```
**********************************************************
   43116 Fri Apr 20 10:14:27 2012
new/usr/src/cmd/dd/dd.c
2556 dd should accept M,G,T,... for sizes
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012, Josef 'Jeff' Sipek <jeffpc@31bits.net>. All rights reserved.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 *      convert and copy
 */

#include        <stdio.h>
#include        <signal.h>
#include        <fcntl.h>
#include        <sys/param.h>
#include        <sys/types.h>
#include        <sys/sysmacros.h>
#include        <sys/stat.h>
#include        <unistd.h>
#include        <stdlib.h>
#include        <locale.h>
#include        <string.h>

/* The BIG parameter is machine dependent.  It should be a long integer */
/* constant that can be used by the number parser to check the validity */
/* of numeric parameters.  On 16-bit machines, it should probably be    */
/* the maximum unsigned integer, 0177777L.  On 32-bit machines where    */
/* longs are the same size as ints, the maximum signed integer is more  */
/* appropriate.  This value is 017777777777L. In 64 bit environments,   */
/* the maximum signed integer value is 07777777777777777777LL        */

#define BIG     07777777777777777777LL

#define BSIZE   512
```



```
/* Option parameters */

#define COPY            0       /* file copy, preserve input block size */
#define REBLOCK         1       /* file copy, change block size */
#define LCREBLOCK       2       /* file copy, convert to lower case */
#define UCREBLOCK       3       /* file copy, convert to upper case */
#define NBASCII         4       /* file copy, convert from EBCDIC to ASCII */
#define LCNBASCII       5       /* file copy, EBCDIC to lower case ASCII */
#define UCNBASCII       6       /* file copy, EBCDIC to upper case ASCII */
#define NBEBCDIC        7       /* file copy, convert from ASCII to EBCDIC */
#define LCNBEBCDIC      8       /* file copy, ASCII to lower case EBCDIC */
#define UCNBEBCDIC      9       /* file copy, ASCII to upper case EBCDIC */
#define NBIBM           10      /* file copy, convert from ASCII to IBM */
#define LCNBIBM         11      /* file copy, ASCII to lower case IBM */
#define UCNBIBM         12      /* file copy, ASCII to upper case IBM */
#define UNBLOCK         13      /* convert blocked ASCII to ASCII */
#define LCUNBLOCK       14      /* convert blocked ASCII to lower case ASCII */
#define UCUNBLOCK       15      /* convert blocked ASCII to upper case ASCII */
#define ASCII           16      /* convert blocked EBCDIC to ASCII */
#define LCASCII         17      /* convert blocked EBCDIC to lower case ASCII */
#define UCASCII         18      /* convert blocked EBCDIC to upper case ASCII */
#define BLOCK           19      /* convert ASCII to blocked ASCII */
#define LCBLOCK         20      /* convert ASCII to lower case blocked ASCII */
#define UCBLOCK         21      /* convert ASCII to upper case blocked ASCII */
#define EBCDIC          22      /* convert ASCII to blocked EBCDIC */
#define LCEBCDIC        23      /* convert ASCII to lower case blocked EBCDIC */
#define UCEBCDIC        24      /* convert ASCII to upper case blocked EBCDIC */
#define IBM             25      /* convert ASCII to blocked IBM */
#define LCIBM           26      /* convert ASCII to lower case blocked IBM */
#define UCIBM           27      /* convert ASCII to upper case blocked IBM */
#define LCASE           01      /* flag - convert to lower case */
#define UCASE           02      /* flag - convert to upper case */
#define SWAB            04      /* flag - swap bytes before conversion */
#define NERR            010     /* flag - proceed on input errors */
#define SYNC            020     /* flag - pad short input blocks with nulls */
#define BADLIMIT        5       /* give up if no progress after BADLIMIT trys */
#define SVR4XLATE       0       /* use default EBCDIC translation */
#define BSDXLATE        1       /* use BSD-compatible EBCDIC translation */

#define USAGE\
        "usage: dd [if=file] [of=file] [ibs=n|nk|nb|nxm] [obs=n|nk|nb|nxm]\n"\
        "          [bs=n|nk|nb|nxm] [cbs=n|nk|nb|nxm] [files=n] [skip=n]\n"\
        "          [iseek=n] [oseek=n] [seek=n] [count=n] [conv=[ascii]\n"\
        "          [,ebcdic][,ibm][,asciib][,ebcdicb][,ibmb]\n"\
        "          [,block|unblock][,lcase|ucase][,swab]\n"\
        "          [,noerror][,notrunc][,sync]]\n"

/* Global references */

/* Local routine declarations */

