Running Head: Working with DD

Working with DD

Rungsree Singholka

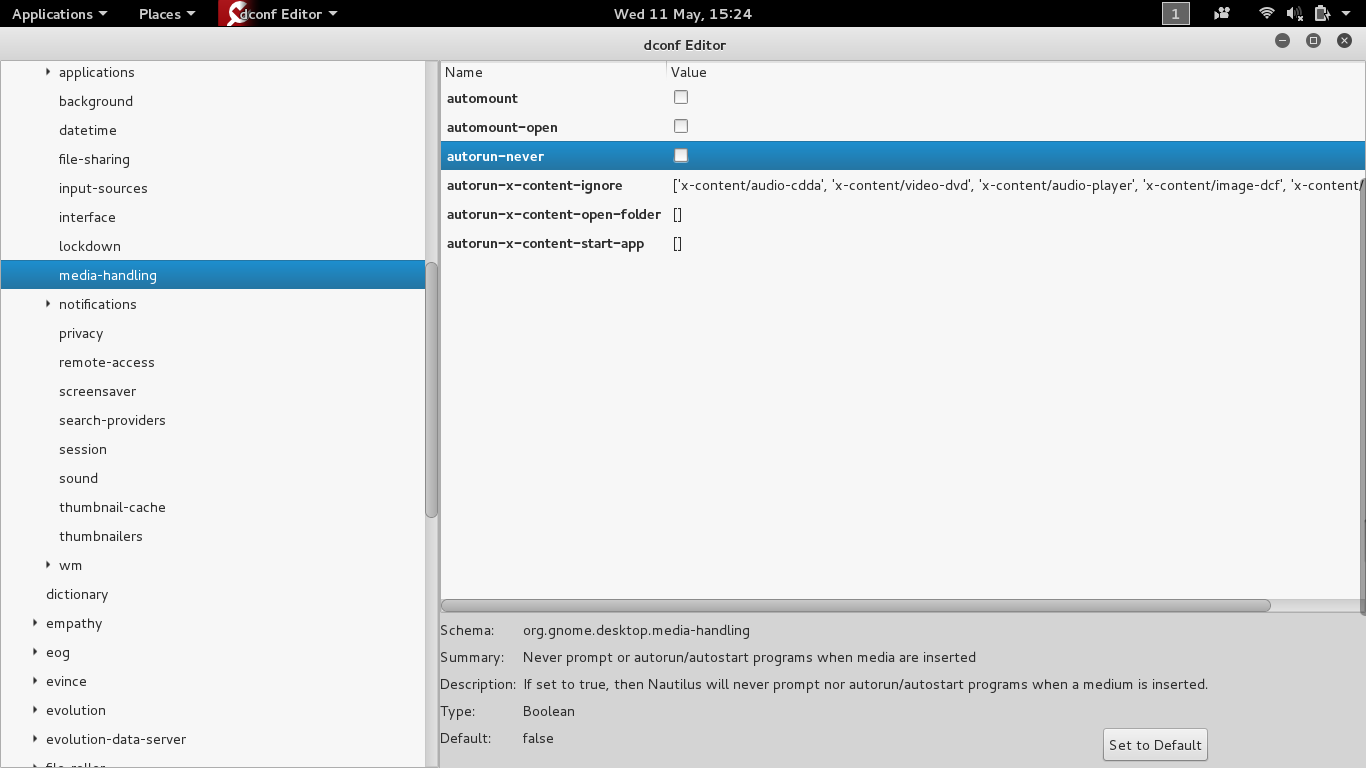
CFR105 Assignment 1.3

University of Advancing Technology

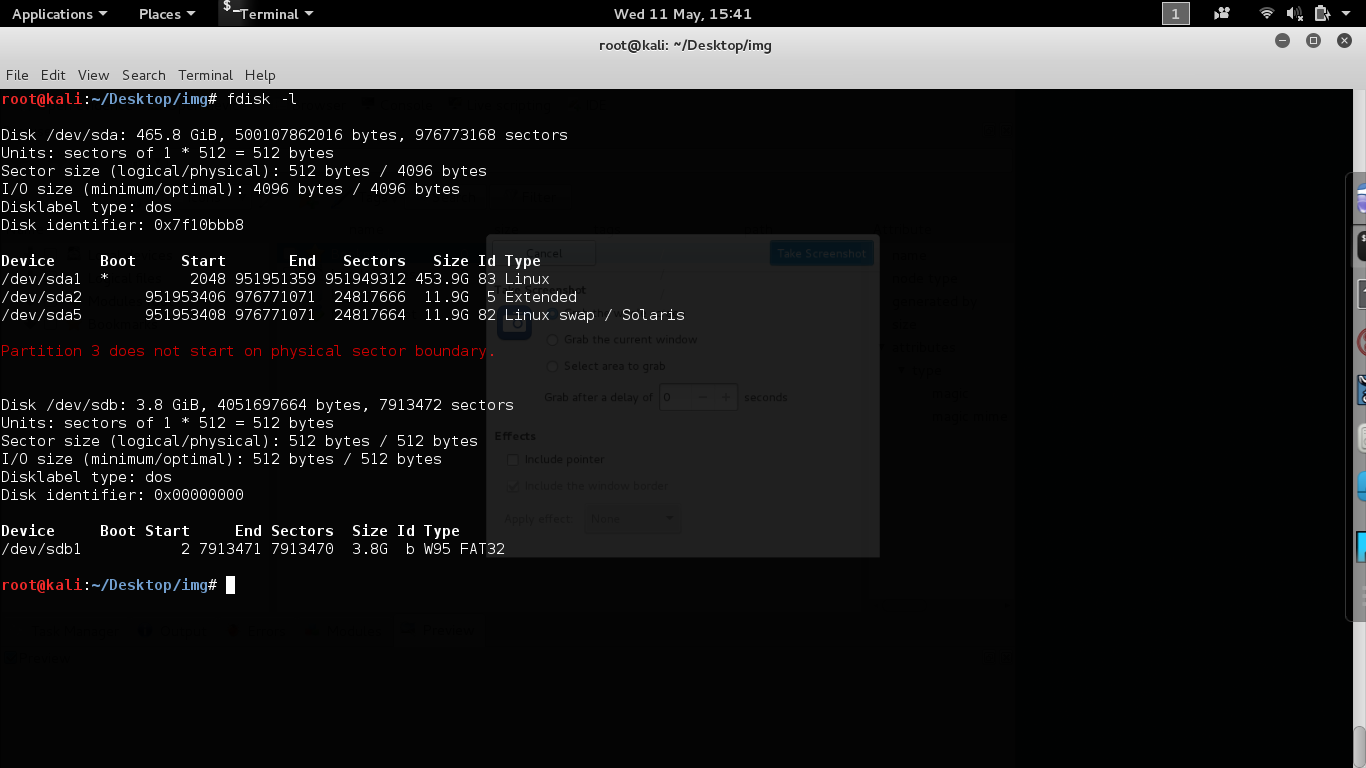
Working with DD

Dd is a great and straightforward too, and it’ll read anything that resembles a file. So learning how to use it is great, but till one gets some hands on, and try it in different ways, it’s hard to grasp the power and flexibility of this tool. The purpose of this paper is to use dd, image a USB stick, and see what it can do with some basic disk examination.

