

Linux DD

AwesomeMachine

The basic command structure is as follows:

```
dd if=<source> of=<target> bs=<byte size> ("USUALLY" some power of 2, and usually not less than 512 bytes (ie, 512, 1024, 2048, 4096, 8192, 16384, but can be any reasonable whole integer value.) skip= seek= conv=<conversion>
```

Source is the data being read. Target is where the data gets written.

Warning!! If you reverse the source and target, you can wipe out a lot of data. This feature has inspired the nickname "dd" Data Destroyer. Warning!! Caution should be observed when using dd to duplicate encrypted partitions.

Examples

Duplicate One Hard Disk Partition To Another Hard Disk Partition: Sda2 and sdb2 are partitions. You want to duplicate sda2 to sdb2.

```
dd if=/dev/sda2 of=/dev/sdb2 bs=4096 conv=notrunc,noerror
```

If sdb2 doesn't exist, dd will start at the beginning of the disk, and create it. Be careful with order of if and of. You can write a blank disk to a good disk if you get confused. If you duplicate a smaller partition to a larger one, using dd, the larger one will now be formatted the same as the smaller one.

And there will be no space left on the drive. The way around this is to use rsync, as described in the beginning of the post.

To make an iso image of a CD: This duplicates sector for sector. MyCD.iso will be a hard disk image file of the CD.

```
dd if=/dev/hdc of=/home/sam/myCD.iso bs=2048 conv=sync,notrunc
```

You can mount the image like this:

```
mkdir /mnt/myCD
mount -o loop /home/sam/myCD.iso /mnt/myCD
```

This will make the CD root directory the working directory, and display the CD root directory: `cd /mnt/myCD`

This will duplicate a floppy disk to hard drive image file: `dd if=/dev/fd0 of=/home/sam/floppy.image`

If you're concerned about spies taking the platters out of your hard drive, and scanning them using superconducting quantum-interference detectors, you can always add a "for" loop for US Government DoD approved secure hard disk erasure. Copy and paste the following two lines into a text editor.

```
#!/bin/bash
for n in `seq 7`; do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc; done
```

Save the file as 'anti_scqid': `chmod a+x anti_swqid`

Don't run the program until you want to wipe the drive.

Best Laptop Backup: Purchase a laptop drive, and a USB 2.0 drive enclosure. Assemble the lappy

Linux DD

AwesomeMachine

drive into the external enclosure. Plug the external drive into the lappy USB port, and boot with The Knoppix live CD. Launch a terminal. This command will backup the existing drive:

```
dd if=/dev/hda of=/dev/sda bs=64k conv=notrunc,noerror
```

This command will restore from the USB drive to the existing drive:

```
dd if=/dev/sda of=/dev/hda bs=64k conv=notrunc,noerror
```

If the existing disk fails, you can boot from the external drive backup and have your system back instantaneously. After you make one backup, you can use the `rsync -avH` command to make faster backups in the future, because rsync will only copy the files changed or new since the last backup.

This series will make a DVD backup of hard drive partition:

```
dd if=/dev/hda3 of=/home/sam/backup_set_1.img bs=1M count=4430
dd if=/dev/hda3 skip=4430 of=/home/sam/backup_set_2.img bs=1M count=4430
dd if=/dev/hda3 skip=8860 of=/home/sam/backup_set_3.img bs=1M count=4430
```

And so on. This series will burn the images to DVD+/-R/RW:

```
wodim -dev=/dev/hdc --driveropts=burnfree /home/sam/backup_set_1.img
```

and so forth.

To restore the from the backup, load the DVDs in order, and use commands like these:

```
dd if=/media/dvd/backup_set_1.img of=/dev/hda3 bs=1M conv=sync,noerror
```

Load another DVD

```
dd if=/media/dvd/backup_set_2.img of=/dev/hda3 seek=4430 bs=1M conv=sync,noerror
```

Load another DVD

```
dd if=/media/dvd/backup_set_3.img of=/dev/hda3 seek=8860 bs=1M conv=sync,noerror
```

and so forth.

