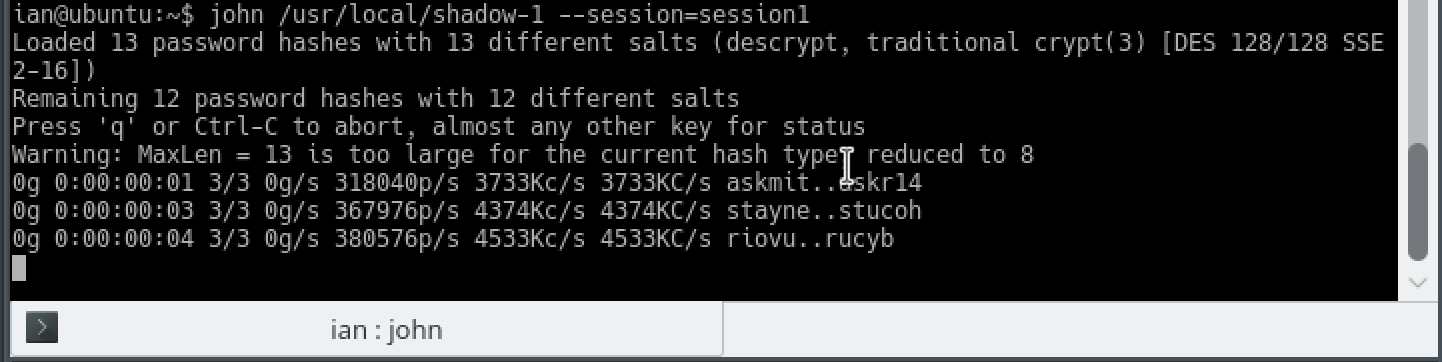
Github: <https://github.com/IanDavis1995/P6>

# Task 1

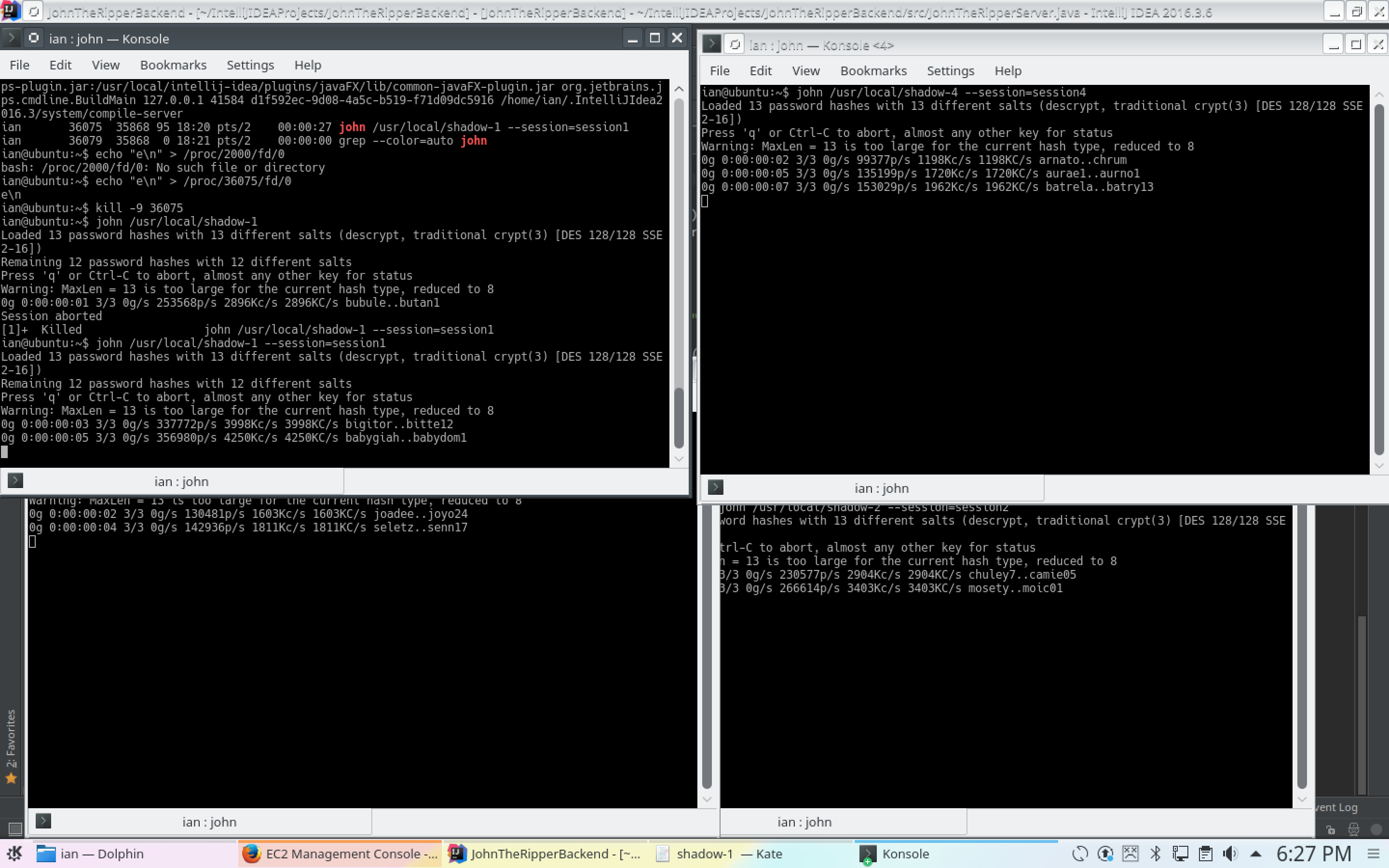
* Screenshots pg. 1-7
* Solo/Multi-Process comparison pg. 7
* Status Report pg. 8
* Experience Report pg. 9