static int      match(char *);
static void             term();
static unsigned long long       number();
static unsigned char    *flsh();
static void             stats();

/* Local data definitions */

static unsigned ibs;    /* input buffer size */
static unsigned obs;    /* output buffer size */
static unsigned bs;     /* buffer size, overrules ibs and obs */
static unsigned cbs;    /* conversion buffer size, used for block conversions */
static unsigned ibc;    /* number of bytes still in the input buffer */
static unsigned obc;    /* number of bytes in the output buffer */
static unsigned cbc;    /* number of bytes in the conversion buffer */
```

```
129 static int      ibf;    /* input file descriptor */
130 static int      obf;    /* output file descriptor */
131 static int      cflag;  /* conversion option flags */
132 static int      skipf;  /* if skipf == 1, skip rest of input line */
133 static unsigned long long       nifr;   /* count of full input records */
134 static unsigned long long       nipr;   /* count of partial input records */
135 static unsigned long long       nofr;   /* count of full output records */
136 static unsigned long long       nopr;   /* count of partial output records */
137 static unsigned long long       ntrunc; /* count of truncated input lines */
138 static unsigned long long       nbad;   /* count of bad records since last */
139                                         /* good one */
140 static int      files;  /* number of input files to concatenate (tape only) */
141 static off_t    skip;   /* number of input records to skip */
142 static off_t    iseekn; /* number of input records to seek past */
143 static off_t    oseekn; /* number of output records to seek past */
144 static unsigned long long       count;  /* number of input records to copy */
145                         /* (0 = all) */
146 static int      trantype; /* BSD or SVr4 compatible EBCDIC */

148 static char             *string;        /* command arg pointer */
149 static char             *ifile;         /* input file name pointer */
150 static char             *ofile;         /* output file name pointer */
151 static unsigned char    *ibuf;          /* input buffer pointer */
152 static unsigned char    *obuf;          /* output buffer pointer */

154 /* This is an EBCDIC to ASCII conversion table  */
155 /* from a proposed BTL standard April 16, 1979  */

157 static unsigned char svr4_etoa [] =
158 {
159         0000, 0001, 0002, 0003, 0234, 0011, 0206, 0177,
160         0227, 0215, 0216, 0013, 0014, 0015, 0016, 0017,
161         0020, 0021, 0022, 0023, 0235, 0205, 0010, 0207,
162         0030, 0031, 0222, 0217, 0034, 0035, 0036, 0037,
163         0200, 0201, 0202, 0203, 0204, 0012, 0027, 0033,
164         0210, 0211, 0212, 0213, 0214, 0005, 0006, 0007,
165         0220, 0221, 0026, 0223, 0224, 0225, 0226, 0004,
166         0230, 0231, 0232, 0233, 0024, 0025, 0236, 0032,
167         0040, 0240, 0241, 0242, 0243, 0244, 0245, 0246,
168         0247, 0250, 0325, 0056, 0074, 0050, 0053, 0174,
169         0046, 0251, 0252, 0253, 0254, 0255, 0256, 0257,
170         0260, 0261, 0041, 0044, 0052, 0051, 0073, 0176,
171         0055, 0057, 0262, 0263, 0264, 0265, 0266, 0267,
172         0270, 0271, 0313, 0054, 0045, 0137, 0076, 0077,
173         0272, 0273, 0274, 0275, 0276, 0277, 0300, 0301,
174         0302, 0140, 0072, 0043, 0100, 0047, 0075, 0042,
175         0303, 0141, 0142, 0143, 0144, 0145, 0146, 0147,
176         0150, 0151, 0304, 0305, 0306, 0307, 0310, 0311,
177         0312, 0152, 0153, 0154, 0155, 0156, 0157, 0160,
178         0161, 0162, 0136, 0314, 0315, 0316, 0317, 0320,
179         0321, 0345, 0163, 0164, 0165, 0166, 0167, 0170,
180         0171, 0172, 0322, 0323, 0324, 0133, 0326, 0327,
181         0330, 0331, 0332, 0333, 0334, 0335, 0336, 0337,
182         0340, 0341, 0342, 0343, 0344, 0135, 0346, 0347,
183         0173, 0101, 0102, 0103, 0104, 0105, 0106, 0107,
184         0110, 0111, 0350, 0351, 0352, 0353, 0354, 0355,
185         0175, 0112, 0113, 0114, 0115, 0116, 0117, 0120,
186         0121, 0122, 0356, 0357, 0360, 0361, 0362, 0363,
187         0134, 0237, 0123, 0124, 0125, 0126, 0127, 0130,
188         0131, 0132, 0364, 0365, 0366, 0367, 0370, 0371,
189         0060, 0061, 0062, 0063, 0064, 0065, 0066, 0067,
190         0070, 0071, 0372, 0373, 0374, 0375, 0376, 0377,
191 };
```

*unchanged_portion_omitted_*


```
1639 /* number ************************************************************ */