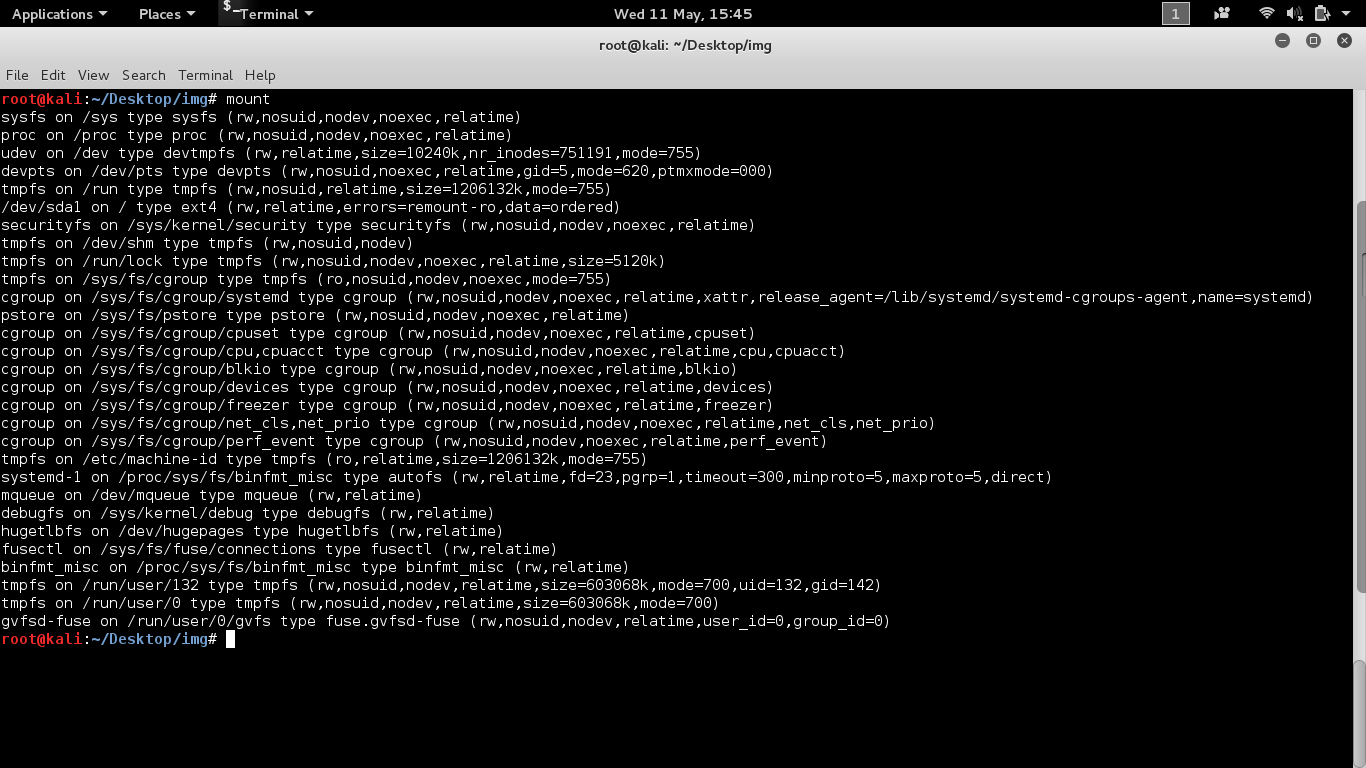
In my testing, I’ll be using a laptop installed with Kali Linux 2.0, and a 4.0 GB USB stick. Because today’s Linux are so nice, in order make a sound forensic image I had to figure out what application was doing auto-mount on my system, and disable it. After a few lookups, I was able to find it thanks to reaperz73 (2013).



