

Using dd within UNIX to Create Physical Backups and Restore Images

Thomas Rude

In the most basic sense, the DD command is used for copying in the UNIX environment. For simplicity, we will consider 'copy' to mean 'to duplicate exactly.' The DD command is used in the Forensics Arena to perform a physical backup of the evidence. DD can be thought of as tool - in the sense that using it is a means of building an evidence file. There are other tools which can be used when making a physical backup, such as EnCase and SafeBack. However, the intent of this paper is to give some insight on what DD is and how to use it.

What is special about the DD copy command is that it has special flags available to it that make it suitable for copying block-oriented devices, such as tapes. DD is capable of addressing these block devices sequentially. We will discuss this later. But, for now, it is good to note that this is why DD can be a powerful tool when acquiring and copying tapes for cases.

I do not want to describe each and every flag option available to DD ('man DD' can show you them). I do, however, want to detail some key flags that are very useful when copying evidence. Before we can get into these, it is imperative to understand the basic syntax of the DD command:

```
dd if=/*source* of=/*destination* where:
```

```
if = infile, or evidence you are copying (ahard disk, tape, etc.)
source = source of evidence
of = outfile, or copy of evidence
destination = where you want to put the copy
```

For example, if our acquired evidence is /dev/hda, the following would produce an exact copy with the name of 'case10img1':

```
dd if=/dev/hda of=/dev/case10img1
```

Now that we see the basic use of DD we can look at the options which make it very suitable for copying in the UNIX environment. As mentioned earlier, DD is very useful when copying and/or restoring block-oriented devices, such as tapes. (NOTE: DD is an excellent tool to use when copying hard disks as well. I am stressing the usage with regards to tapes because it has proved quite useful in reducing the amount of time required to copy tapes of large sizes.) There are a few options available when copying tapes (or any device). Of the options available, I have found some more useful than others. These are shown below:

```
ibs = input block size
obs = output block size
count = number of blocks to copy
skip = number of blocks to skip at start of
inputseek = number of blocks to skip at start of output
conv = conversion
```

Let's say we have a 2GB hard disk seized as evidence. We will use DD to make a complete physical backup of the hard disk:

```
dd if=/dev/hda of=/dev/case5img1
```

Using dd within UNIX to Create Physical Backups and Restore Images

Thomas Rude

Now let's say we have an unknown tape to examine. If we are unsure of the block size used on the tape, we could use the `ibs/obs` flags to find the correct size. Finding the correct size speeds up the copying process - sometimes dramatically!

```
dd if=/dev/st0 ibs=128 of=/dev/case10img1 obs=1 count=1
```

The above usage will attempt to take 1 block with size of 128 from 'st0' and create 'case10img1' output with a block size of 1. The 'count' flag is used so that only 1 block is read. We do this because we want to limit DD to just the 1 block. If we did not set a count size DD would continue on and a whole lot of time would be wasted! What this example attempts to show is that by setting the input block size to 128 we can effectively find what the real block size is (unless, of course, it is 128!). With 512 as the standard block size, assuming 128 is virtually a failproof way to find the real block size.

The output of the above command would most likely be an 'error' message (which was our intent) with the real block size revealed (say 1024, for example).

Another example of DD usage is the following. Let's say we have an image which we need to chop up into smaller pieces. Perhaps our backup media is limited to 4 1GB discs and the evidence is 4GB in size. We could use DD with the flags below to create 4 images of the evidence, each 1GB in size.

```
dd
if=/dev/st0 count=1000000 of=/dev/case10img1dd
if=/dev/st0 count=1000000 skip=1000000 of=/dev/case10img2dd
if=/dev/st0 count=1000000 skip=2000000 of=/dev/case10img3dd
if=/dev/st0 count=1000000 skip=3000000 of=/dev/case10img4dd
```

Now, we have taken the 4GB evidence tape and chopped it into 4 separate 1GB images. Each image is 1GB in size. Let's look at this example more closely. Notice that the first command takes 1GB (count=1000000) and copies it, naming the copy 'case10img1.' The second command skips the first 1GB (skip=1000000) and then copies the next 1GB (count=1000000), naming this image 'case10img2.' We can now see exactly what the 'count' and 'skip' flags do.

As you can see, DD is a very resourceful tool to use when performing physical backups of evidence. It is especially useful when working with large hard disks and/or tapes. The examples above were created to show you different ways you can get DD to work for you. As you become more familiar with it, you will find that you can do more than what I've shown above. You may even find out that DD is also quite useful when restoring evidence! I recommend that you create some evidence disks and tapes and play with DD. Read the man page on it and try the different flags. The learning curve is not steep, and the cost (free) can't be beat.

Restoring Images

Alright, folks, we're back! For Part Due! The purpose of this paper is to get you familiar with how to restore that image you made (from 'Examples of Using DD within UNIX to Create Physical Backups'). But before I get too far along, I would like to review some basic information first.

Once again, we know that DD is a powerful command to issue! DD doesn't really care about what you are imaging (and remember, for all intents and purposes, imaging and copying will be equivalent for our work). Rather, DD only cares that you issue the command correctly, with the appropriate options set correctly (if used). DD does NOT care about file systems (it is so low level that it slides under the radar screen!). DD does NOT care about media (hard disk, floppy, CD-ROM, tape, etc., it is your

Using dd within UNIX to Create Physical Backups and Restore Images

Thomas Rude

choice. Mix and match if you like!). DD does not care about source or destination (same hard disk, different hard disks, hard disk to tape, hard disk across network to remote hard disk, etc.).