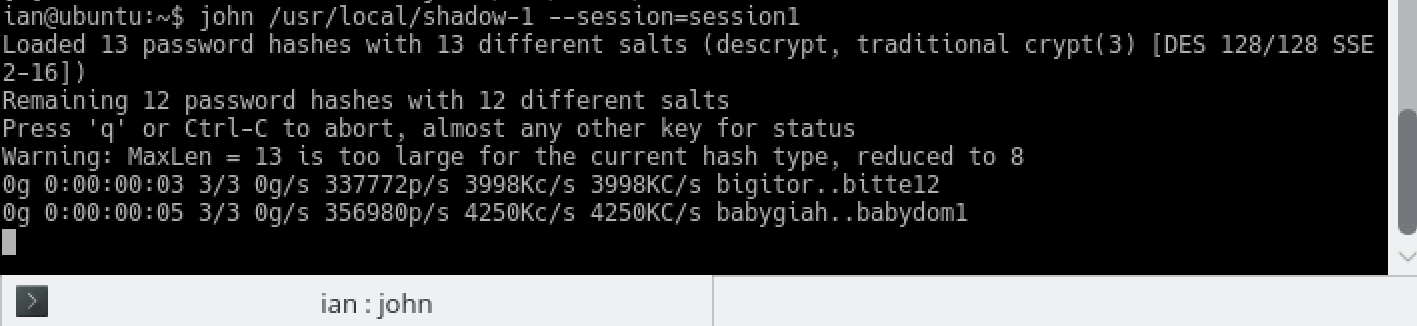
## Ubuntu-Only

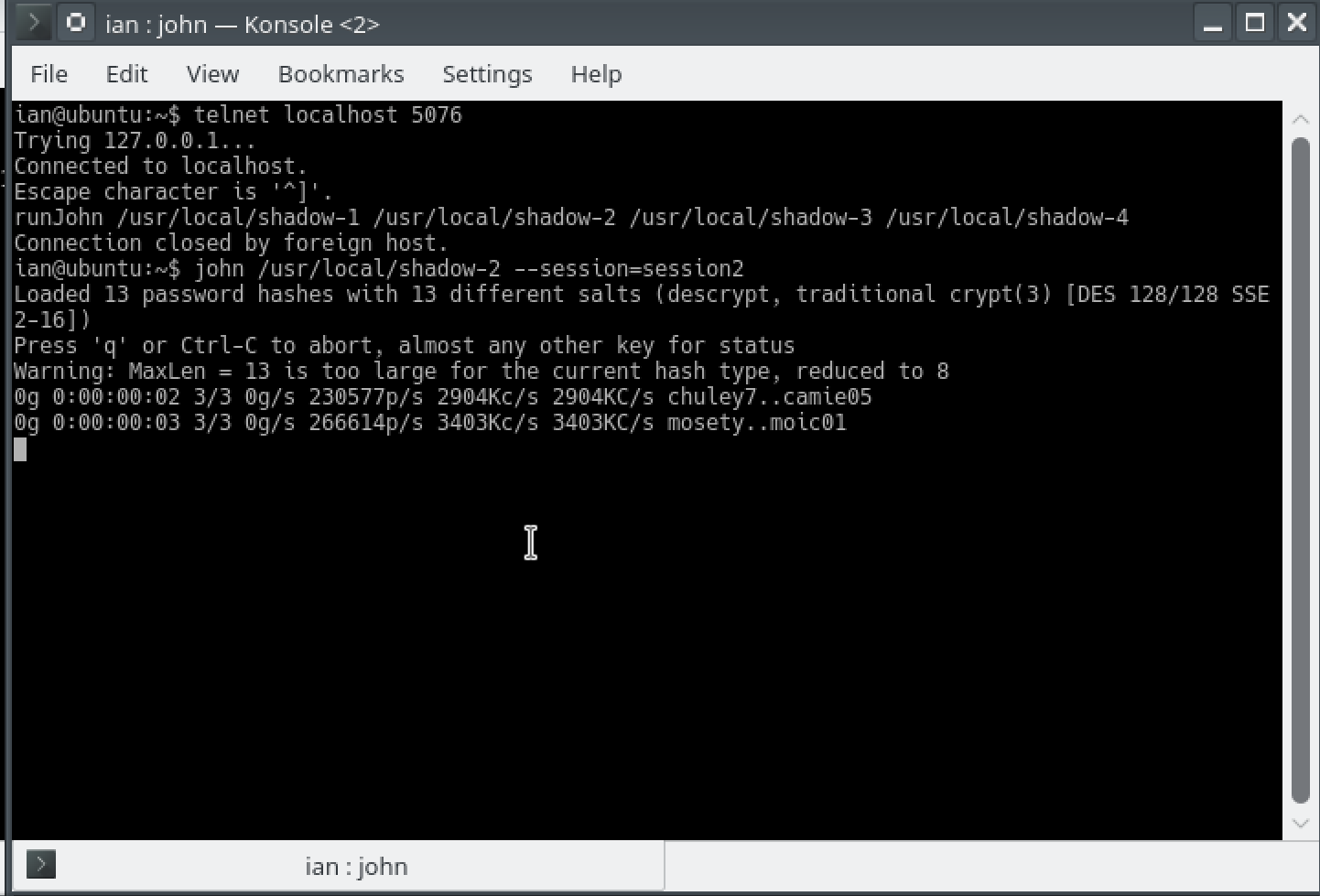
### Solo

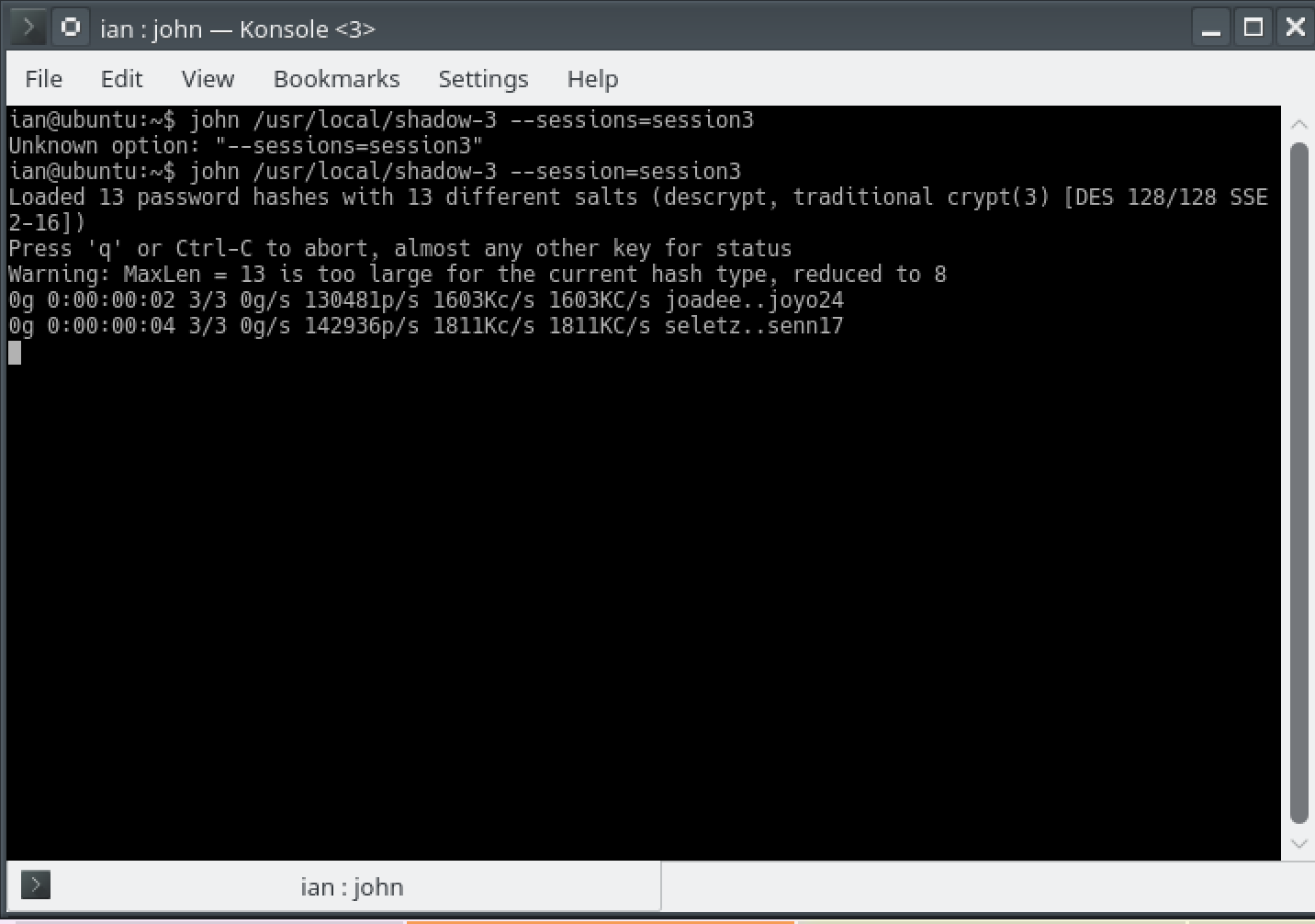


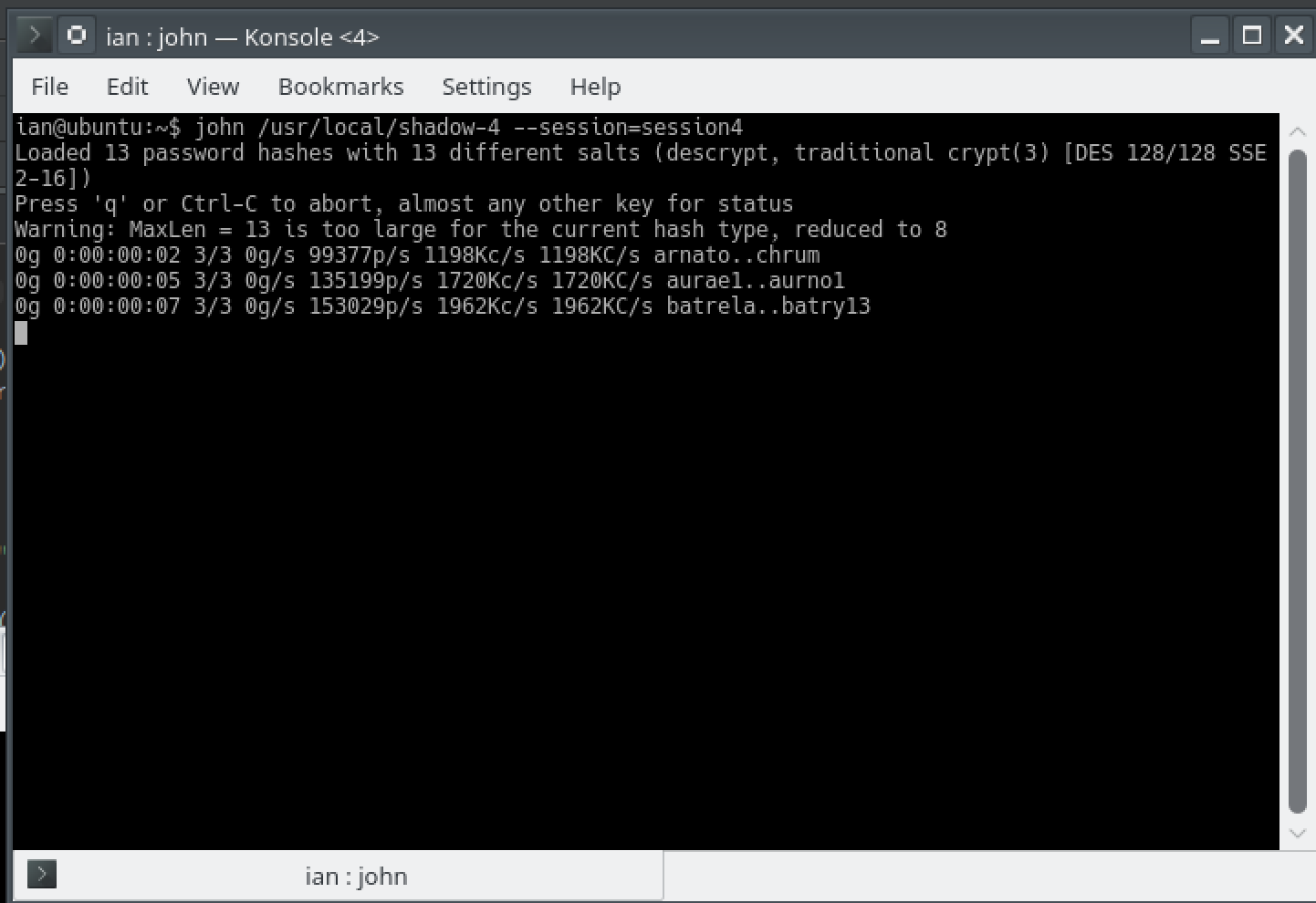
### Multi-processing

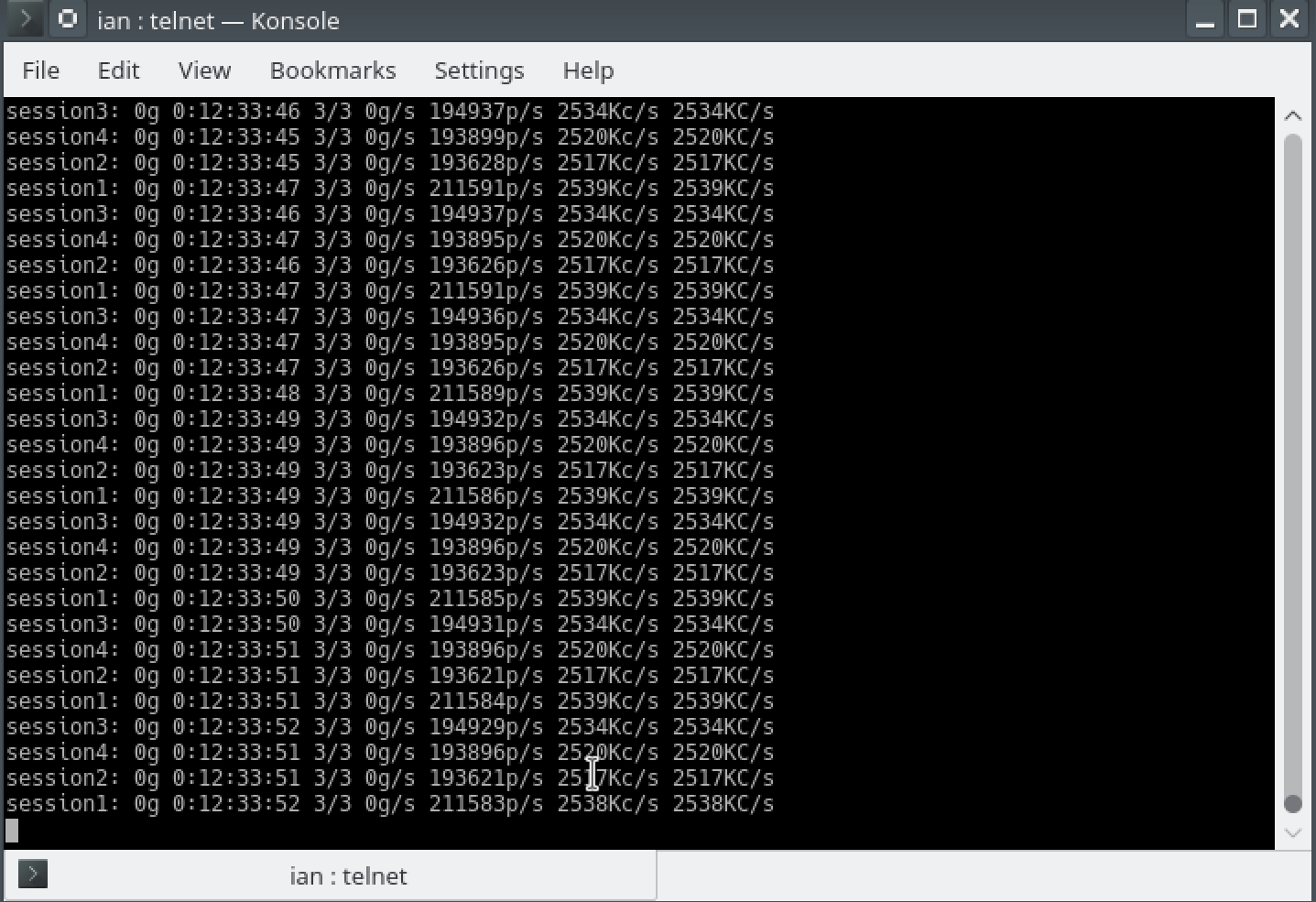






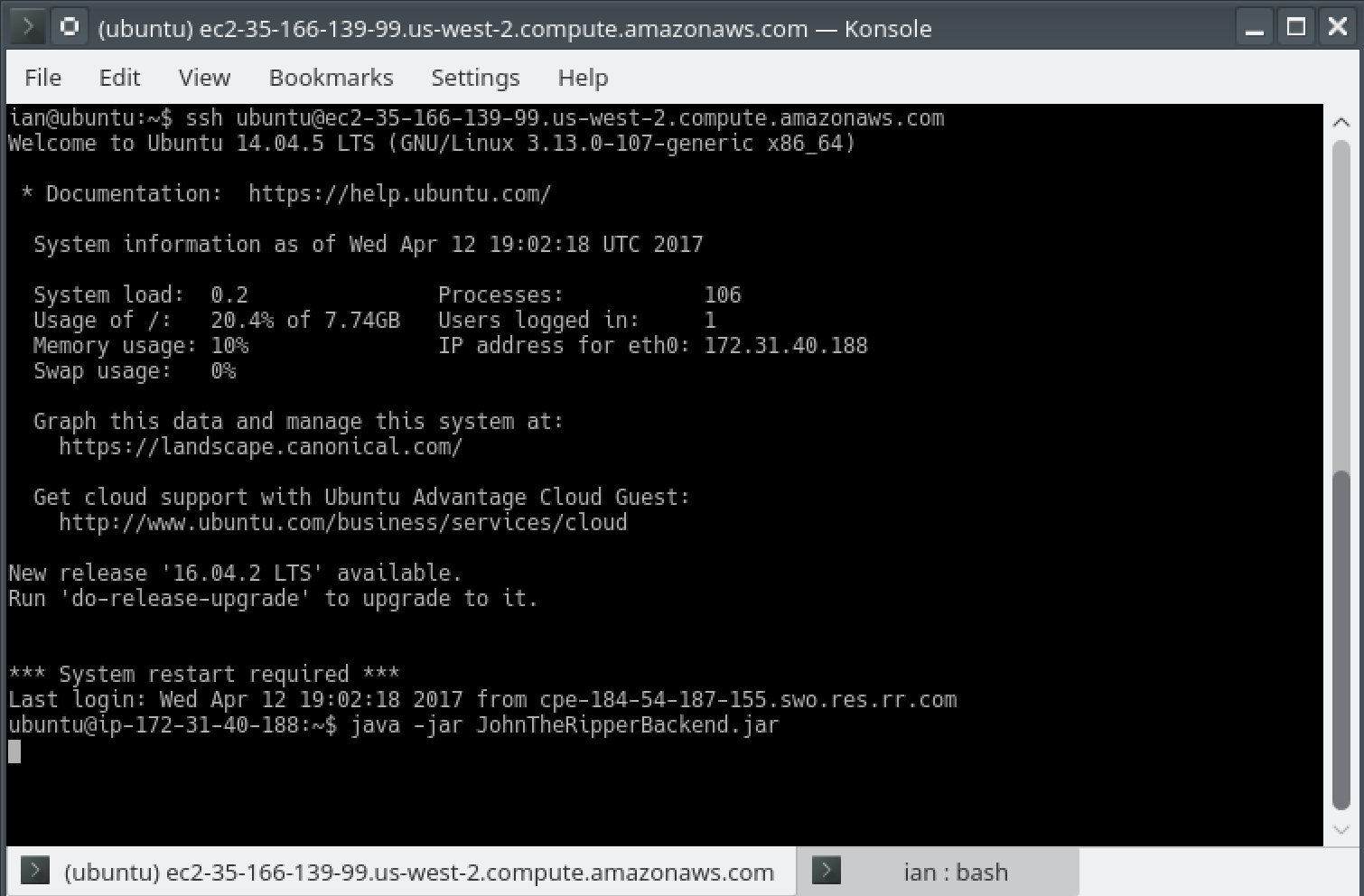
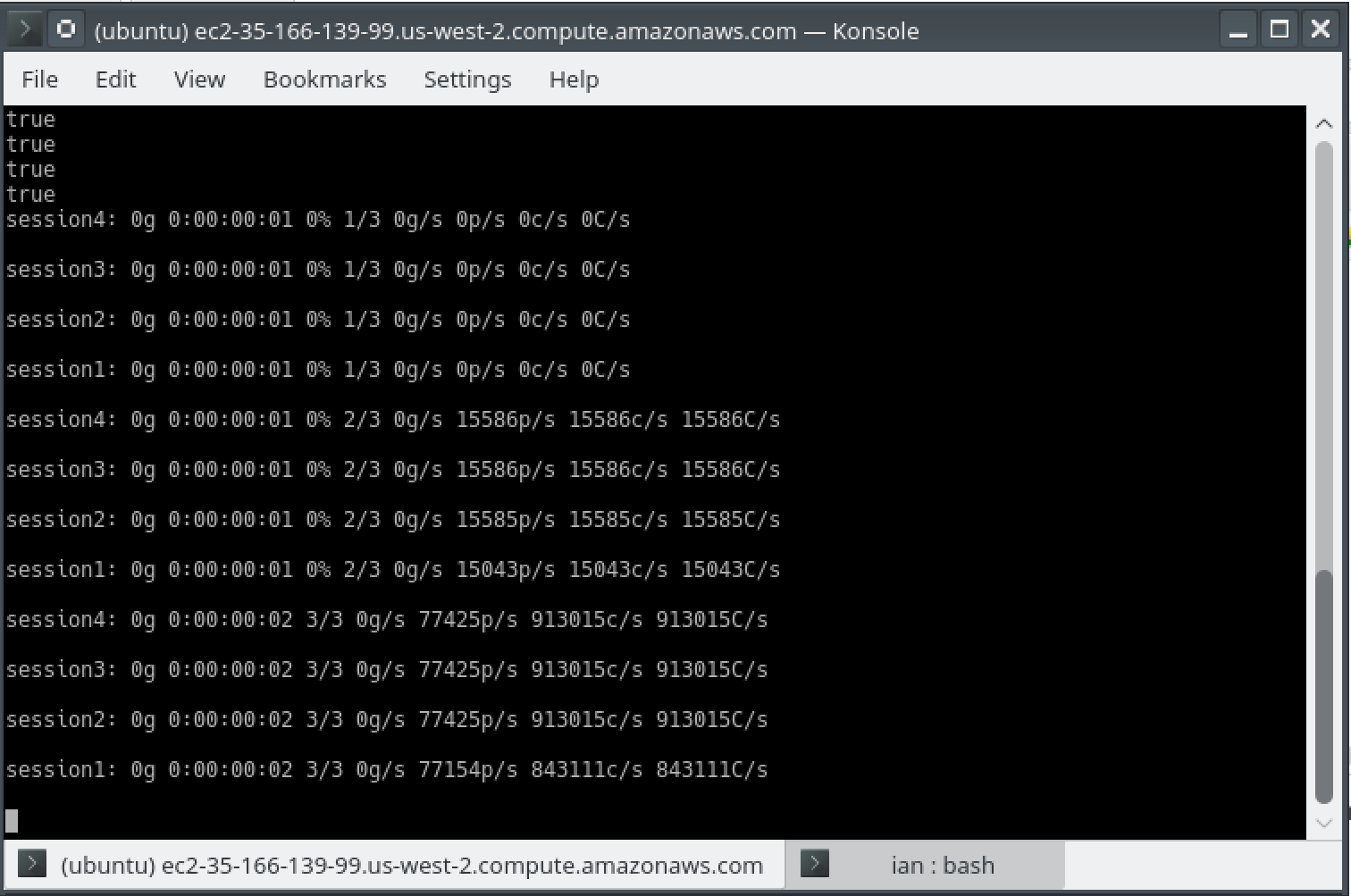