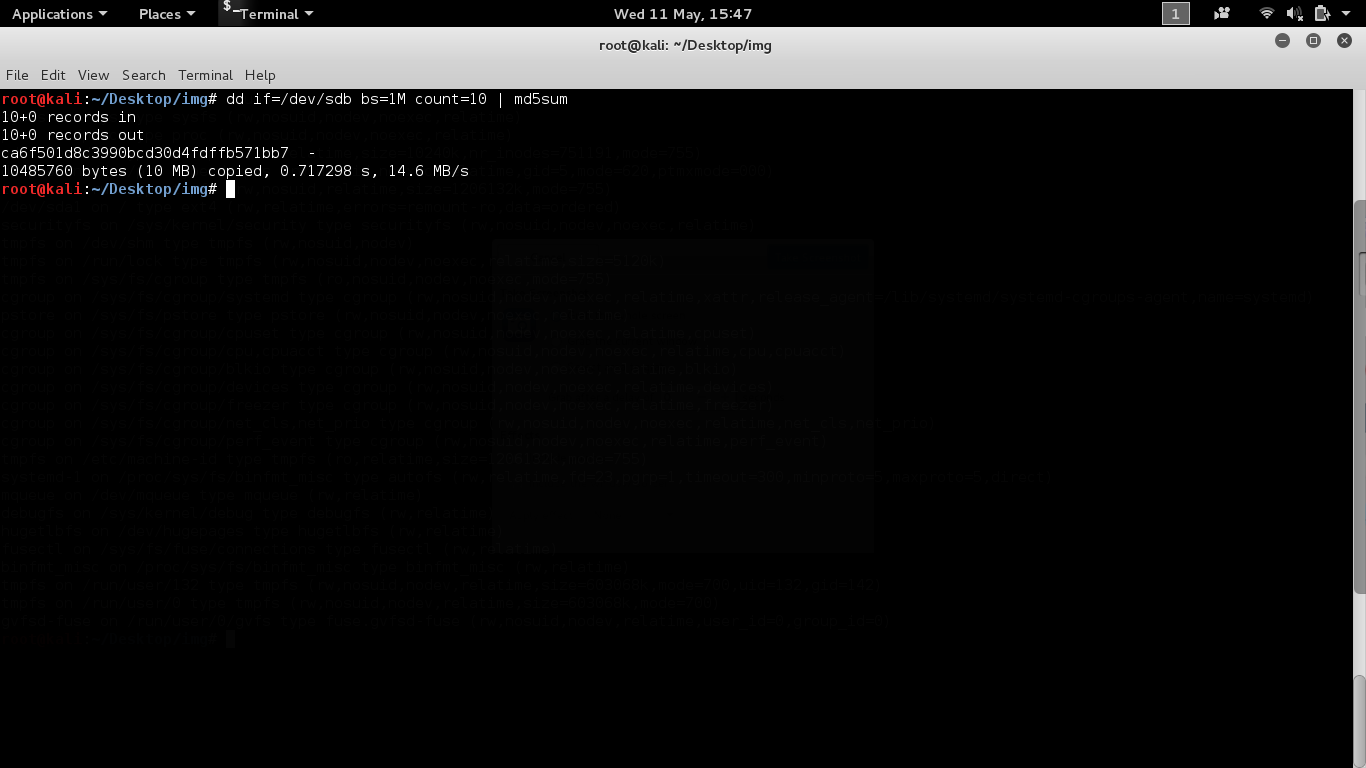
To disable auto-mount in Kali, go to dconf-editor -> org -> gnome -> desktop -> media-handling, and uncheck auto-mount. I could have boot into forensic mode as well but I feel I need to know these things as well. So I plugged in the USB, and no mount image on my desktop, which is good.



For sanity check I ran fdisk –l to see what disk the OS detects, and I see my HD and the USB on /dev/sdb with a FAT32 volume on sdb1.