If you wrote chat messages and emails to another girl, on your girlfriend's computer, you can't be sure the files you deleted are unrecoverable. But you can make sure if anyone were to recover them, that you wouldn't be busted.

```
dd if=/dev/sda | sed 's/Wendy/Janet/g' | dd of=/dev/sda
```

Where every instance of Wendy is replaced by Janet, over every millimeter of disk. I picked names with the same number of characters, but you can pad a smaller name with blanks.

This command will overwrite the drive with zeroes:

```
dd if=/dev/zero of=/dev/sda bs=4k conv=notrunc
```

I just want to make sure my drive is really zeroed out!!

Linux DD

AwesomeMachine

```
dd if=/dev/sda | hexdump -C | grep [^00]
```

... will return output of every nonzero byte on the drive. Play around with it. Sometimes drives don't completely zero out on the first try.

To make a bootable flash drive: Download 50 MB Debian based distro. Plug in the thumb drive into a USB port. Do: `dmesg | tail`

Look where the new drive is, `sdb1`, or something similar. Do:

```
dd if=/home/sam/insert.iso of=/dev/sdb ibs=4b obs=1b conv=notrunc,noerror
```

Set the BIOS to USB boot, and boot. This command will duplicate the MBR and boot sector of a floppy disk to hard drive image:

```
dd if=/dev/fd0 of=/home/sam/MBRboot.image bs=512 count=2
```

To clone an entire hard disk. `/dev/sda` is the source. `/dev/sdb` is the target:

```
dd if=/dev/sda of=/dev/sdb bs=4096 conv=notrunc,noerror
```

Do not reverse the intended source and target. It happens once in a while, especially to the inexperienced user. `Notrunc` means 'do not truncate the output file'. `Noerror` means to keep going if there is an error. `Dd` normally terminates on any I/O error.

Duplicate MBR, but not partition table. This will duplicate the first 446 bytes of the hard drive to a file:

```
dd if=/dev/sda of=/home/sam/MBR.image bs=446 count=1
```

If you haven't already guessed, reversing the objects of `if` and `of`, on the `dd` command line, reverses the direction of the write.

To wipe a hard drive: (Boot from a live CD distro to do this.)

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

This is useful for making the drive like new. Most drives have `0x00h` written to every byte, from the factory.

To overwrite all the free disk space on a partition (deleted files you don't want recovered):

```
dd if=/dev/urandom of=/home/sam/bigfile.file
```

When `dd` outputs `no room left on device` all the free space has been overwritten with random characters. Delete the big file with `rm bigfile.file`.

The output of the command line: `less /home/sam/file.bin` looks like `GobbleDeeGook`, because it's a binary file.

This command returns readable output: `dd if=/home/sam/file.bin | hexdump -C | less`

Linux DD

AwesomeMachine

Sometimes one wants to look inside a binary file, with no idea of what they are looking for, only clues, such as the tools made the file, which compression method a file uses, the header bytes, or the author's name and/or email.

Let's say you accidentally deleted some JPEG files. If you knew the header bytes for JPEG files, you could use dd to find the deleted JPEG files. This command will display the header bytes:

```
dd if=/home/sam/JPEG_file.jpg | hexdump -C | less
```

The header bytes are always the first three bytes in the file. Every JPEG file has the same first three bytes. To see the footer bytes, while you still have the first command output on the screen, hit the End key on the keyboard. Less will move to the end of the JPEG file. The last three bytes are the footer bytes. Those are likewise the same for all JPEG files.

Once you know the header and footer bytes of JPEG files, you can search the drive for them using this command: `dd if=/dev/sda3 | hexdump -C | grep '<header bytes>' | '<footer bytes>'`

After you have an output of the offsets of the beginning and end of each deleted JPEG file, you can carve them out into files again, like so: Let's say grep returned JPEG header bytes at offset: `0xba0002f` and footer bytes at offset: `0xbaff02a`

You would convert the hex offsets to decimal offsets, using one of the many logic capable calculators for Linux. You would calculate 195 035 183 as the offset of the header bytes, and 196 079 658 as the offset of the footer bytes. If we subtract the smaller number from the larger one, we can get an idea of proper block size and count to use with dd. To find the proper count= figure: $195\ 035\ 183 - 196\ 079\ 658 = 1\ 044\ 475 / bs=4096 = 254.998$. That's really close to 255. If we could land exactly at the header bytes using `bs=4096`, we could use `count=255`. But I'm going to use `count=257`, because we'll start reading a little before the header bytes.