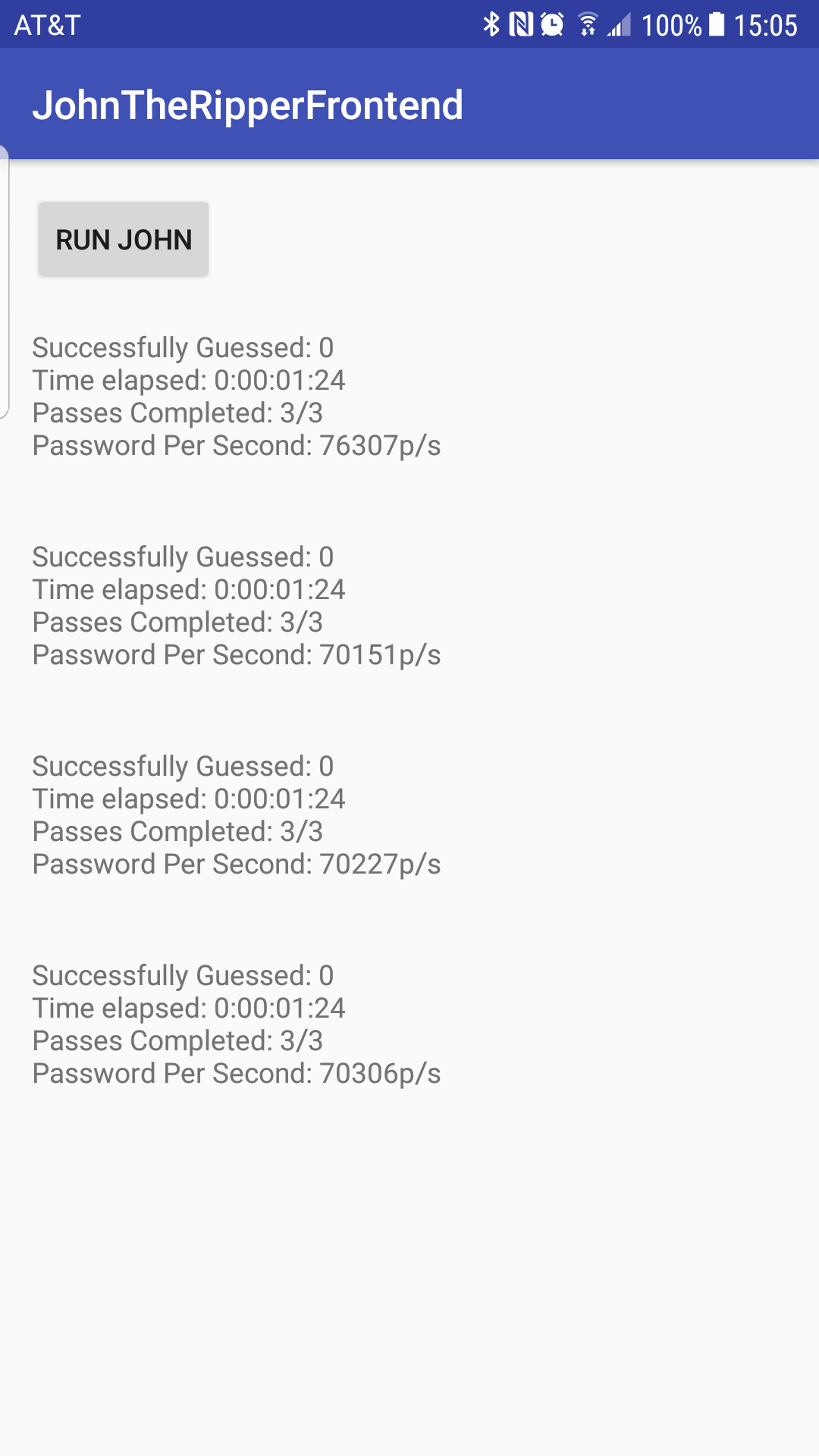
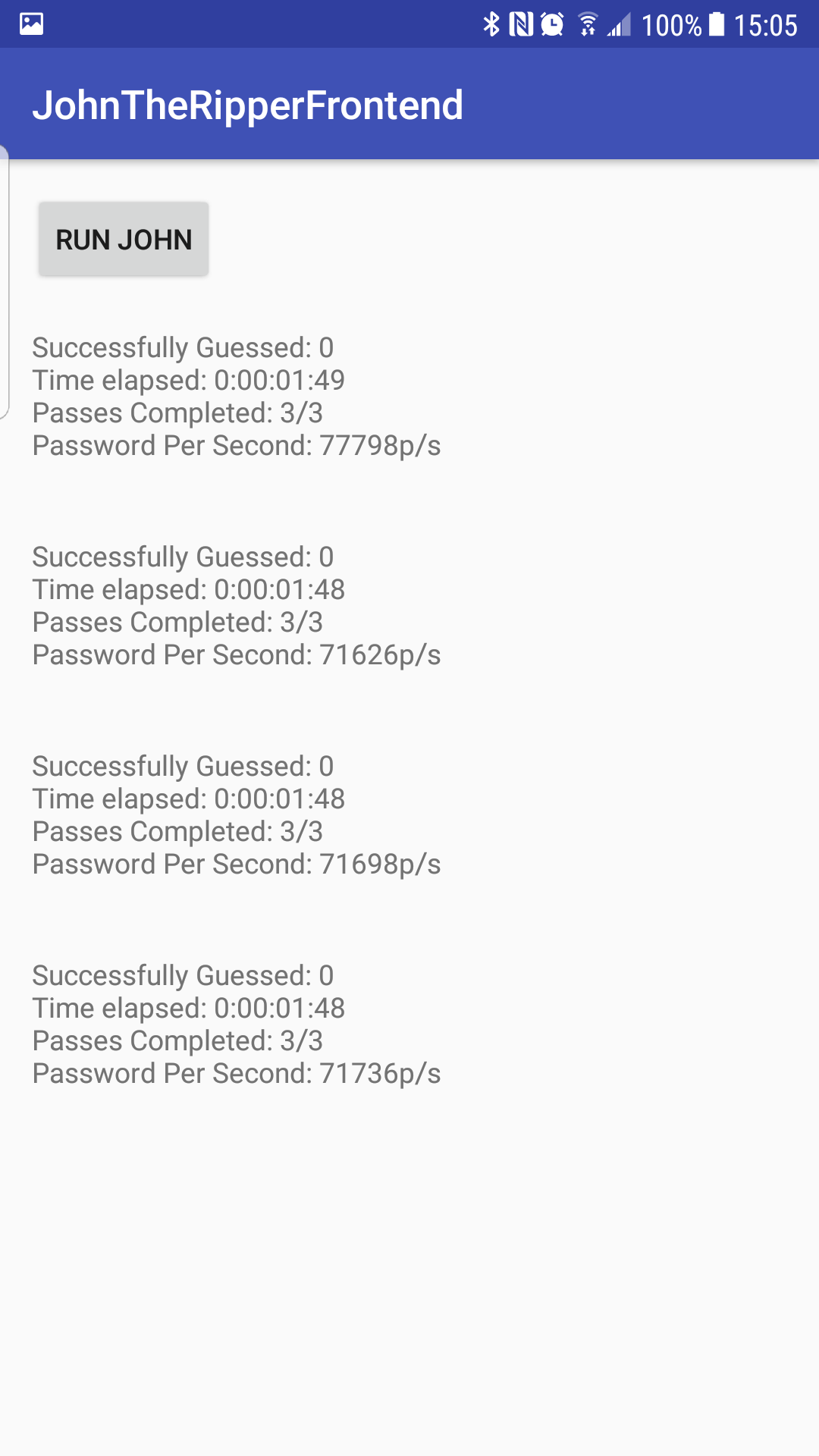


## Cloud

## ../../../../Desktop/P6-screens/amazon-aws-ssh.pn

## Android

## Solo/Multi-process Comparison

Running john as both a solo and a multi-process ran for 8+ hours and had not completed, so the elapsed time between the two did not seem to be “measurably slower”. However, john reports statistics about how many passwords are being checked per second, and all 4 of the simultaneous instances reported the same passwords per second as the solo instance, so this leads me to believe that running 4 simultaneous instances would indeed be 4 times faster, but whether it’s 4 times faster than a week, month, or year is uncertain. Running the process in the cloud (Amazon EC2 instance) seemed to run surprisingly slower than on my laptop. I’m not sure if this is due to free-tier processing limits of AWS or something else. For comparison, john reported about 300,000 passwords/second/process on my laptop, but only about 198,000 passwords/second/process on the EC2 instance.

## Status Report

Getting john up and running from the command line and the cloud was simple, however I was unsuccessful in getting them to complete even after 8+ hours on two separate occasions (with presumed session continuation by john). I have the majority of a cloud-based java server that will launch john instances via a socket request (to be used by an android APK when finished), but am having some issues trying to get john to update stdout periodically with status reports. I am spawning john processes in java via the Process class (and Runtime.getRuntime().exec()), and then attempting to pass a character through the stdin of the spawned process to get it to spit out a progress update, but I can’t seem to get it to recognize it. I’ve tried appending a newline character to the end of the text I’m writing to stdin (john updates when any key is typed so I wouldn’t assume it would matter what I give it, but for reference, I’m piping it an ‘e’ character), and calling flush in several different configurations on the BufferedWriter instance I’m using, to no avail. I looked through the documentation on john to see if maybe there was an option to force it to just periodically spit out progress updates automatically, but there doesn’t appear to be such a feature. I will continue to search for a different option, but at the end of the day I may have to settle for just passing back “pseudo” progress updates to the apk via the socket so long as the process is still running.

UPDATE: I resolved the issues with getting status updates from the running john instances, the solution was to spawn up separate processes, one that sends a SIGHUP to john (via kill -1) and another that runs a john –-status command with the session name to get the status line that john normally provides. This is then piped back via the client socket to the android APK/whomever the client may be.