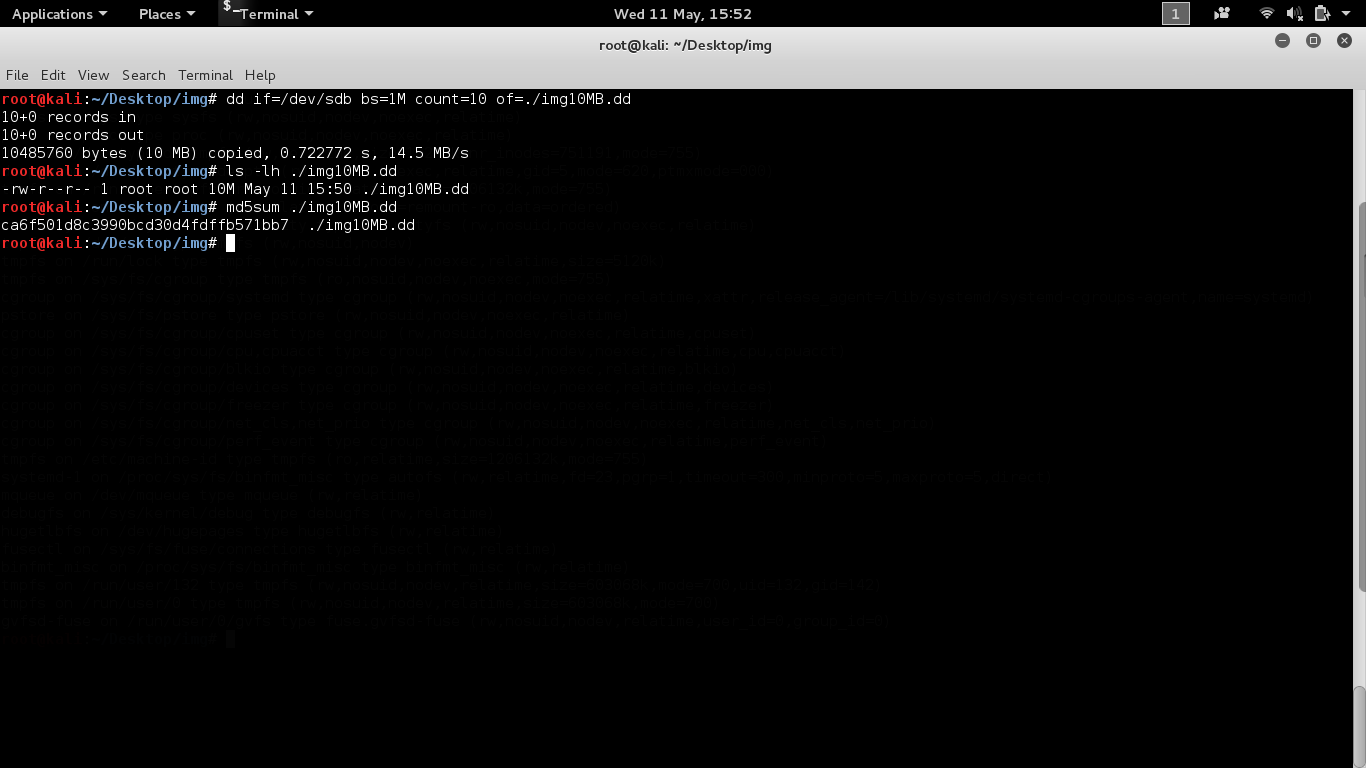
I then ran mount to check if it was mounted, and it wasn’t. Now that I’m sure that auto mount is disabled, I can start playing with dd and md5sum. Since I don’t want to wait 15-20 minutes to imaging a full 4 GB, I’m going to limit the amount of test data to just 10 MB. To do that I’ll