1640 /*                                                                     */
1641 /* Convert a numeric arg to binary                                     */
1642 /*                                                                     */
1643 /* Arg:         big - maximum valid input number                       */
1644 /* Global arg:  string - pointer to command arg                        */
1645 /*                                                                     */
1646 /* Valid forms: 123 | 123k | 123M | 123G | 123T | 123P | 123E | 123Z |  */
1647 /*              123w | 123b | 123*123 | 123x123                       */
1645 /* Valid forms: 123 | 123k | 123w | 123b | 123*123 | 123x123          */
1648 /*              plus combinations such as 2b*3kw*4w                    */
1649 /*                                                                     */
1650 /* Return:      converted number                                       */
1651 /*                                                                     */
1652 /* ****************************************************************** */

1654 static unsigned long long
1655 number(big)
1656 long long big;
1657 {
1658         char *cs;
1659         long long n;
1660         long long cut = BIG / 10;       /* limit to avoid overflow */

1662         cs = string;
1663         n = 0;
1664         while ((*cs >= '0') && (*cs <= '9') && (n <= cut))
1665         {
1666                 n = n*10 + *cs++ - '0';
1667         }
1668         for (;;)
1669         {
1670                 switch (*cs++)
1671                 {

1673                 case 'Z':
1674                         n *= 1024;
1675                         /* FALLTHROUGH */

1677                 case 'E':
1678                         n *= 1024;
1679                         /* FALLTHROUGH */

1681                 case 'P':
1682                         n *= 1024;
1683                         /* FALLTHROUGH */

1685                 case 'T':
1686                         n *= 1024;
1687                         /* FALLTHROUGH */

1689                 case 'G':
1690                         n *= 1024;
1691                         /* FALLTHROUGH */

1693                 case 'M':
1694                         n *= 1024;
1695                         /* FALLTHROUGH */

1697                 case 'k':
1698                         n *= 1024;
1699                         continue;

1701                 case 'w':
1702                         n *= 2;
1703                         continue;
```

```
1705                 case 'b':
1706                         n *= BSIZE;
1707                         continue;

1709                 case '*':
1710                 case 'x':
1711                         string = cs;
1712                         n *= number(BIG);

1714                 /* FALLTHROUGH */
1715                 /* Fall into exit test, recursion has read rest of string */
1716                 /* End of string, check for a valid number */

1718                 case '\0':
1719                         if ((n > big) || (n < 0))
1720                         {
1721                                 (void) fprintf(stderr, "dd: %s \"%llu\"\n",
1722                                         gettext("argument out of range:"), n);
1723                                 exit(2);
1724                         }
1725                         return (n);

1727                 default:
1728                         (void) fprintf(stderr, "dd: %s \"%s\"\n",
1729                                 gettext("bad numeric argument:"), string);
1730                         exit(2);
1731                 }
1732         } /* never gets here */
1733 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   16805 Fri Apr 20 10:14:27 2012
new/usr/src/man/man1m/dd.1m
2556 dd should accept M,G,T,... for sizes
**********************************************************
'\" te
.\" Copyright (c) 1992, X/Open Company Limited  All Rights Reserved
.\" Copyright 1989 AT&T
.\" Portions Copyright (c) 1995, Sun Microsystems, Inc.  All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH DD 1M "Sep 16, 1996"
.SH NAME
dd \- convert and copy a file
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/dd\fR [\fIoperand=value\fR]...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBdd\fR utility copies the specified input file to the specified output
with possible conversions. The standard input and output are used by default.