We need to use the `skip=` command to skip to the offset of the header bytes: $195\ 035\ 183 / bs=4096 = 47\ 606.011$. We always round down, so dd will start reading before the beginning of the file. In this case we round down to `skip=47 605`. Our command line to recover the deleted JPEG is then:

```
dd if=/dev/sda3 skip=47605 of=/home/sam/work_file.bin count=257 bs=4096
```

This gives us a file with the deleted JPEG inside of it, with some unwanted bytes before and after the part we want.

Open `work_file.bin` in a hex editor, and search for the `<header bytes>`. Delete everything before the first header byte. Search for the footer bytes. Delete everything after the footer bytes. Save the file as: `file.jpg`. View the file in an image viewing program, and it should look good. You can see why, if someone deleted 1,000 JPEG files accidentally, that they're not going to do this whole process to recover each one. There are more automated file carving programs. But this method really gets your hands dirty deep in digital data.

I put two identical drives in every one of my machines. Before I do anything that most probably spells disaster, like an untested command line in a root shell, that contains

```
find / -regex ?*.*?* -type f | xargs rm -f "$1"
```

```
I do: dcfldd if=/dev/sda of=/dev/sdb bs=4096 conv=notrunc,noerror
```

Linux DD

AwesomeMachine

and duplicate my present working /dev/sda drive system to the /dev/sdb drive. If I wreck the installation on sda, I boot from a live CD distro, and do:

```
dd if=/dev/sdb of=/dev/sda bs=4096 conv=notrunc,noerror
```

And I get everything back exactly the same it was before whatever daring maneuver I was trying didn't work. You can really, really learn Linux this way, because you can't wreck what you have an exact duplicate of. You also might consider making the root partition separate from /home, and make /home big enough to hold the root partition, plus more. Then, To make a backup of root:

```
dd if=/dev/sda2 (root) of=/home/sam/root.img bs=4096 conv=notrunc,noerror
```

To write the image of root back to the root partition, if you messed up and can't launch the X server, or edited /etc/fstab, and can't figure out what you did wrong. It only takes a few minutes to restore a 15 GB root partition from an image file:

```
dd if /home/sam/root.img of=/dev/sda2 (root) bs=4096 conv=notrunc,noerror
```

How to make a swap file, or another swapfile on a running system:

```
dd if=/dev/zero of=/swapspace bs=4k count=250000  
mkswap /swapspace  
swapon /swapspace
```

This can solve out of memory issues due to memory leaks on servers that cannot easily be rebooted.

How to pick proper block size:

```
dd if=/dev/zero bs=1024 count=1000000 of=/home/sam/1Gb.file  
dd if=/dev/zero bs=2048 count=500000 of=/home/sam/1Gb.file  
dd if=/dev/zero bs=4096 count=250000 of=/home/sam/1Gb.file  
dd if=/dev/zero bs=8192 count=125000 of=/home/sam/1Gb.file
```

This method can also be used as a drive benchmark, to find strengths and weaknesses in hard drives:

Read: `dd if=/home/sam/1Gb.file bs=64k | dd of=/dev/null`

Write: `dd if=/dev/zero bs=1024 count=1000000 of=/home/sam/1Gb.file`

When dd finishes it outputs (total size)/(total time). You get the idea.

Play with 'bs=' and 'count=', always having them multiply out to the same total size. You can calculate bytes/second like this: 1Gb/total seconds = Gb/s. You can get more realistic results using a 3Gb file.

Rejuvenate a Hard Drive

To cure input/output errors experienced when using dd. Over time the data on a drive, especially a drive that hasn't been used for a year or two, grows into larger magnetic flux points than were originally recorded. It becomes more difficult for the drive heads to decipher these magnetic flux points. This results in I/O errors. Sometimes sector 1 goes bad, resulting in a useless drive. Try:

```
dd if=/dev/sda of=/dev/sda
```

Linux DD

AwesomeMachine

to rejuvenate the drive. Rewrites all the data on the drive in nice tight magnetic patterns that can then be read properly. The procedure is safe and economical.