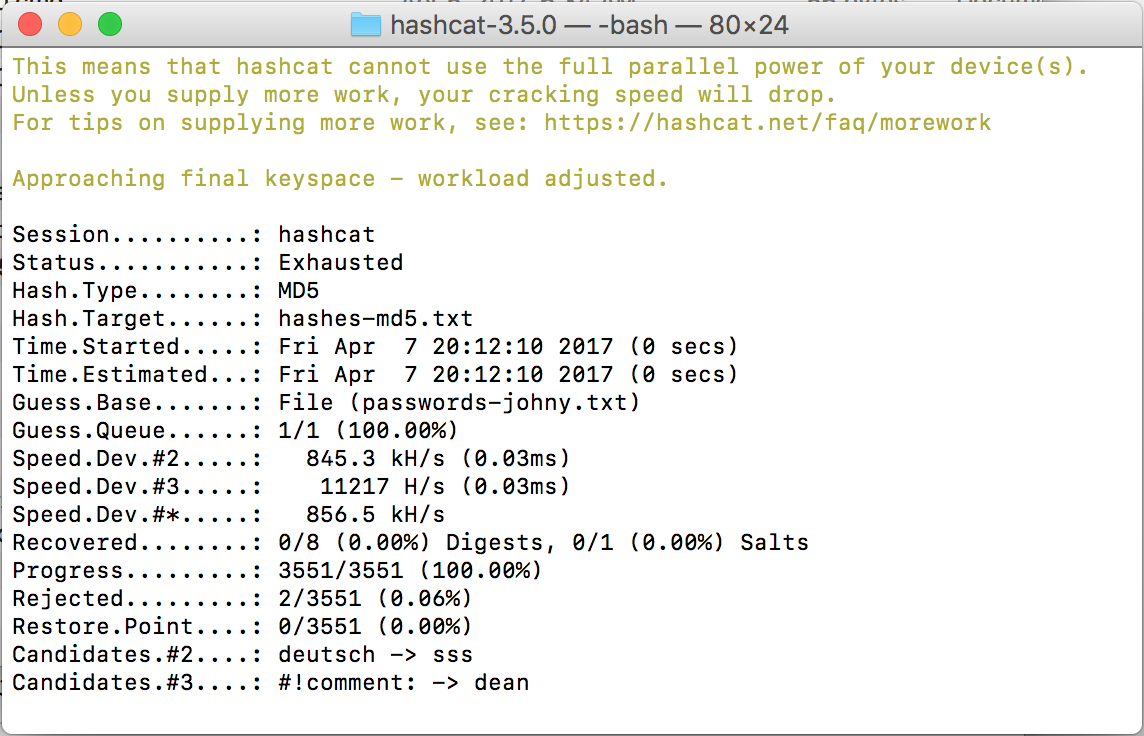
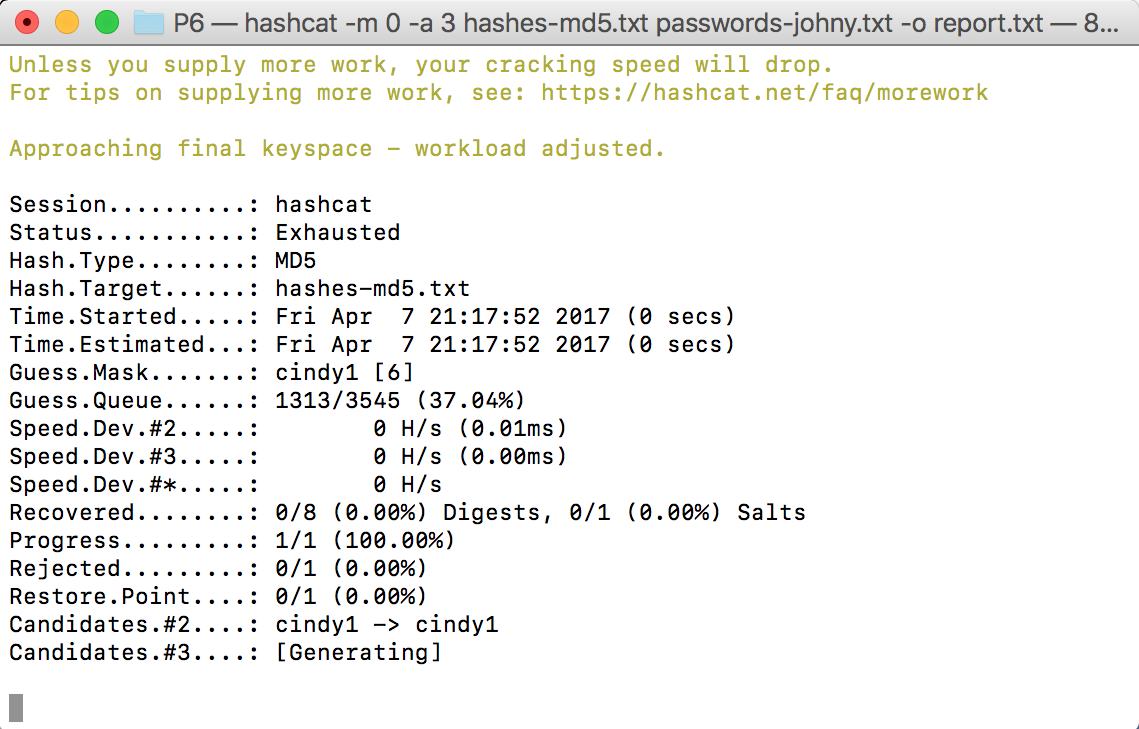
## Experience Report

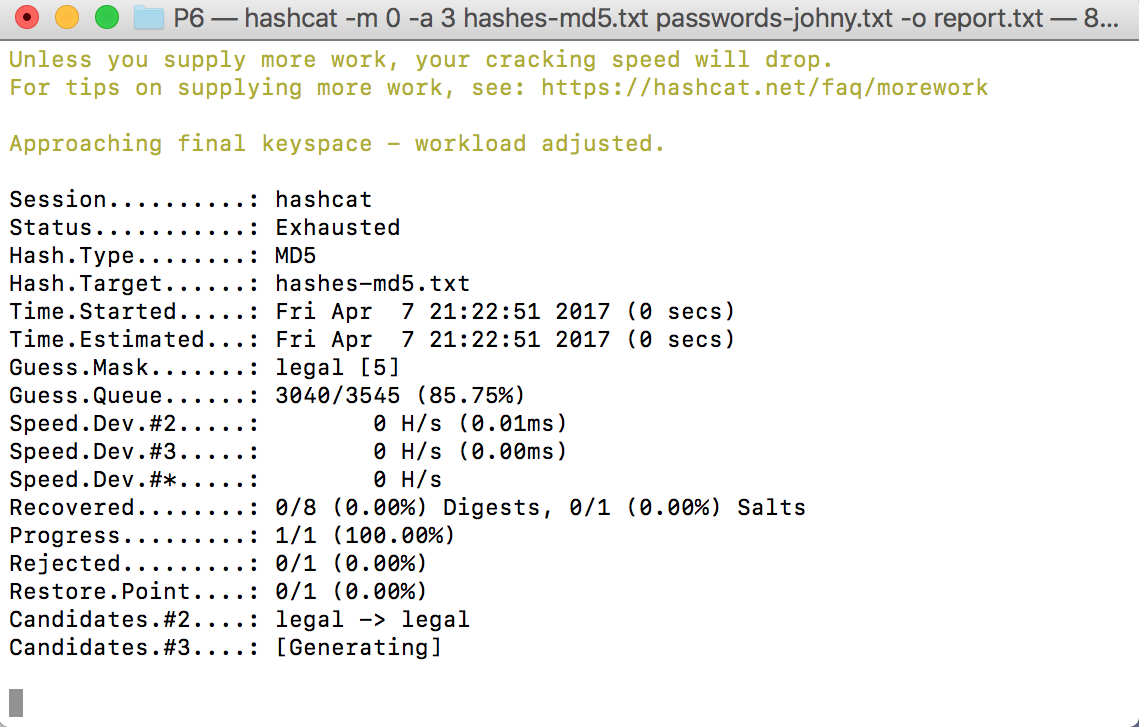
John itself seems to be a pretty powerful tool, but it seems to take a very long time to crack reasonably sized password dumps. After two sessions of 8+ hours each (with at least claimed status resuming) I was still unable to get through the shadow file provided fully. It has also proven to be quite the challenge to interact with john processes spawned programmatically via Java (as detailed in status report).

UPDATE: After another 12 hour session running overnight, John managed to crack one password from the first fourth of the blocks.

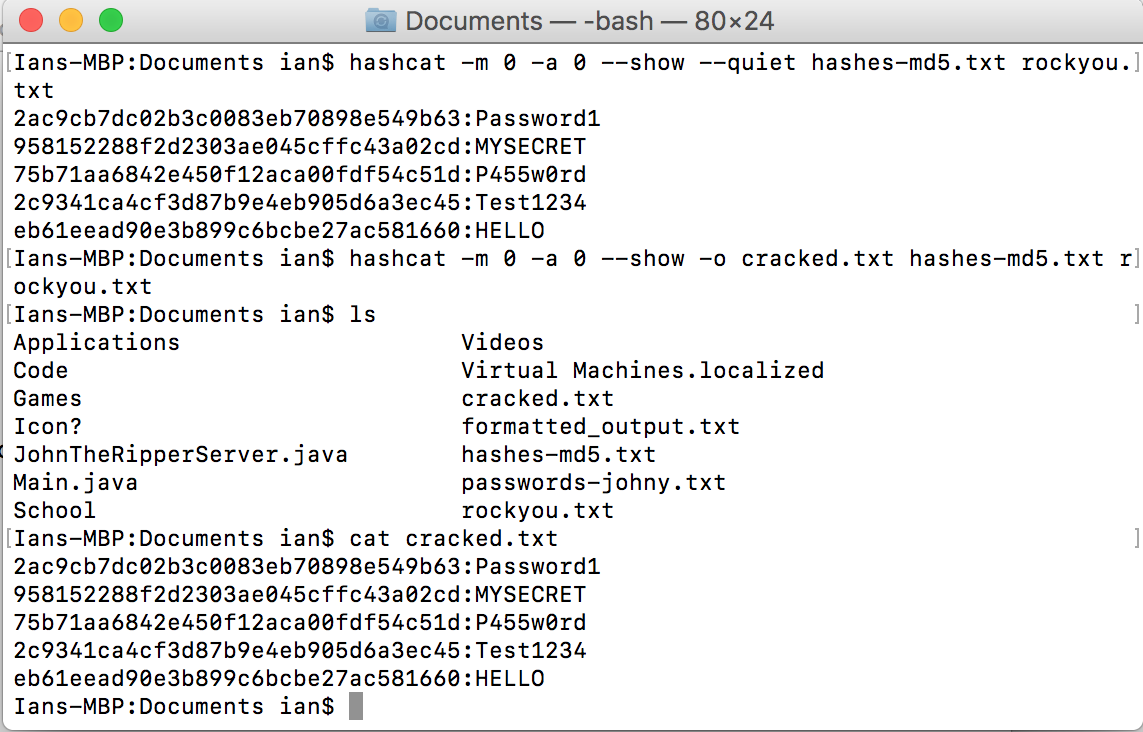
# Task 2

* Screenshots pg. 10-12
* Status Report pg. 12-13
* Experience Report pg. 13

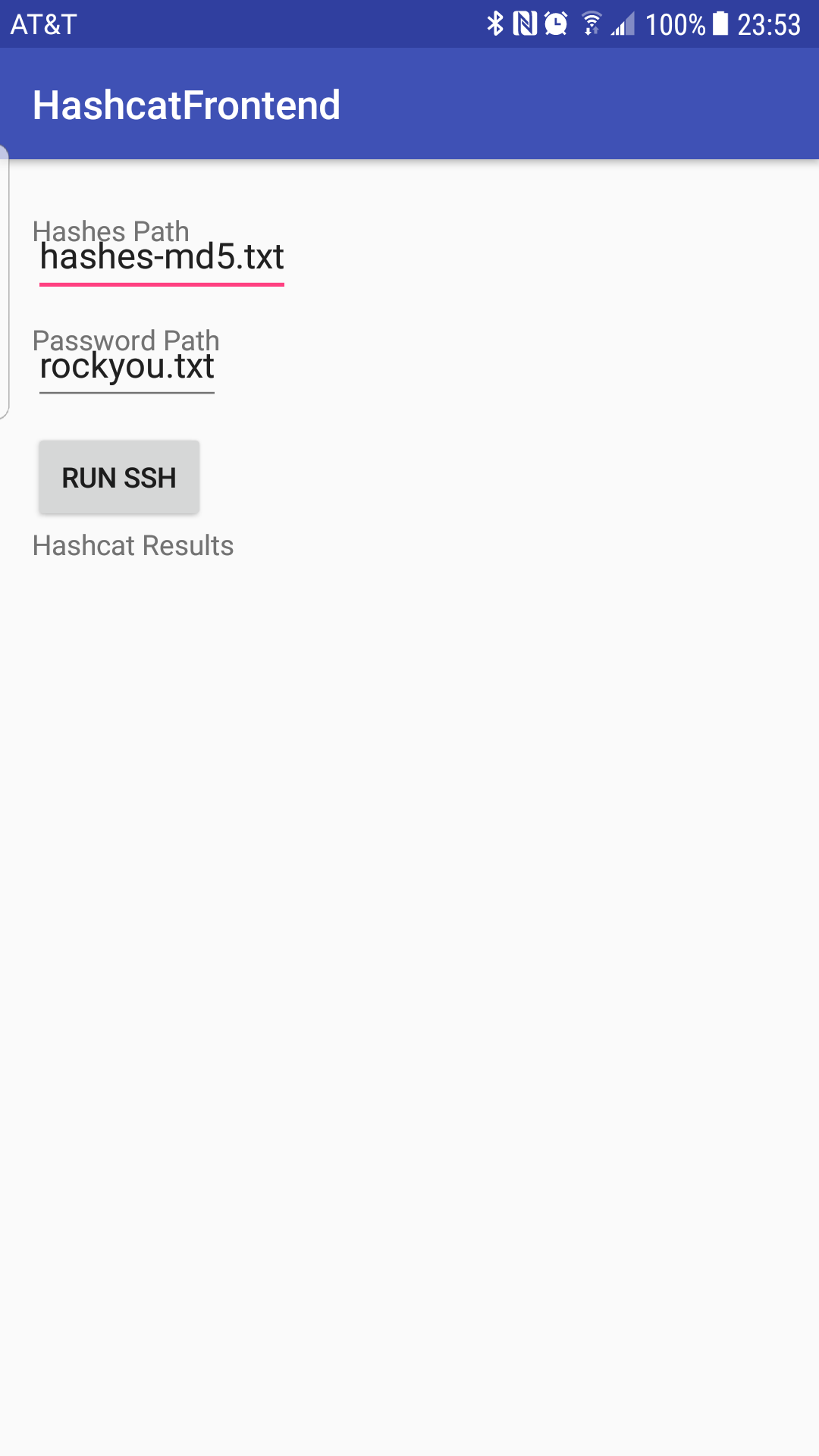
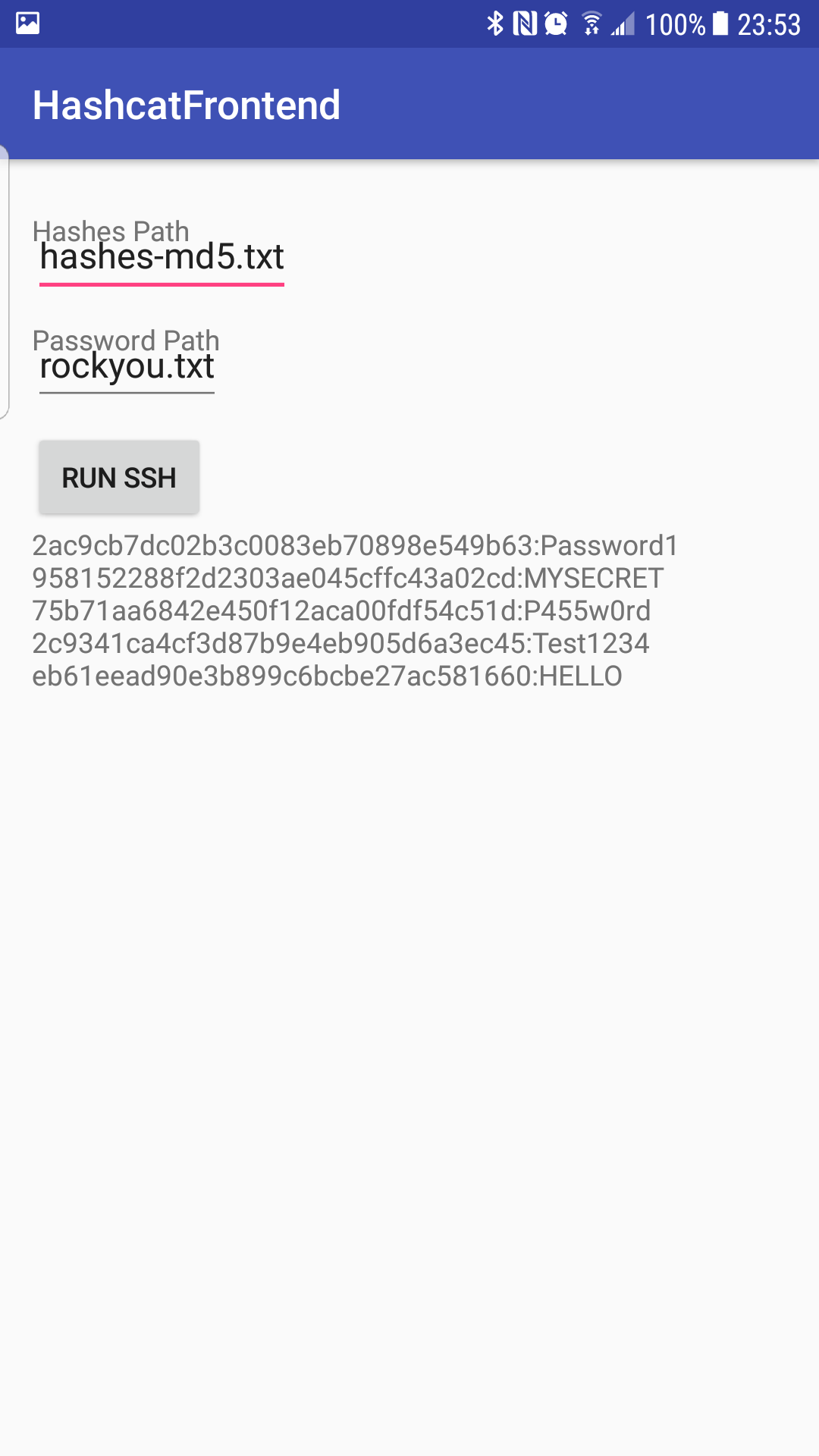
Not Working  




