

# dd

Sam Chessman

The dd command is one of the original Unix utilities and should be in everyone's tool box. It can strip headers, extract parts of binary files and write into the middle of floppy disks; it is used by the Linux kernel Makefiles to make boot images. It can be used to copy and convert magnetic tape formats, convert between ASCII and EBCDIC, swap bytes, and force to upper and lowercase.

For blocked I/O, the dd command has no competition in the standard tool set. One could write a custom utility to do specific I/O or formatting but, as dd is already available almost everywhere, it makes sense to use it.

Like most well-behaved commands, dd reads from its standard input and writes to its standard output, unless a command line specification has been given. This allows dd to be used in pipes, and remotely with the rsh remote shell command.

Unlike most commands, dd uses a keyword=value format for its parameters. This was reputedly modeled after IBM System/360 JCL, which had an elaborate DD "Dataset Definition" specification for I/O devices. A complete listing of all keywords is available from GNU dd with

```
dd --help
```

Some people believe dd means "Destroy Disk" or "Delete Data" because if it is misused, a partition or output file can be trashed very quickly. Since dd is the tool used to write disk headers, boot records, and similar system data areas, misuse of dd has probably trashed many hard disks and file systems.

In essence, dd copies and optionally converts data. It uses an input buffer, conversion buffer if conversion is specified, and an output buffer. Reads are issued to the input file or device for the size of the input buffer, optional conversions are applied, and writes are issued for the size of the output buffer. This allows I/O requests to be tailored to the requirements of a task. Output to standard error reports the number of full and short blocks read and written.

## Example 1

A typical task for dd is copying a floppy disk. As the common geometry of a 3.5" floppy is 18 sectors per track, two heads and 80 cylinders, an optimized dd command to read a floppy is:

### Example 1a: Copying from a 3.5" floppy

```
dd bs=2x80x18b if=/dev/fd0 of=/tmp/floppy.image1+0 records in1+0 records out
```

The 18b specifies 18 sectors of 512 bytes, the 2x multiplies the sector size by the number of heads, and the 80x is for the cylinders—a total of 1474560 bytes. This issues a single 1474560-byte read request to /dev/fd0 and a single 1474560 write request to /tmp/floppy.image, whereas a corresponding cp command:

```
cp /dev/fd0 /tmp/floppy.image
```

issues 360 reads and writes of 4096 bytes. While this may seem insignificant on a 1.44MB file, when larger amounts of data are involved, reducing the number of system calls and improving performance can be significant.

This example also shows the factor capability in the GNU dd number specification. This has been around since before the Programmers Work Bench and, while not documented in the GNU dd man page, is present in the source and works just fine, thank you.

# dd

Sam Chessman

To finish copying a floppy, the original needs to be ejected, a new diskette inserted, and another dd command issued to write to the diskette:

## Example 1b: Copying to a 3.5" floppy

```
dd bs=2x80x18b < /tmp/floppy.image > /dev/fd01+0 records in1+0 records out
```

Here is shown the stdin/stdout usage, in which respect dd is like most other utilities.

## Example 2

The original need for dd came with the 1/2" tapes used to exchange data with other systems and boot and install Unix on the PDP/11. Those days are gone, but the 9-track format lives. To access the venerable 9-track, 1/2" tape, dd is superior. With modern SCSI tape devices, blocking and unblocking are no longer a necessity, as the hardware reads and writes 512-byte data blocks.

However, the 9-track 1/2" tape format allows for variable length blocking and can be impossible to read with the cp command. The dd command allows for the exact specification of input and output block sizes, and can even read variable length block sizes, by specifying an input buffer size larger than any of the blocks on the tape. Short blocks are read, and dd happily copies those to the output file without complaint, simply reporting on the number of complete and short blocks encountered.

Then there are the EBCDIC datasets transferred from such systems as MVS, which are almost always 80-character blank-padded Hollerith Card Images! No problem for dd, which will convert these to newline-terminated variable record length ASCII. Making the format is just as easy and dd again is the right tool for the job.

## Example 2: Converting EBCDIC 80-character fixed-length record to ASCII variable-length newline-terminated record

```
dd bs=10240 cbs=80 conv=ascii,unblock if=/dev/st0 of=ascii.out40+0 records in38+1 records out
```

The fixed record length is specified by the cbs=80 parameter, and the input and output block sizes are set with bs=10240. The EBCDIC-to-ASCII conversion and fixed-to-variable record length conversion are enabled with the conv=ascii,noblock parameter.

Notice the output record count is smaller than the input record count. This is due to the padding spaces eliminated from the output file and replaced with newline characters.

## Example 3

Sometimes data arrives from sources in unusual formats. For example, every time I read a tape made on an SGI machine, the bytes are swapped. The dd command takes this in stride, swapping the bytes as required. The ability to use dd in a pipe with rsh means that the tape device on any \*nix system is accessible, given the proper rlogin setup.

## Example 3: Byte Swapping with Remote Access of Magnet Tape:rsh sgi.with.tape

```
dd bs=256b if=/dev/rmt0 conv=swab | tar xvf -
```

# dd

## Sam Chessman

The dd runs on the SGI and swaps the bytes before writing to the tar command running on the local host.

### Example 4

Murphy's Law was postulated long before digital computers, but it seems it was specifically targeted for them. When you need to read a floppy or tape, it is the only copy in the universe and you have a deadline past due, that is when you will have a bad spot on the magnetic media, and your data will be unreadable. To the rescue comes dd, which can read all the good data around the bad spot and continue after the error is encountered. Sometimes this is all that is needed to recover the important data.

### Example 4: Error Handlingi

```
dd bs=265b conv=noerror if=/dev/st0 of=/tmp/bad.tape.image
```

### Example 5

The Linux kernel Makefiles use dd to build the boot image. In the Alpha Makefile /usr/src/linux/arch/alpha/boot/Makefile, the srmbot target issues the command:

### Example 5. Kernel Image Makefile:

```
dd if=bootimage of=$(BOOTDEV) bs=512 seek=1 skip=1
```

This skips the first 512 bytes of the input bootimage file (skip=1) and writes starting at the second sector of the \$(BOOTDEV) device (seek=1). A typical use of dd is to skip executable headers and begin writing in the middle of a device, skipping volume and partition data. As this can cause your disk to lose file system data, please test and use these applications with care.

### Credits

The dd command has been around since the 1970s, ported to many systems, rewritten many times, and tested by time as a useful tool. The current Linux version is GNU dd GNU fileutils 3.12, written by Paul Rubin, David MacKenzie, and Stuart Kemp, Copyright © 1985, 1990, 1991 Free Software Foundation, Inc.

GNU dd is found in the fileutils collection, with the current version at the URL <ftp://prep.ai.mit.edu/pub/gnu/fileutils-3.12.tar.gz> or a mirror near you.

Other major versions include SYSV and BSD, with the BSD source version 5.16 4/28/93 derived from software contributed to Berkeley by Keith Muller of the University of California, San Diego and Lance Visser of Convex Computer Corporation, Copyright © 1991 The Regents of the University of California.