control the amount of data read by dd with bs=1M and count=10.



Before imaging, I’ll first do a md5 hash on the first 10MB of the USB.

dd if=/dev/sdb bs=1M count=10 | md5sum

With that, I got an md5 of ca6f50...571bb7



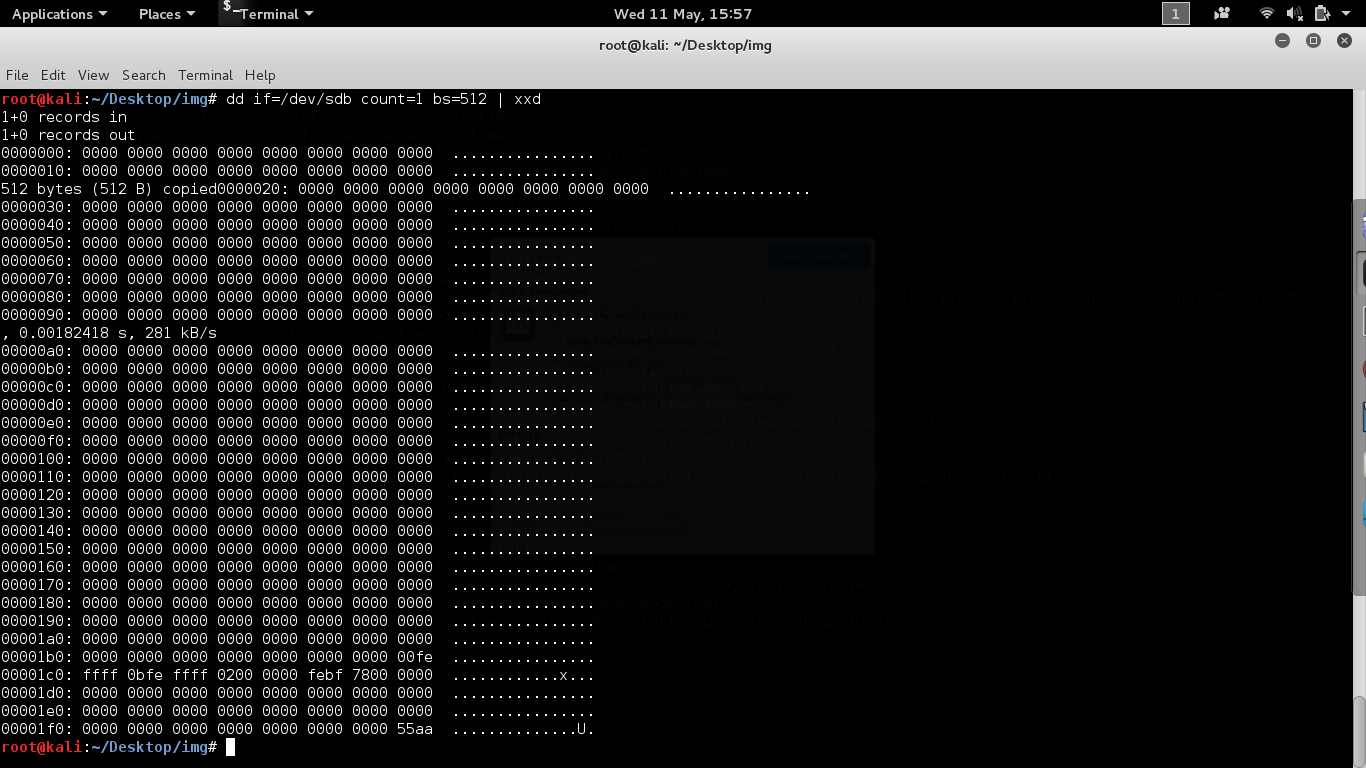
Now to make a 10 MB copy to an image file ./img10MB.dd, I’ll run.