Working



Android APK

## Status Report

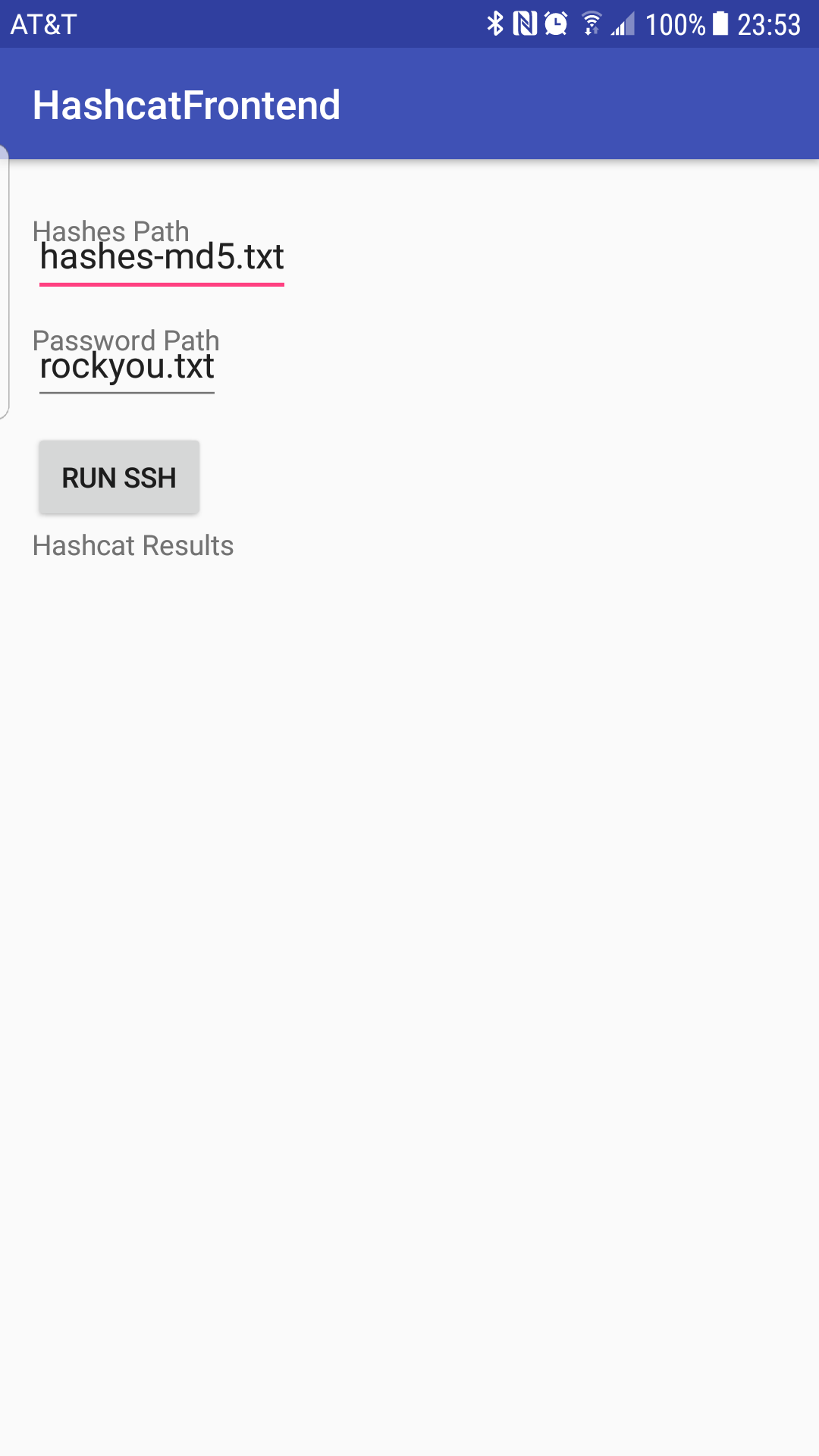
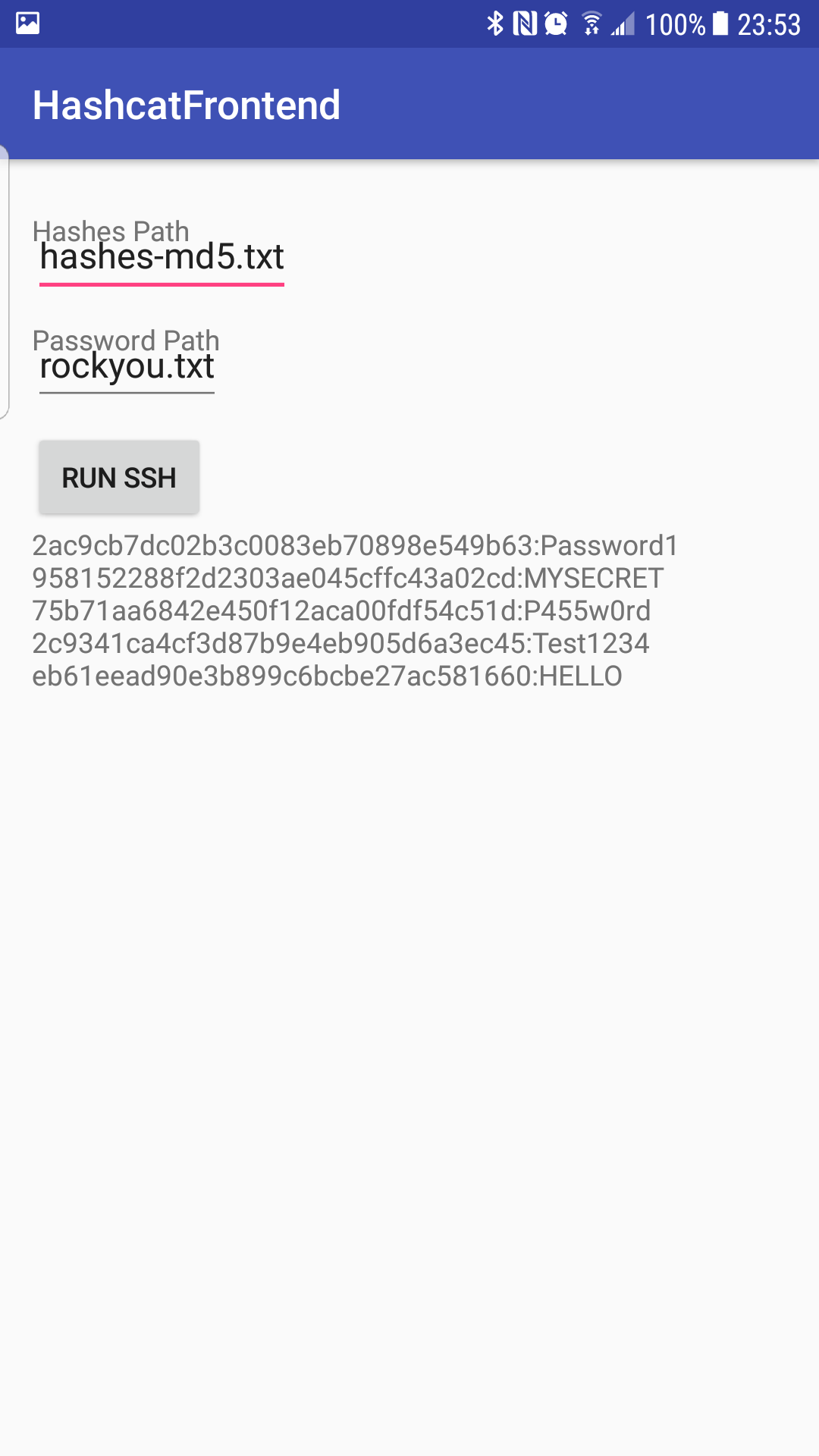
I was not able to get hashcat to install through apt-get, no package was found. I downloaded the binaries from their website and used those instead. I use a linux VM on my laptop, and through my VM hashcat could not see any graphics card drivers that it could use. I tried a couple solutions online to allow hashcat to run via the CPU but could not get them to function. Instead, I installed hashcat (via source) on my host Mac machine, which was able to access the GPU drivers and ran successfully. After that, I had troubles with the wordlist provided with P6, it complained that there were not enough words and appeared to be unsuccessful in hacking any passwords, as was mentioned in a pilot discussion, I found the rockyou.txt wordlist online (<https://wiki.skullsecurity.org/Passwords)> and used that, which solved this issue and allowed hashcat to successfully crack all password hashes given in hashes-md5.txt. To implement the Android APK, I kept a generic SSH Client implementation that runs hashcat remotely on either my laptop or an Amazon EC2 instance running in the cloud (to solve this and task 3 together). I was unsure on the specifications of the Upload URL portion of the settings dialog, so I opted to simply have two text edits that specified the files on the remote server that would be used by the command. When considering how to upload files to the server, I wasn’t sure where the files would be supplied on the android device. Whether a file picker should be shown for each item, should they be embedded in the android APK, etc? Documentation on the rules files used by hashcat (via hashcat’s wiki) were a little bit lackluster, and didn’t quite describe the overall flow of the process. Despite that, I was able to piece together one of the larger rules files provided in the hashcat source code distribution with in-line comments explaining what each command tries to do (included on Github repo as rules.txt).

## Experience Report

Installing and getting hashcat to run on my laptop was quite a pain, but once I got the proper libraries setup and everything configured I was able to breeze through this task. This task took approximately 5 hours to complete fully.

# Task 3

* Screenshots pg. 14
* Status Report pg. 14
* Experience Report pg. 14

## Status Report

This task was completely solved in conjunction with the end of Task 2. However, in addition to the apk that runs hashcat via an ssh connection, I had to find a working solution to the issue I had running hashcat on my VM, as my Amazon EC2 instance (obviously) had no GPU drivers, or a GPU for that matter. I managed to find links on their website to the Intel OpenCL drivers that allow the CPU to behave(?) like a GPU as far as hashcat is concerned (by my understanding). Installing these fixed hashcat and it ran correctly from my EC2 instance.