Make a file of 100 random bytes: `dd if=/dev/urandom of=/home/sam/myrandom bs=100 count=1`

/dev/random produces only as many random bits as the entropy pool contains. This yields quality randomness for cryptographic keys. If more random bytes are required, the process stops until the entropy pool is refilled (wagging your mouse helps). /dev/urandom does not have this restriction. If the user demands more bits than are currently in the entropy pool, it produces them using a pseudo random number generator. Here, /dev/urandom is the Linux random byte device. Myrandom is a file.

Randomize data over a file before deleting it: `ls -l` to find filesize. In this case it is 3769:

```
ls -l afile -rw----- ... 3769 Nov 2 13:41 <filename>
```

```
dd if=/dev/urandom of=afile bs=3769 count=1 conv=notrunc
```

duplicate a disk partition to a file on a different partition. Warning!! Do not write a partition image file to the same partition.

```
dd if=/dev/sdb2 of=/home/sam/partition.image bs=4096 conv=notrunc,noerror
```

This will make a file that is an exact duplicate of the sdb2 partition. You can substitute hdb, sda, hda, etc ... OR

```
dd if=/dev/sdb2 ibs=4096 | gzip > partition.image.gz conv=noerror
```

Makes a gzipped archive of the entire partition.

To restore use: `dd if=partition.image.gz | gunzip | dd of=/dev/sdb2`

For bzip2 (slower, smaller), substitute bzip2 and bunzip2, and name the file `< filename >.bz2`.

Restore a Disk Partition From An Image File

```
dd if=/home/sam/partition.image of=/dev/sdb2 bs=4096 conv=notrunc,noerror
```

Convert a File To Uppercase

```
dd if=filename of=filename conv=ucase
```

Make a Ramdrive

The Linux kernel makes a number a ramdisks you can make into ramdrives. You have to populate the drive with zeroes like so: `dd if=/dev/zero of=/dev/ram7 bs=1k count=16384` populates a 16 MB ramdisk.

`mke2fs -m0 /dev/ram7 4096` puts a file system on the ramdisk, turning it into a ramdrive. Watch this puppy smoke.

```
debian:/home/sam # hdparm -t /dev/ram7/dev/ram7:
```

Timing buffered disk reads: 16 MB in 0.02 seconds = 913.92 MB/sec

Linux DD

AwesomeMachine

You only need to do the timing once, because it's cool. Make the drive again, because hdparm is a little hard on ramdrives. You can mount the ramdrive with:

```
mkdir /mnt/mem
mount /dev/ram7 /mnt/mem
```

Now you can use the drive like a hard drive. This is particularly superb for working on large documents or programming. You can duplicate the large file or programming project to the ramdrive, which on my machine is at least 27 times as fast as /dev/sda, and every time you save the huge document, or need to do a compile, it's like your machine is running on nitromethane. The only drawback is data security. The ramdrive is volatile. If you lose power, or lock up, the data on the ramdrive is lost. Use a reliable machine during clear skies if you use a ramdrive.

Duplicate Ram Memory To A File

```
dd if=/dev/mem of=/home/sam/mem.bin bs=1024
```

The device /dev/mem is your system memory. You can actually duplicate any block or character device to a file using dd. Memory capture on a fast system, with bs=1024 takes about 60 seconds, a 120 GB HDD about an hour, a CD to hard drive about 10 minutes, a floppy to a hard drive about 2 minutes. With dd, your floppy drive images will not change. If you have a bootable DOS diskette, and you save it to your HDD as an image file, when you restore that image to another floppy it will be bootable.

Dd will print to the terminal window if you omit the of=/dev/output part.

```
dd if=/home/sam/myfile
```

 will print the file myfile to the terminal window.

To search the system memory:

```
dd if=/dev/mem | hexdump -C | grep 'some-string-of-words-in-the-file-you-forgot-to-save-before-the-power-failed'
```

If you need to cover your tracks quickly, put the following commands in a script to overwrite system ram with zeroes. Don't try this for fun.

```
mkdir /mnt/mem
mount -t ramfs /dev/mem /mnt/mem
dd if=/dev/zero > /mnt/mem/bigfile.file
```

This will overwrite all unprotected memory structures with zeroes, and freeze the machine so you have to reboot (Caution, this also prevents committment of the file system journal, and could trash the file system).