dd if=/dev/sdb bs=1M count=10 of=./img10MB.dd

To make sure the copy was really 10 MB, I ran ls –lh ./img10MB.dd to check the file size, and it seem to be exactly 10 MB. Now the fun part; to see if the copy is exactly like the original 10 MB. To check that; I create md5 hash on the copy, and compare it with the previous md5.

md5sum ./img10MB.dd

Everything looks good and I got a matching MD5 of ca6f50...571bb7



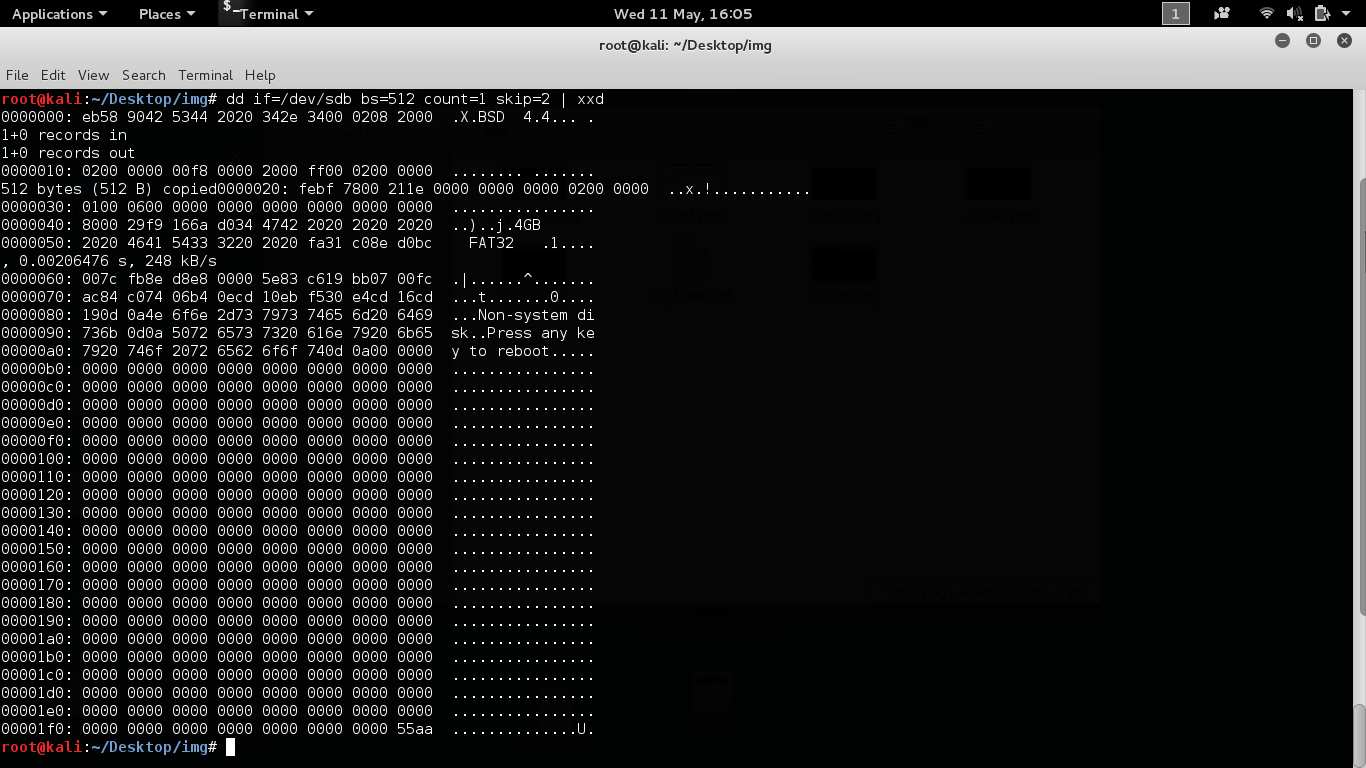
One thing I’m a little curious is why fdisk –l says my partition was only 3.8 GB when the disk is really 4 GB. So I tried the hdparam –N /dev/sdb, and found that HPA is enabled, but it doesn’t say how much but “bad/missing sense data”, and “max sector 0/1”. I couldn’t find a good explanation of this issue, so I decided to consult the partition table at offset 0x01be by running