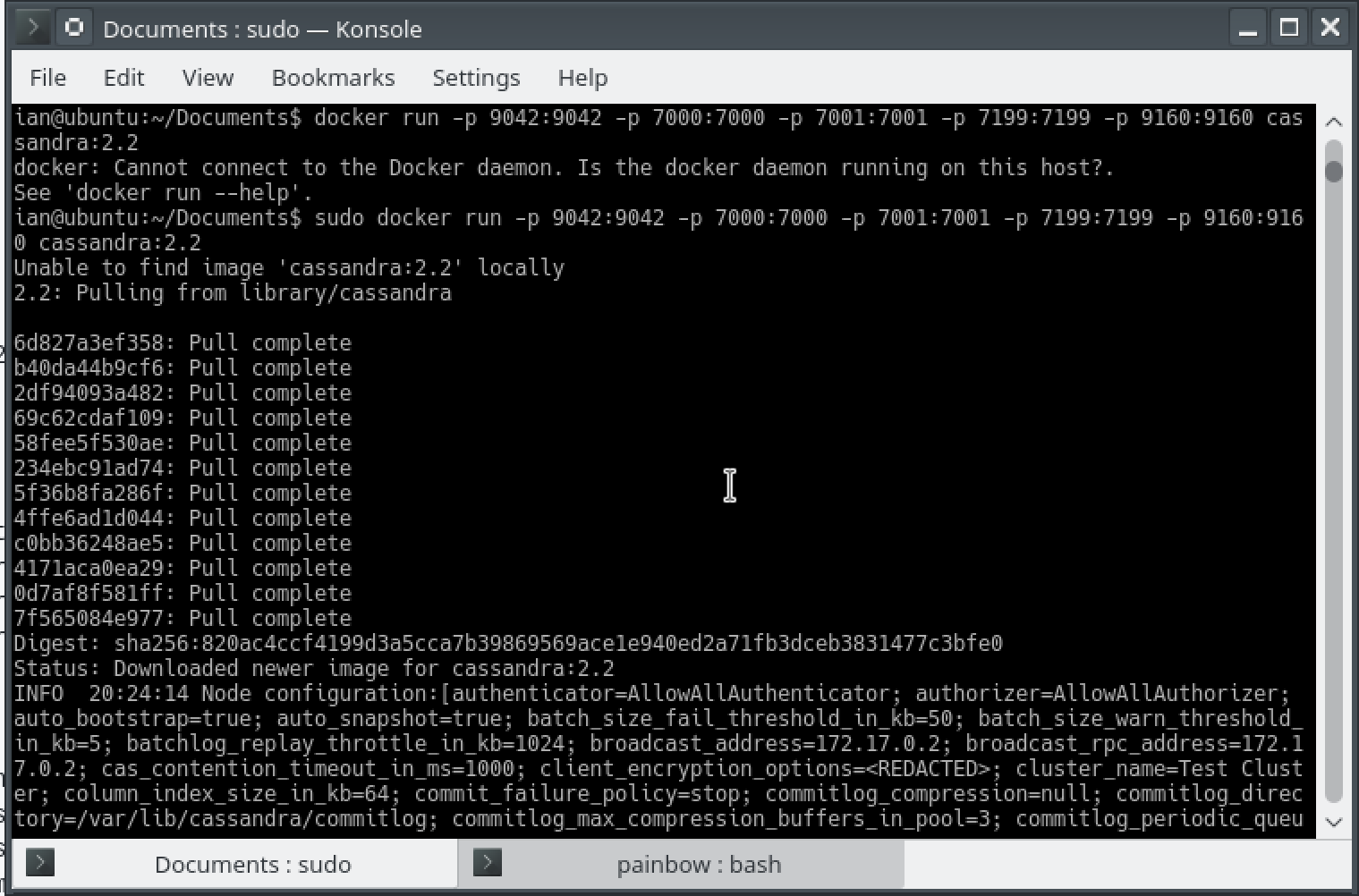
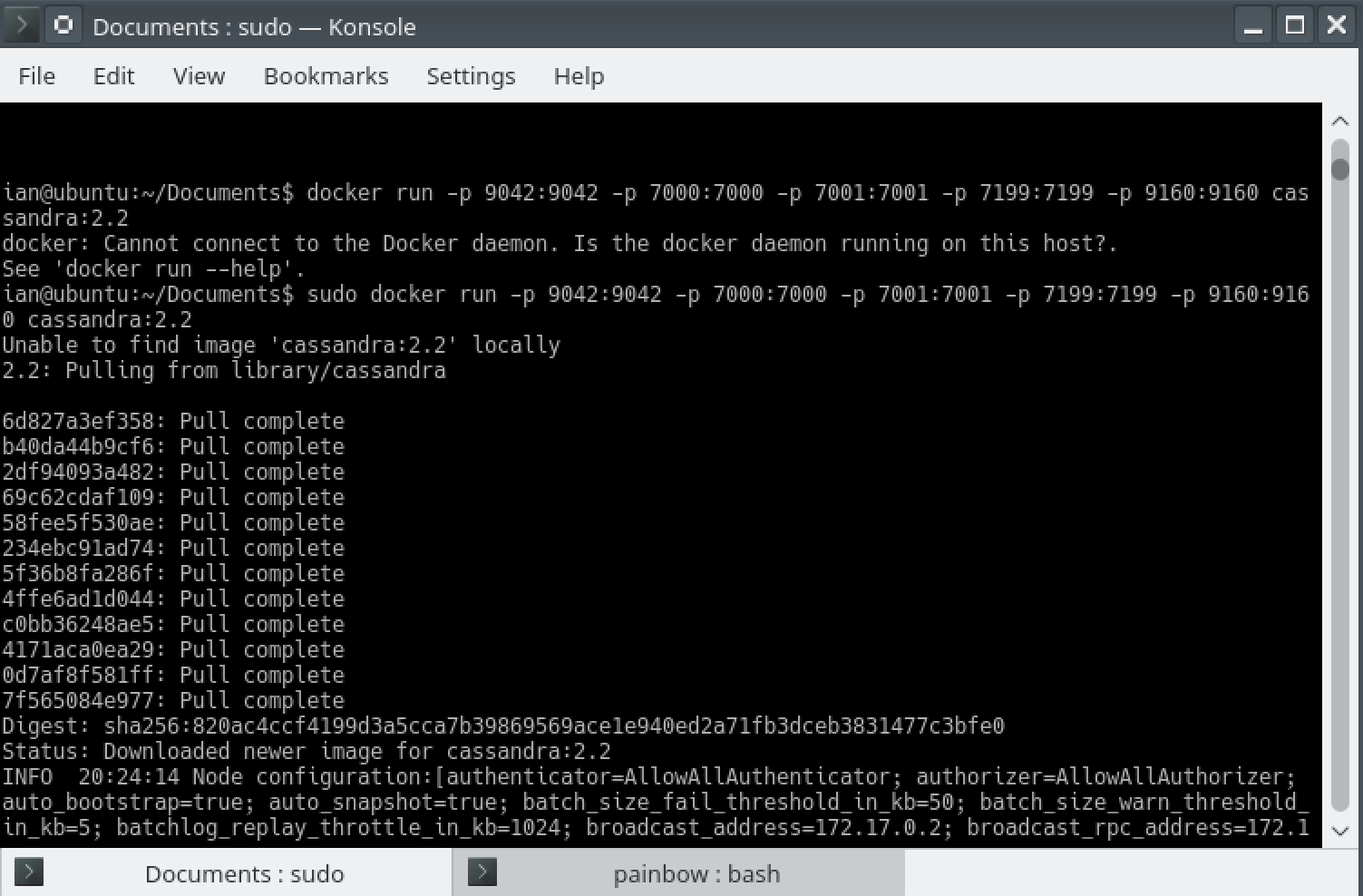
## Experience Report

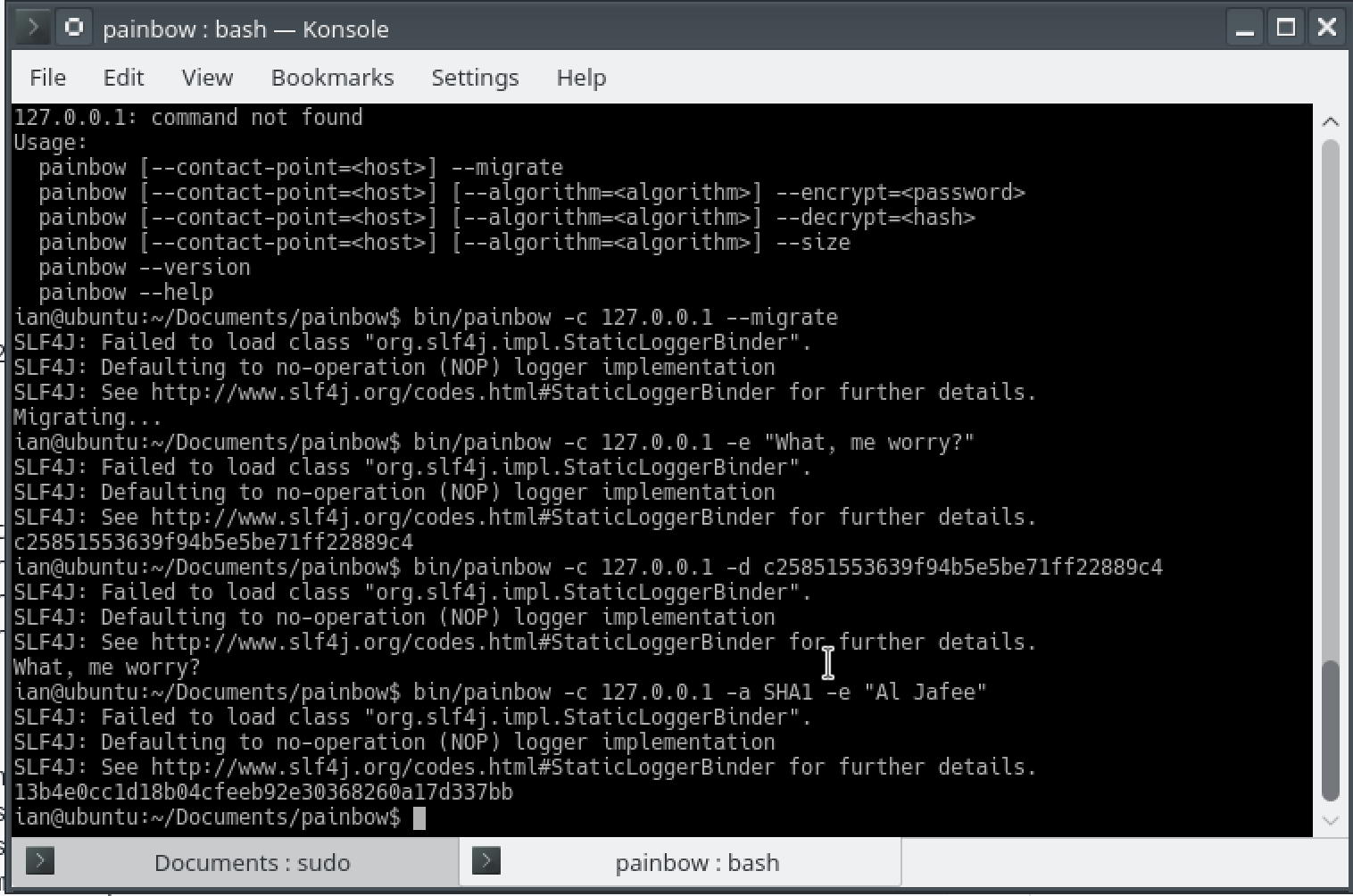
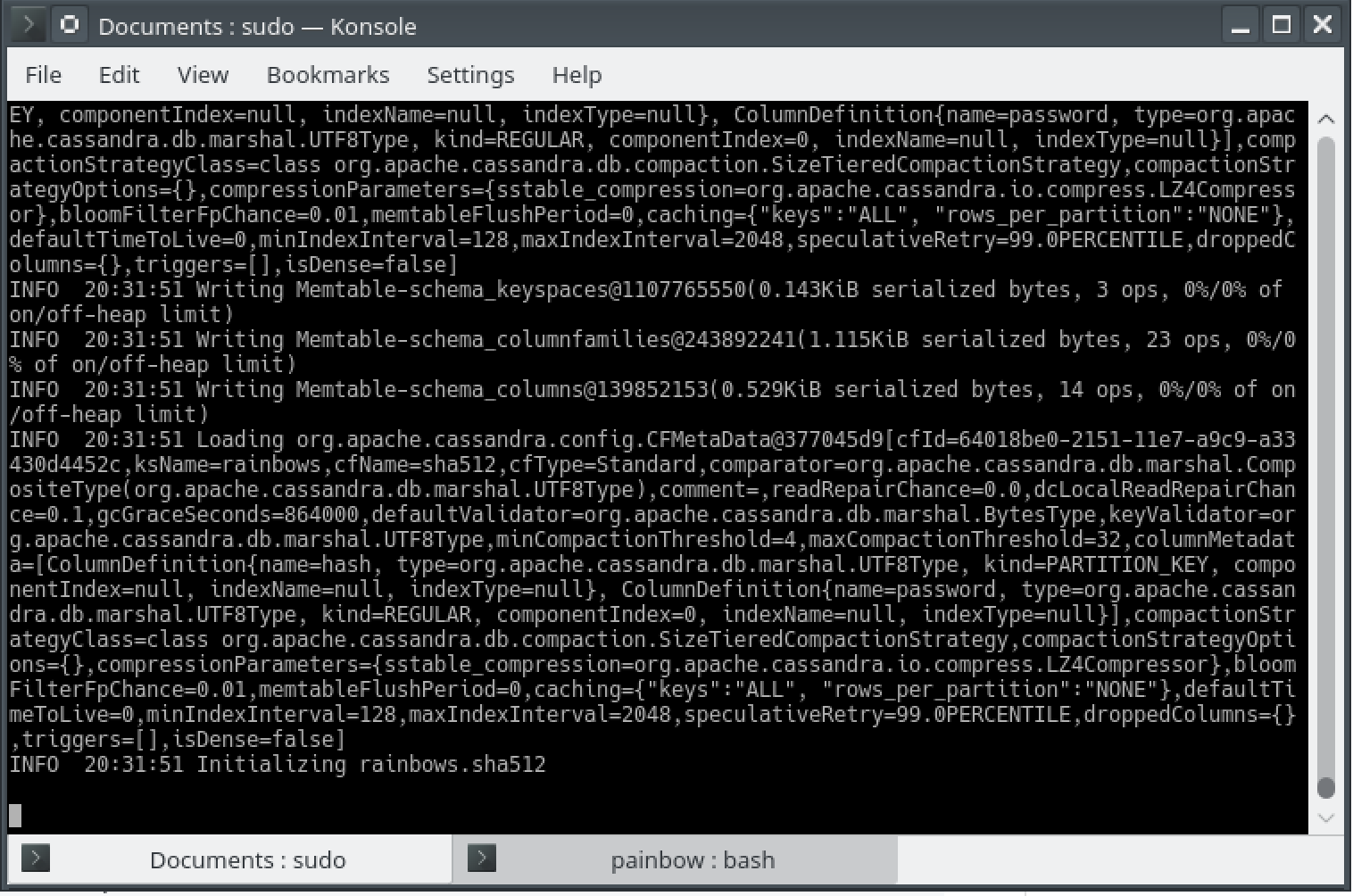
This task fit very closely together with Task 2, and only took me an hour or so to resolve the issues with the hashcat drivers. The bonus task was not attempted.