So, what does DD care about? That you set the option parameters correctly! To maximize your mileage, learn the options and set them correctly! I can only stress that the best way to do this is to practice, practice, practice! For a brief review:

```
if = file [infile] (I.E., read from FILE vs. standard input)
of = file [outfile] (I.E., write to FILE vs. standard output)
ibs = bytes [input block size] (I.E., specify the number of bytes per read operation)
obs = bytes [output block size] (I.E., specify the number of bytes per write operation)
skip = blocks (I.E., number of blocks to skip before copying starts)
seek = blocks (I.E., number of blocks to skip before writing starts)
count = blocks (I.E., number of blocks to copy)
```

Please make a note that there are a few more options you can use with DD that I do not discuss here (but a simple 'man dd' issued command will get you those!).

Now, if any of the above options seem somewhat foreign, I will try to make them a little more obvious here:

- In *nix (UNIX and Linux), standard input is the keyboard. Since we're not wanting a key logger here, we must specify a file!
- In *nix, standard output is the monitor. If we do not set this, look out!
- In *nix, everything is considered a file. A hard disk is a file. A monitor is a file. A keyboard is a file. You get the idea. Just remember that everything is a file (and in the Win32 world, you can think of most things as objects instead of files). Default block size is 512 bytes. You can multiply by a factor of 2 to set 1024, 2048, etc.

*nix accesses the hardware directly - so no worries if the computer BIOS give inaccurate drive geometry, DD will not stumble (unlike other tools that rely on the BIOS for drive settings!)

With so many cool options available, just what can we do with DD? Well, we can create boot floppies! We can create bootable CD images! We can backup our data! We can use it to find out block sizes for unknown tapes! We can use it to wipe media! And what I really like, we can create an image for data forensic analysis! And, not just any image, mind you. But a TRUE bit-stream image! That's right, bit by bit, byte by byte, a true and exact image. And, best yet, for free! That's right. No, you aren't imagining anything! FREE!

For quick review, let's create an image based on the following:

We have a 2GB hard disk we want to image. We connect the 2GB disk as a slave in our PC. Upon boot, we mount the 2GB disk read only. We are now ready to image! Using DD, how 'bout:

```
dd if=/dev/hdb of=/images/hdbApril42001.img
```

Okay, what are we doing here, you ask? Well, let's assume we are 'hda' and the primary hard disk. We first connected the 2GB hard disk as a slave (hdb), and upon boot up, we mounted hdb as read

Using dd within UNIX to Create Physical Backups and Restore Images

Thomas Rude

only (don't want to change those dates/times!). From there, we use DD to make our image, and we pump that image out to a file in our '/images' directory and call the file 'hdbApril42001.img' (I prefer to use dates as reminders, but you can use anything you like!). Now, I haven't set any block sizes simply because the default block size is the one I want: 512 bytes. This will allow me to make the image as a true bit-stream image.

Now, to add a little twist, I seriously recommend using another built in capability of many *nix systems; md5sum. That's right, md5sum. For authentication and validation, why not use it? It's free! So, let us assume we are using Red Hat Linux. Prior to DD-ing it up, we would:

```
md5sum /dev/hdb > /images/hdbAUTHoriginal.txt
```

So what do we have here? Well, the md5sum command tells the system to calculate the md5 hash authentication information for '/dev/hdb' and then pipe it (via the '> filename') to our '/images' directory, giving the file the name 'hdbAUTHoriginal.txt' (again, trying to make it easy for me later on when I want to find the AUTH info for the original 2GB hard disk).

After we calculate the AUTH info for the 2GB disk, we would then issue the DD command noted above. (You may be wondering why I have listed these out of order. Basically, because not everyone is interested in data forensics. Therefore, perhaps the md5 value is not of value to them. Though, personally, I think since it is free, why not use it?)

Great - so we calculated the md5 hash value, imaged the disk, now what's left? How about we calculate the md5 value for our image ('hdbApril42001.img') and then compare that value to our original value ('hdbAUTHoriginal.txt')? Sounds good to me!

```
md5sum /images/hdbApril42001.img > /images/hdbAUTHimage.txt
```

Now we have the md5 value for our image. We can compare the two values to ensure that they are the same (and that we have produced a true bit-stream image). For our purposes, let us assume the values are equivalent. What we could do now is one of two things: a) we could analyze the image using whatever tools we have in our toolbox, or b) we could hold off on the analysis for a moment and make sure that our slave 2GB hard disk is still virgin (I.E., unmolested and unchanged. That, in fact, it was mounted as read only and no writes were made to the disk). Sound like a good idea? Cool.

```
md5sum /dev/hdb > /images/hdbAUTHafteroriginal.txt
```

This 'hdbAUTHafteroriginal.txt' value should equal the 'hdbAUTHoriginal.txt' value. If it does, excellent! This means the contents of the slave 2GB disk did not change while we mounted it and DD-d it. That is what we want!