dd if=/dev/sdb count=1 bs=512 | xxd

With this result, I can view the boot sector and the partition table. With the help from Sedory 2013, the only partition on the disk is as follow

00fe ffff 0bfe ffff 0200 0000 febf 7800

From this I know its non-bootable = 0x00, start CHS: 0xfeffff, 0x0b = FAT32, and ends CHS: 0xfeffff. Without even translating CHS, I know it can’t be right, so I continue reading Sedory 2013’s article, and he say if the size of the disk is bigger than what CHS can support, both CHS will be 0xfeffff. However, the partition table also had the starting position in LBA in the following 4 bytes from ending CHS, and the partition size in sectors in another 4 bytes block. So from the partition table above, the partition starting block in LBA is 0x02000000 and partition size of 0xfeb77800 sectors. Since they are in Little-Indian, I’ll have to reverse the bytes and translate them into decimal which are BLA: 2 and partition size of 7913470 \* 512 = 4051696640 ~ 4GB. If I look back at the fdisk –l result earlier, the program did show /dev/sdb to have 7913472 sectors, but only 3.8 GB. And the partition /dev/sdb1 starting at LBA 2 (3rd block), ending at block 7913471, total sectors of 7913470, but still with size of 3.8 GB.



By running dd /dev/sdb bs=512 count=1 skip 2 | xxd, I can see that I’m right at the beginning of the FAT table, and my reading of the partition table is correct. However, I still can’t figure out why the fdisk –l could not correctly return the true 4 GB size even when its returning the correct number of sectors. I know it can do math much better than me, but I wish I could conclude why the fdisk tool doesn’t work as I expected. Maybe it was because the HPA, but the hdparam –N didn’t help much.

So, dd is awesome, and its quite easy to use once one gets the hand of the options and flags. And since its command line, it is very flexible when used with other programs like xxd or md5sum. The skip and count option are probably my favorite because it give me control in what I want to read such as the MBR, or even various sections in the GPT. Too bad I couldn’t figure out why the fdisk –l was not returning the full 4 GBs capacity. When read it my MacBook, it shows the full 4 GB, but on the Kali machine, it’s missing 2 GBs. But the worse part is fdisk –l did show the correct number of sectors equal to 4 GB.

References

reaperz73 (2013), Automounting devices question

https://forums.kali.org/showthread.php?396-Automounting-devices-question

Sedory (2013), MBR/EBR Partition Tables

http://thestarman.pcministry.com/asm/mbr/PartTables.htm