You can get arrested in 17 states for doing this next thing. Make an AES encrypted loop device:

```
dd if=/dev/urandom of=/home/sam/aes-drv bs=16065b count=100
modprobe loop
modprobe cryptoloop
modprobe aes
losetup -e aes /dev/loop1 ./aes-drv
password:
mkreiserfs /dev/loop1
mkdir /aes
mount -o loop,encryption=aes,acl ./aes-drv /aes
```

Linux DD AwesomeMachine

```
password:  
mv /home/sam/porno /aes
```

to get the porno on the aes drive image.

```
umount /aes  
losetup -d /dev/loop1  
rmmod aes  
rmmod cryptoloop  
rmmod loop
```

to make 'aes-driv' look like a 400 MB file of random bytes. Every time the lo interface is configured using losetup, according to the above, and the file 'aes-driv' is mounted, as above, the porno stash will be accessible in /aes/porno. You don't need to repeat the dd command, OR, the format with reiserfs, OR, the mv command. You only do those steps once. If you forget the password, there is no way to recover it besides guessing. Once the password is set, it can't be changed. To change the password, make a new file with the desired password, and move everything from the old file to the new file. Acl is a good mount option, because it allows use of acls. Otherwise your stuck with u,g,o and rwx.

If you are curious about what might be on you disk drive, or what an MBR looks like, or maybe what is at the very end of your disk:

```
dd if=/dev/sda count=1 | hexdump -C
```

Will show you sector 1, or the MBR. The bootstrap code and partition table are in the MBR. To see the end of the disk you have to know the total number of sectors, and the MAS must be set equal to the MNA. The helix CD has a utility to set this correctly. In the dd command, your skip value will be one less than MNA of the disk.

For a 120 GB Seagate SATA drives: `dd if=/dev/sda of=home/sam/myfile skip=234441646 bs=512`

So this reads sector for sector, and writes the last sector to myfile. Even with LBA addressing, disks still secretly are read in sectors, cylinders, and heads. There are 63 sectors per track, and 255 heads per cylinder. There is a total cylinder count.

```
512_bytes/sector*63_sectors/track*255heads=16065*512bytes/cylinder=8,225,280_bytes  
/cylinder
```

```
63_sectors/track*255_heads=sectors/cylinder
```

With 234441647 total sectors, and 16065 sectors per cylinder, you get some trailing sectors which do not make up an entire cylinder: `14593.317584812_cylinders/drive`.

This leaves 5102 sectors which cannot be partitioned, because to be in a partition you have to be a whole cylinder. It's like having part of a person. That doesn't really count as a person. These become surplus sectors after the last partition. You can't ordinarily read past the last partition. But dd can. It's a good idea to check for anything writing to surplus sectors. For our Seagate 120 GB drive:

```
234,441,647_sectors/drive - 5102_surplus_sectors = 234,436,545_partitionable  
sectors
```

```
dd if=/dev/sda of=/home/sam/myfile skip=234436545
```

Linux DD

AwesomeMachine

writes the last 5102 sectors to myfile. Launch midnight commander (mc) to view the file. If there is something in there, you do not need it for anything. In this case you would write over it with random characters: `dd if=/dev/urandom of=/dev/sda bs=512 seek=234436545` will overwrite the 5102 surplus sectors on our 120 GB Seagate drive.

Block Size

One cylinder in LBA mode =
 $255_heads * 63_sectors / track = 16065_sectors = 16065 * 512_bytes = 8,225,280_bytes$.

The b means '* 512'. 32130b represents a two cylinder block size. Cylinder block size always works to cover every sector in a partition, because partitions are made of a whole number of cylinders. One cylinder is 8,225,280 bytes. If you want to check out some random area of the disk:

```
dd if=/dev/sda of=/home/sam/myfile bs=4096 skip=2000 count=1000
```

Will give you 8,000 sectors in myfile, after the first 16,000 sectors. You can open that file with a hex editor, edit some of it, and write the edited part back to disk:

```
dd if=/home/sam/myfile of=/dev/sda bs=4096 seek=2000 count=1000
```

Image a Partition to Another Machine

On source machine: `dd if=/dev/hda bs=16065b | netcat < targethost-IP > 1234`

On target machine: `netcat -l -p 1234 | dd of=/dev/hdc bs=16065b`

Variations on target machine: `netcat -l -p 1234 | bzip2 > partition.img` makes a compressed image file using bzip2 compression. `netcat -l -p 1234 | gzip > partition.img` makes a compressed image file using gzip compression. I back up a 100 GB lappy disk on a desktop drive, over a lan connection, and the 100 GB compresses to about 4.0 GB. Most of the drive is empty, so it's mostly zeroes. Repetitive zeroes compress well.