# Task 4

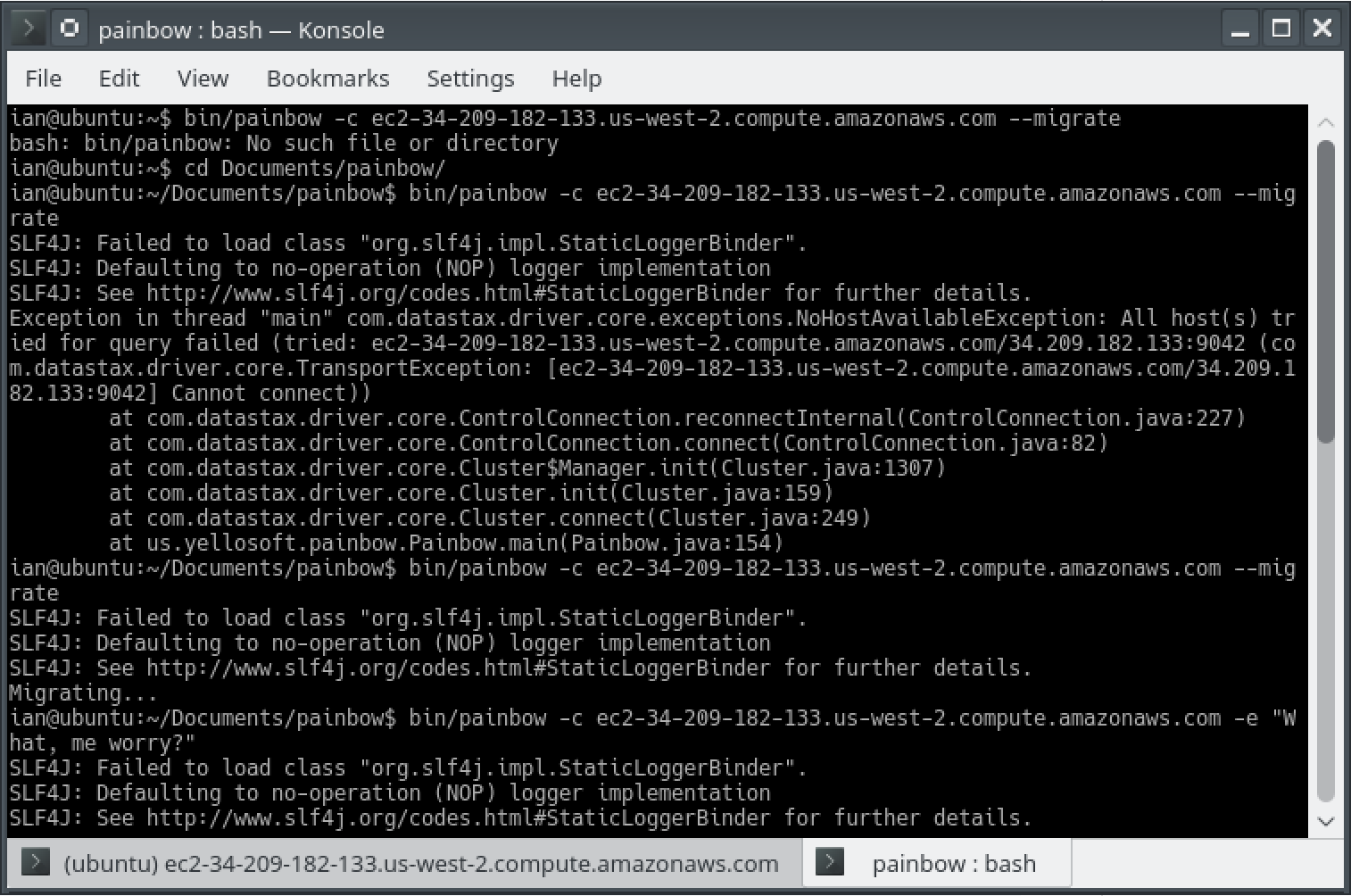
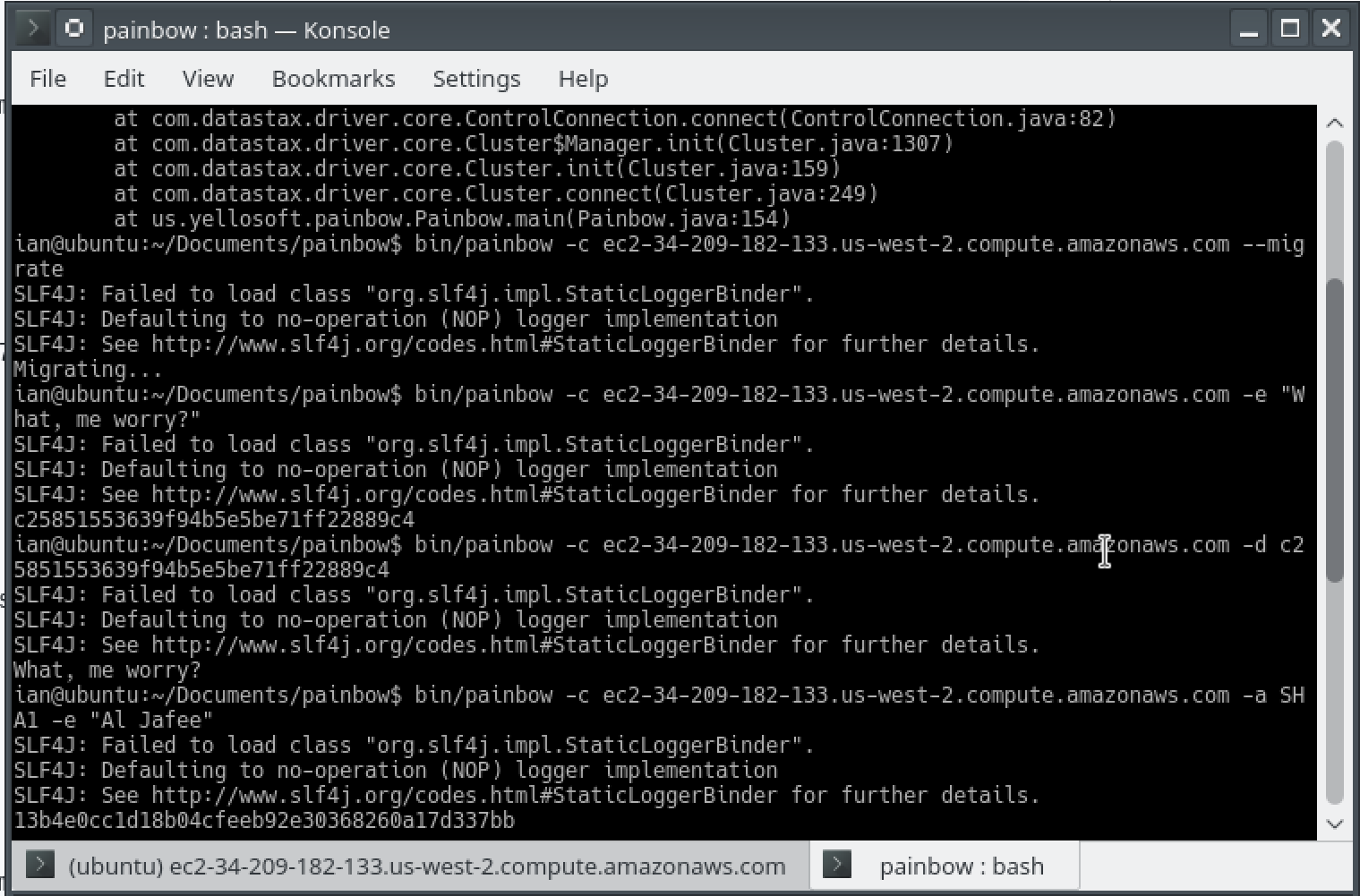
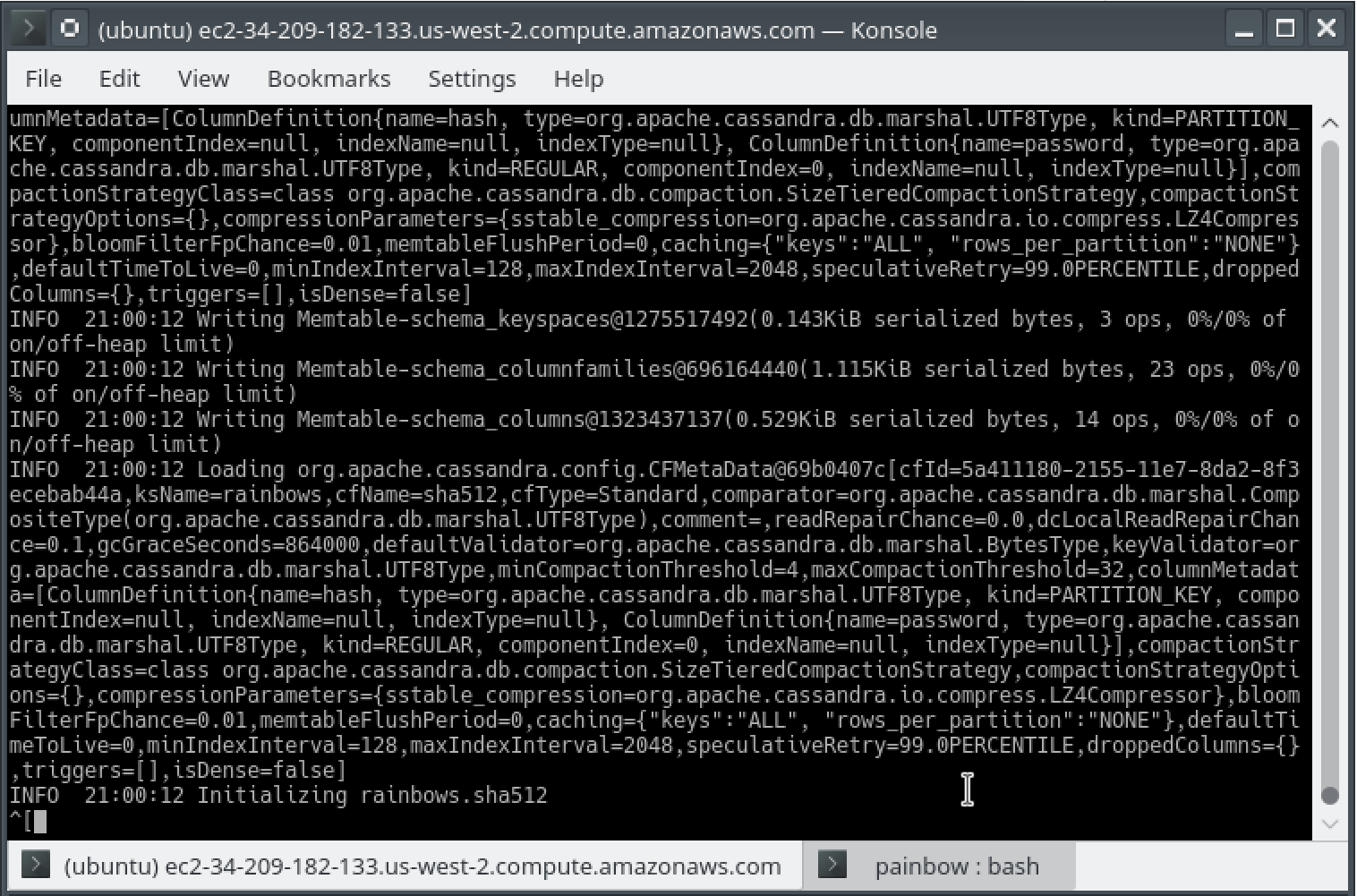
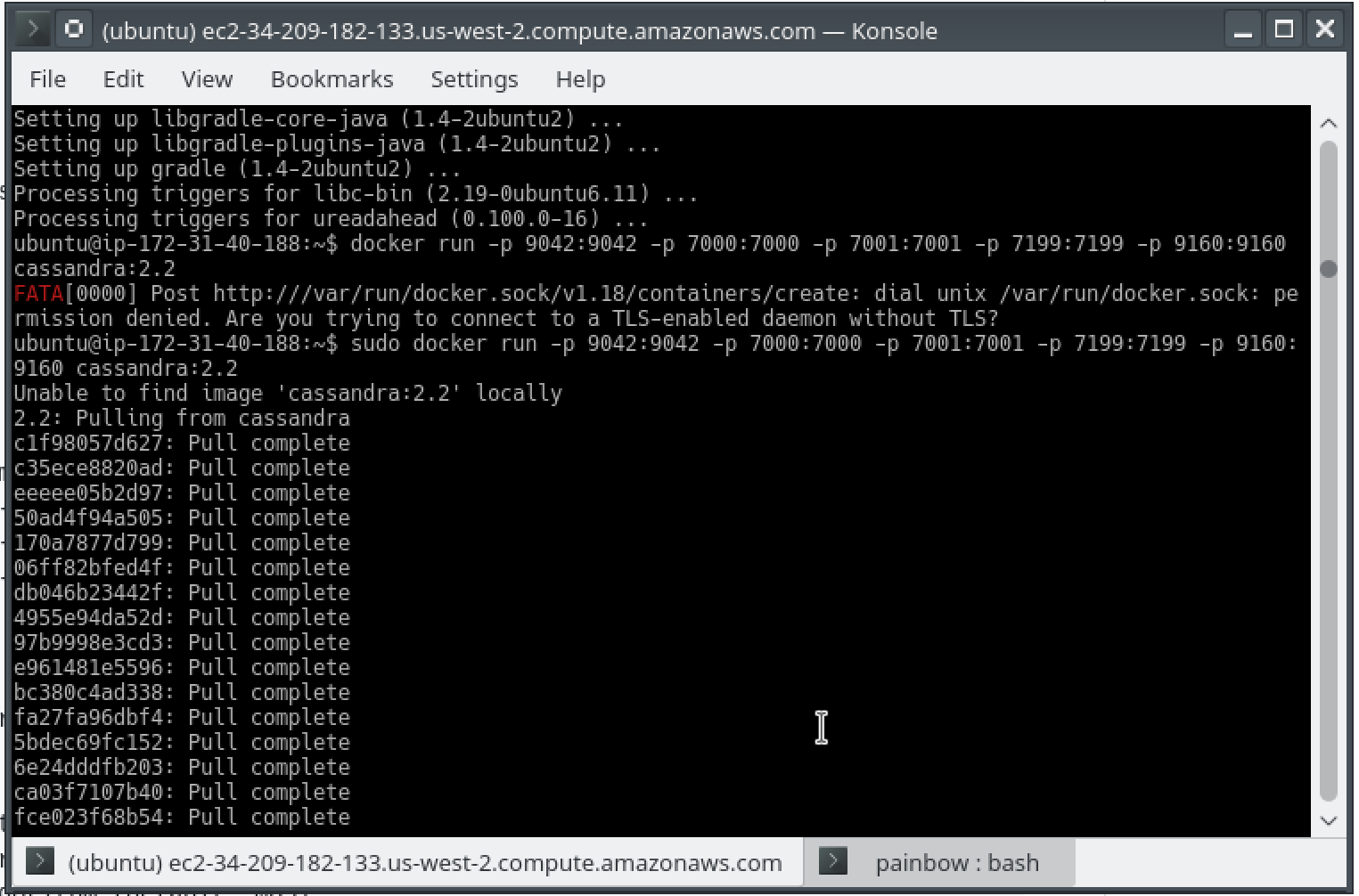
* Screenshots pg. 16-19
* Status report pg. 20
* Experience Report pg. 20