Alert!! Don't hit enter yet. Hit enter on the target machine. THEN hit enter on the source machine.

Netcat is a program, available by default, on most linux installations. It's a networking swiss army knife. In the preceding example, netcat and dd are piped to one another. One of the functions of the linux kernel is to make pipes. The pipe character looks like two little lines on top of one another, both vertical. Here is how this command behaves: This byte size is a cylinder. `bs=16065b` equals one cylinder on an LBA drive. The dd command is piped to netcat, which takes as its arguments the IP address of the target (like 192.168.0.1, or any IP address with an open port) and what port you want to use (1234).

How to Make a Bootable CD from a Bootable Floppy

Put the floppy in the floppy drive, and don't mount it, YET.

```
dd if=/dev/fd0 of=/home/sam/floppy.img
```

Mount /home/sam/floppy.img:

```
mkdir /mnt/floppy.img  
mount -o loop /home/sam/floppy.img /mnt/floppy.img
```

Customize the image:

Linux DD

AwesomeMachine

```
cd /mnt/floppy.img/  
rm <files you don't need>  
cp /home/sam/bios.bin .
```

or any other files you want on the CD. But don't exceed the 1.44 MB size of a floppy.

Check space left in the mounted image: `df -h`

Unmount the floppy image:

```
cd ..  
umount /mnt/floppy.img
```

Make the .iso CD image file:

```
mkisofs -o /home/sam/floppy.img.iso -b /home/sam/floppy.img /home/sam/floppy.img
```

Burn the iso file to a CD:

```
wodim dev=/dev/hdc -sao driveropts=burnfree -dummy /home/sam/floppy.img.iso
```

This is a dummy burn, with the drive laser off. After you check the dummy run for errors, by looking at the program output, hit the up arrow, delete '-dummy', Enter.

You want to find out if your girlfriend is cheating on you, having cyber whoopie, or just misbehaving. Even if the computer is secured with a password you can boot with the <http://www.efense.com/helix> CD and search the entire drive partition for text strings:

```
dd if=/dev/sda2 bs=16065 | hexdump -C | grep 'I love you.'
```

... will search the whole drive partition for the text string specified between the single quotes.

Searching an entire disk partition several times can be quite tedious. This particular command string prints the search results to the screen, with the offset where it is located in the partition. Dd works in the decimal system. Hexdump works in hexadecimal. The output text string is at offset 0x020d0d90h.

You convert that to decimal with one of the many calculators found in Linux. This is decimal offset: 34,409,872. We want some manageable numbers, custom designed for speed and ease of use. The decimal disk offset is roughly 34 million, so the data we want to view is 34 MB into the partition. We divide 34,409,872 by some power of 2. Experience says 2^{13} is about what we want to get a quotient in the thousands. $34,409,872/8192 \approx 4200$. The data we want is 8,192 4,200 byte blocks, OR, 4,200 8,192 byte blocks, into the partition. We check: $4200 * 8192 = 34406400$; $34,409,872 - 34406400 = 3472$. This means the following command line will start reading 3,472 bytes before the search string location.

```
dd if=/dev/sda2 bs=4200 skip=8192 count=2 | hexdump -C > file.txt
```

... and finish reading approximately 4,200 bytes after the string. This will net you 3.4k of disk contents before the search string, and 4.2k after. That's a 7.6k chunk of disk, plenty for what we're doing. With this method you search all the deleted files, any chat activity, and emails. It works no matter what security is being employed on the machine. It works with NTFS, ext2, ext3, reiserfs, swap, and FAT partitions. But, it is illegal to use this method on a computer you aren't authorized to search. People

Linux DD

AwesomeMachine

can be sued, or imprisoned for performing unauthorized searches. On a related note, you can search system memory with this method, by substituting `/dev/mem` for `/dev/sda2`.

Write system memory to a CD. This is useful for documenting memory contents without contaminating the HDD. I recommend using a CD-RW so you can practice a little. This doesn't involve `dd`, but it's cool.

```
wodim /dev=/dev/scd0 -raw driveropts=burnfree /dev/mem
```

to find the cdwriter: `wodim --devices`

This method records raw, so you have to do a: `dd if=/dev/hdd | hexdump -C | less` to view the recorded memory. You can also employ the string search method above, substituting `/dev/hdd` for `/dev/sda2`.

```
dd if=/dev/hdd | hexdump -C | grep 'string'
```

string is any ascii sequence, hex sequence (must be separated with a space: `'55<space>aa<space>09'` searches for the hex string `'55aa09'`), list:

```
'[:alnum:]' any alphanumeric characters
'[:alpha:]' any alpha character
'[:digit:]' any numeric character
'[:blank:]' tabs and spaces
'[:lower:]' any lower case alpha characters
'[:upper:]' any uppercase alpha character
'[:cntrl:]' ASCII characters 000 thru 037, and 177 octal
'[:graph:]' [:alnum:] and [:punct:]
'[:punct:]' any punctuation character ` ! ' # $ % ' ( ) * + - . / : ; < = > ? @
[ \ ] ^ _ { | } ~
'[:space:]' tab, newline, vertical tab, form feed, carriage return, and space
'[:xdigit:]' any hex digit ranges('[a-d]' = any, or all abcd, '[0-9]' = any, or all 0123456789)
```

```
dd if=/dev/sda | hexdump -C | grep '[:punct:]' | less
```

... will return every line from the `hexdump -C` output that contains any punctuation characters specified above. It will not gather only punctuation characters.

Back up your MBR

```
dd if=/dev/sda of=mbr.bin count=1
```

Put this on a floppy you make with: `dd if=boot.img of=/dev/fd0`

I back up floppies to a HDD. Floppies don't last forever, so I do:

```
dd if=/dev/fd0 of=/home/sam/floppies/backup.bin conv=notrunc
```

If my floppy fails, I can make unlimited copies:

```
dd if=/home/sam/floppies/backup.bin of=/dev/fd0 conv=notrunc
```

Here is a command line to read your BIOS, and interfaces:

Linux DD

AwesomeMachine

```
dd if=/dev/mem bs=1k skip=768 count=256 2>/dev/null | strings -n 8
```

dd will not duplicate or erase an HPA, OR, host protected area. Dd will erase a disk completely, but not as well as using the hardware secure erase, security erase unit command.

Public Domain Copyright Material Begins Here

Note that sending a SIGUSR1 signal to a running 'dd' process makes it print to standard error the number of records read and written so far, then to resume copying.

```
$ dd if=/dev/zero of=/dev/null& pid=$!  
$ kill -USR1 $pid; sleep 1; kill $pid
```

```
10899206+0 records in 10899206+0 records out
```

BLOCKS and BYTES may be followed by the following multiplicative suffixes:

```
c 1, w 2, b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000, G  
1024*1024*1024
```

So, `dd if=/dev/sda of=/dev/sdb bs=1GB` will use one gigabyte block sizes.

`bs=4b` would give dd a block size of 4 disk sectors. 1 sector=512 bytes. `bs=4k` would indicate dd use a 4 kilobyte block size. I have found `bs=4k` to be the fastest for copying disk drives on a modern machine.

Operands

The following operands are supported:

`if=file` specifies the input path. Standard input is the default.

`of=file` specifies the output path. Standard output is the default.

`seek=blocks` skip this many blocks in the output file.

`ibs=n` specifies the input block size in n bytes (default is 512).

`obs=n` specifies the output block size in n bytes (default is 512).

If no conversion other than `sync`, `noerror`, and, `notrunc` is specified, each input block is copied to the output as a single block without aggregating short blocks.

`cbs=n` specifies the conversion block size for `block` and `unblock` in bytes by n (default is 0). If `cbs=` is omitted or given a value of 0, using `block` or `unblock` produces unspecified results. This option is used only if ASCII or EBCDIC conversion is specified.

`ascii` and `asciib` operands, the input is handled as described for the `unblock` operand except that characters are converted to ASCII before the trailing SPACE characters are deleted.