Laptop



Cloud



## Status Report

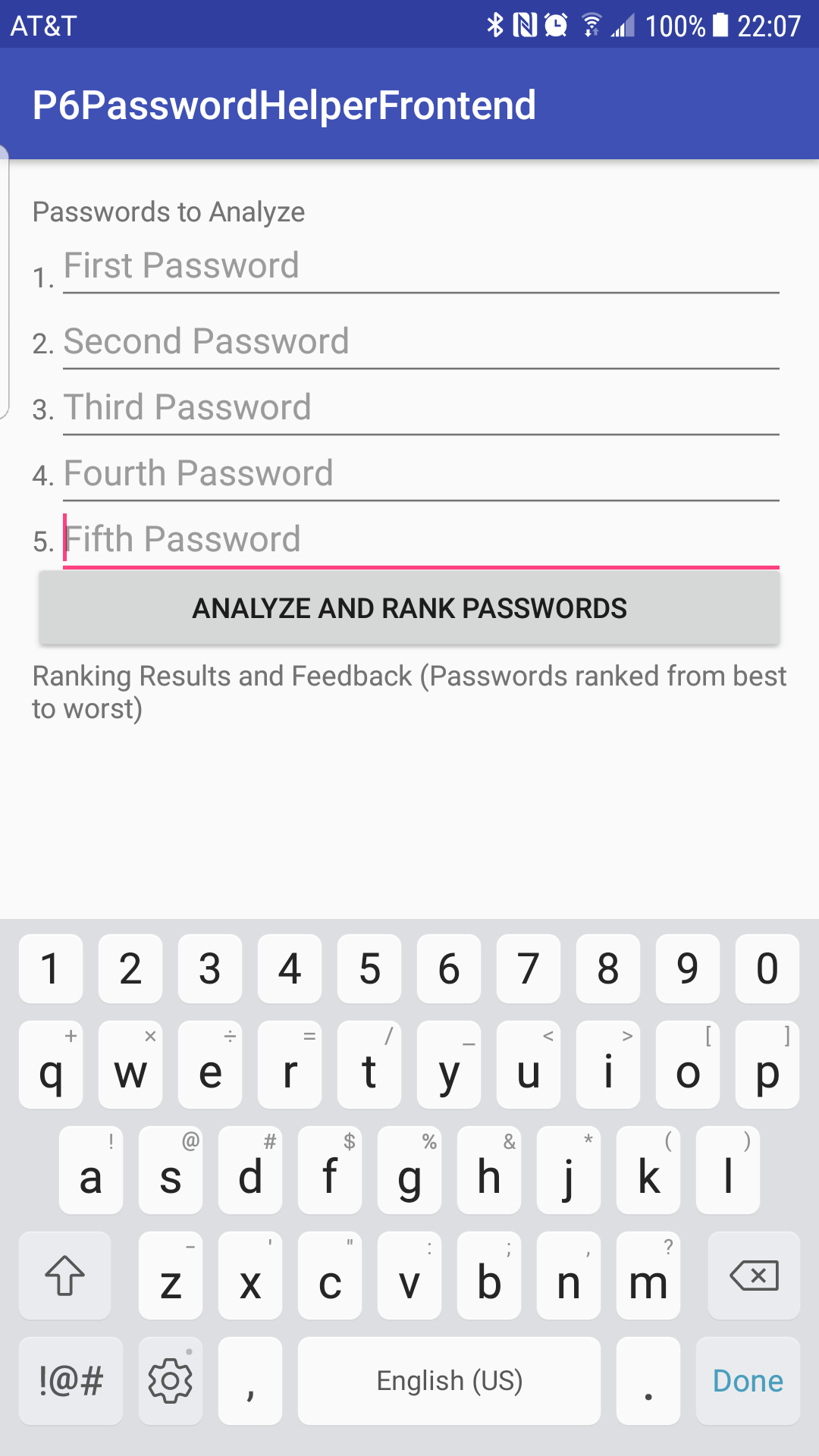
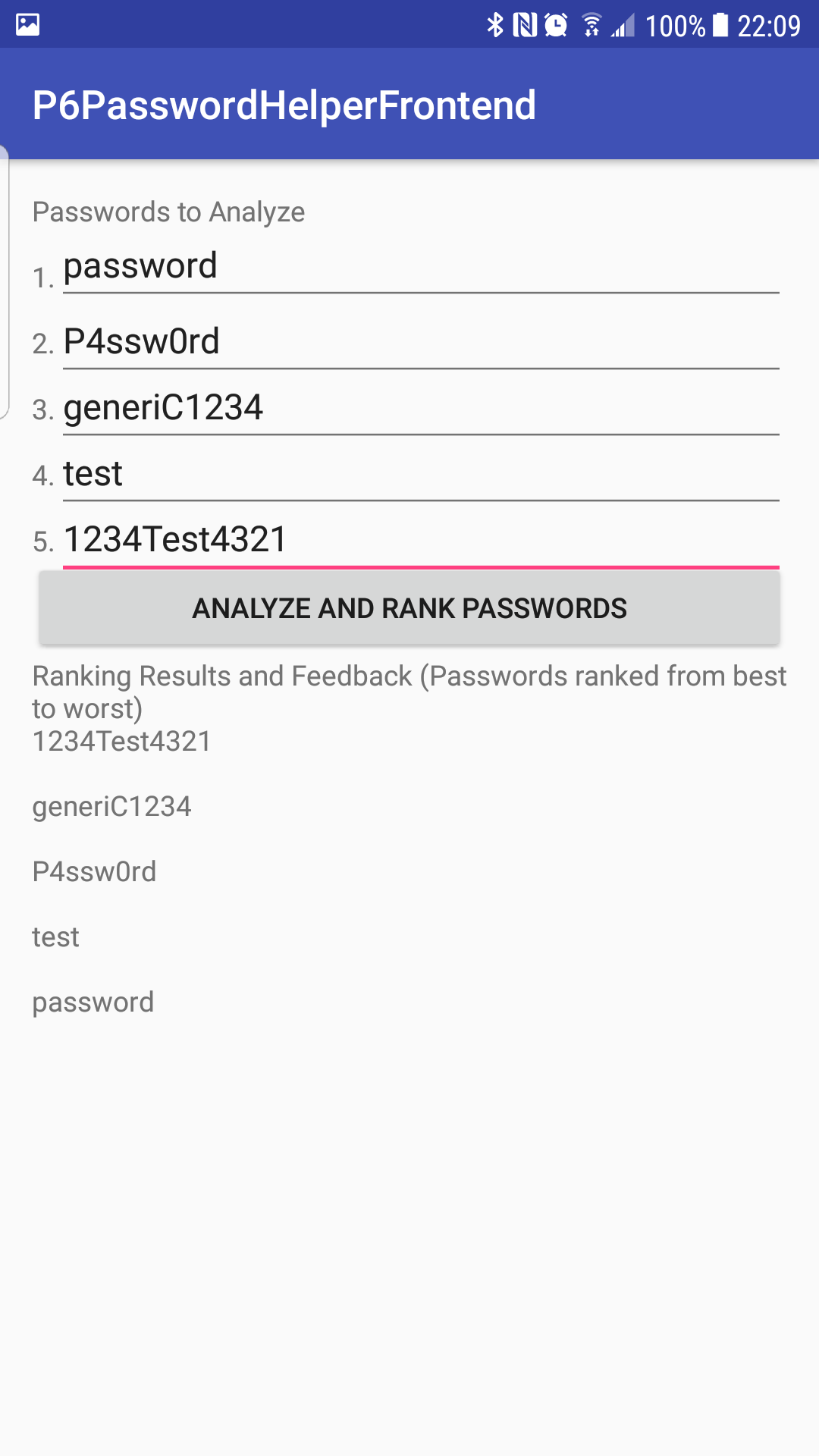
Setting up the docker container locally was a breeze, but for some reason I ended up having to run it as root, or it would complain about not being able to connect to the docker daemon (which was running). There were a few extra issues setting up in the cloud, namely with getting all the dependencies that painbow needed. After adding a repository to apt-get and installing a few items, this was resolved. Although, I was unable to build the painbow executable on the cloud, and had to copy one built locally up in order to get it running.

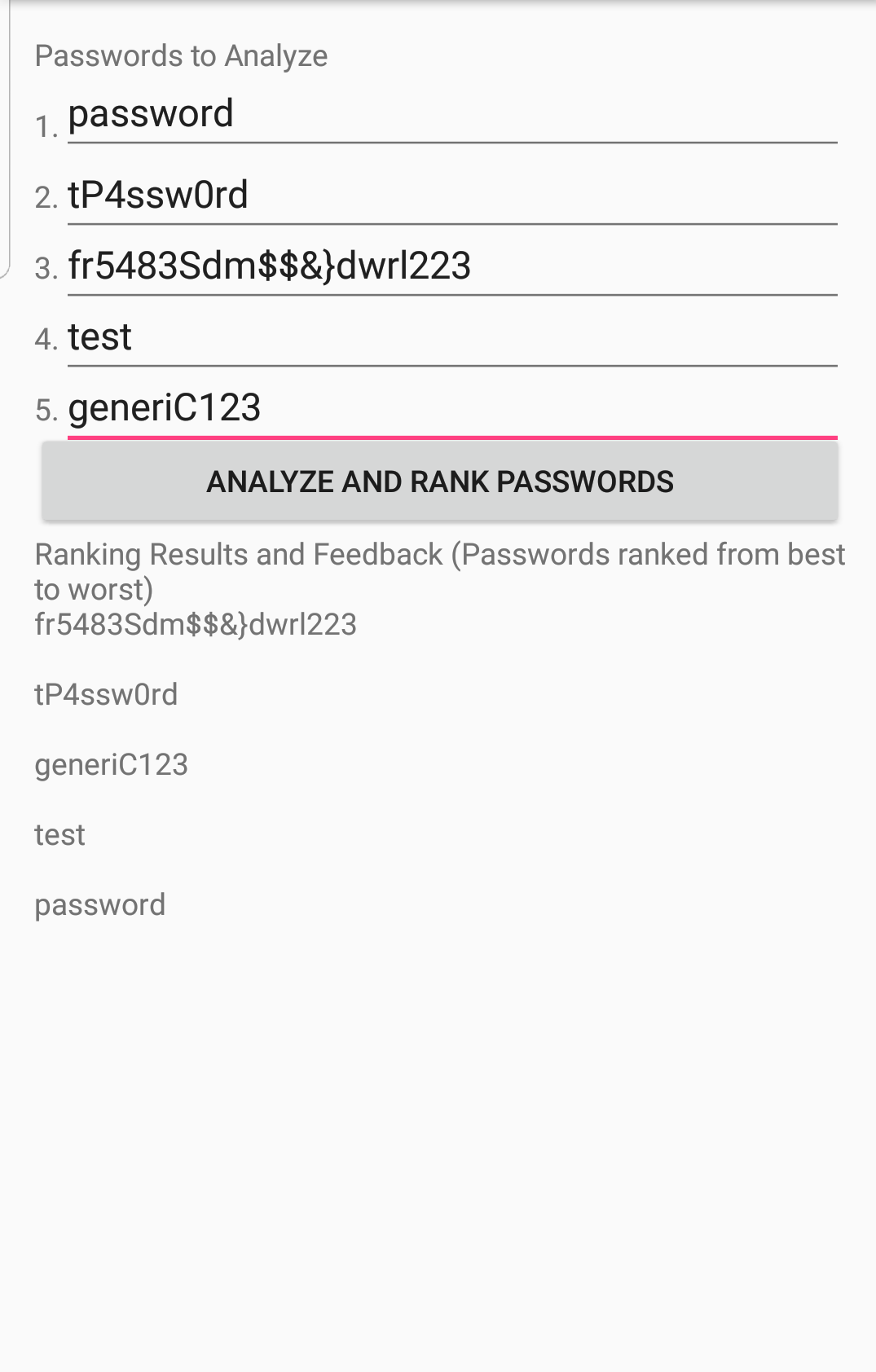
## Experience Report

Other than a few troubleshooting and dependency issues in the cloud, this task was very easy and only took about an hour to complete.

# Task 5

* Screenshots pg. 21-22
* Status Report pg. 22
* Experience Report pg. 22



## Status Report

There were a few issues incorporating the Password Strength Estimator as a library into my backend project, but adding it as a remote Maven project resolved these issues. Beyond that, I had some trouble setting up messaging between my backend and frontend. Initially, I was adding in feedback messages to the apk results view for each password. After some initial testing, however, it turned out that the feedback (or the way I was retrieving the feedback) was almost identical for every password supplied. I decided to remove this feedback from the display because of this, as it wasn’t very helpful. The internal messaging with this feedback is still in place, however, so it could be easily reincorporated if this issue were resolved. As of now, the app takes 5 passwords from the user, sends a message to a backend running on an Amazon EC2 instance in the cloud with those passwords, and the instance runs it through the estimator library, sorts all 5 passwords based on their Entropy value, and returns a sorted list of best->worst passwords out of the 5 given.

## Experience Report

This task was the most interesting out of P6. I was unclear on all the details of providing tutorial web pages, etc. I decided to implement the password estimator as suggested, taking 5 possible passwords and ranking them based on their strength. Overall this task took me about 4 hours to complete.