`ebcdic`, `ebcdicb`, `ibm`, and `ibmb` operands, the input is handled as described for the `block` operand except that the characters are converted to EBCDIC or IBM EBCDIC after the trailing SPACE characters are added.

Linux DD

AwesomeMachine

files=n copies and concatenates n input files before terminating (makes sense only where input is a magnetic tape or similar device).

skip=n skips n input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation reads the blocks or seeks past them. On non-seekable files, the blocks are read and the data is discarded.

iseek=n seeks n blocks from beginning of input file before copying (appropriate for disk files, where skip can be incredibly slow).

oseek=n seeks n blocks from beginning of output file before copying.

seek=n skips n blocks (using the specified output block size) from beginning of output file before copying. On non-seekable files, existing blocks are read and space from the current end-of-file to the specified offset, if any, is filled with null bytes. On seekable files, the implementation seeks to the specified offset or reads the blocks as described for non-seekable files.

count=n copies only n input blocks.

conv=value [,value. . .] Where values are comma-separated symbols from the following list:

conv=notrunc tells dd not to abbreviate blocks of all zero value, or multiple adjacent blocks of zeroes, with five asterisks (when you want to maintain size) Do not use **notrunc** for copying a larger volume to a smaller volume. Without **conv=notrunc** Do not truncate the output file.

ascii converts EBCDIC to ASCII.

asciib converts EBCDIC to ASCII using BSD-compatible character translations.

ebcdic converts ASCII to EBCDIC. If converting fixed-length ASCII records without NEWLINES, sets up a pipeline with **dd conv=unblock** beforehand.

ebcdicb converts ASCII to EBCDIC using BSD-compatible character translations. If converting fixed-length ASCII records without NEWLINES, sets up a pipeline with **dd conv=unblock** beforehand.

ibm slightly different map of ASCII to EBCDIC. If converting fixed-length ASCII records without NEWLINES, sets up a pipeline with **dd dd conv=unblock** beforehand.

ibmb slightly different map of ASCII to EBCDIC using BSD-compatible character translations. If converting fixed-length ASCII records without NEWLINES, sets up a pipeline with **dd conv=unblock** beforehand.

The **ascii** (or **asciib**), **ebcdic** (or **ebcdicb**), and **ibm** (or **ibmb**) values are mutually exclusive. **block** Treats the input as a sequence of NEWLINE-terminated or EOF-terminated variable-length records independent of the input block boundaries. Each record is converted to a record with a fixed length specified by the conversion block size. Any NEWLINE character is removed from the input line.

SPACE characters are appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size are truncated to the largest number of characters that will fit into that size. The number of truncated lines is reported. **unblock** Converts fixed-

Linux DD

AwesomeMachine

length records to variable length. Reads a number of bytes equal to the conversion block size (or the number of bytes remaining in the input, if less than the conversion block size), delete all trailing SPACE characters, and append a NEWLINE character. The block and unblock values are mutually exclusive.

lcase maps upper-case characters specified by the LC_CTYPE keyword to lower to the corresponding lower-case character. Characters for which no mapping is specified are not modified by this conversion.

ucase maps lower-case characters specified by the LC_CTYPE keyword to upper to the corresponding upper-case character. Characters for which no mapping is specified are not modified by this conversion. The lcase and ucase symbols are mutually exclusive.

swab swaps every pair of input bytes. If the current input record is an odd number of bytes, the last byte in the input record is ignored.

noerror does not stop processing on an input error. When an input error occurs, a diagnostic message is written on standard error, followed by the current input and output block counts in the same format as used at completion. If the **sync** conversion is specified, the missing input is replaced with null bytes and processed normally. Otherwise, the input block will be omitted from the output. **notrunc** Does not truncate the output file. Preserves blocks in the output file not explicitly written by this invocation of dd.

sync pads every input block to the size of the **ibs=** buffer, appending null bytes. (If either block or unblock is also specified, appends SPACE characters, rather than null bytes.)

Environment Variables

The following environment variables affect the messages and errors messages of dd:

LANG provides a default value for the internationalisation variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-dependent default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL if set to a non-empty string value, override the values of all the other internationalisation variables.

LC_CTYPE determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), the classification of characters as upper- or lower-case, and the mapping of characters from one case to the other.

LC_MESSAGES determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH determine the location of message catalogues for the processing of **LC_MESSAGES**.