The input and output block sizes may be specified to take advantage of raw
physical I/O. Sizes are specified in bytes; a number may end with \fBk\fR,
\fBb\fR, or \fBw\fR to specify multiplication by 1024, 512, or 2, respectively.
Numbers may also be separated by \fBx\fR to indicate multiplication.
.sp
.LP
The \fBdd\fR utility reads the input one block at a time, using the specified
input block size. \fBdd\fR then processes the block of data actually returned,
which could be smaller than the requested block size. \fBdd\fR applies any
conversions that have been specified and writes the resulting data to the
output in blocks of the specified output block size.
.sp
.LP
\fBcbs\fR is used only if \fBascii\fR, \fBasciib\fR, \fBunblock\fR,
\fBebcdic\fR, \fBebcdicb\fR, \fBibm\fR, \fBibmb\fR, or \fBblock\fR conversion
is specified. In the first two cases, \fBcbs\fR characters are copied into the
conversion buffer, any specified character mapping is done, trailing blanks are
trimmed, and a \fBNEWLINE\fR is added before sending the line to output. In the
last three cases, characters up to \fBNEWLINE\fR are read into the conversion
buffer and blanks are added to make up an output record of size \fBcbs\fR.
\fBASCII\fR files are presumed to contain \fBNEWLINE\fR characters. If
\fBcbs\fR is unspecified or \fB0\fR, the \fBascii\fR, \fBasciib\fR,
\fBebcdic\fR, \fBebcdicb\fR, \fBibm\fR, and \fBibmb\fR options convert the
character set without changing the input file's block structure. The
\fBunblock\fR and \fBblock\fR options become a simple file copy.
.sp
.LP
After completion, \fBdd\fR reports the number of whole and partial input and
output blocks.
.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
```



```
\fB\fBif=\fR\fIfile\fR\fR
.ad
.sp .6
.RS 4n
Specifies the input path. Standard input is the default.
.RE

.sp
.ne 2
.na
\fB\fBof=\fR\fIfile\fR\fR
.ad
.sp .6
.RS 4n
Specifies the output path. Standard output is the default. If the
\fBseek=\fR\fBexpr\fR conversion is not also specified, the output file will be
truncated before the copy begins, unless \fBconv=notrunc\fR is specified. If
\fBseek=\fR\fBexpr\fR is specified, but \fBconv=notrunc\fR is not, the effect
of the copy will be to preserve the blocks in the output file over which
\fBdd\fR seeks, but no other portion of the output file will be preserved. (If
the size of the seek plus the size of the input file is less than the previous
size of the output file, the output file is shortened by the copy.)
.RE

.sp
.ne 2
.na
\fB\fBibs=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Specifies the input block size in \fIn\fR bytes (default is \fB512\fR).
.RE

.sp
.ne 2
.na
\fB\fBobs=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Specifies the output block size in \fIn\fR bytes (default is \fB512\fR).
.RE

.sp
.ne 2
.na
\fB\fBbs=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Sets both input and output block sizes to \fIn\fR bytes, superseding \fBibs=\fR
and \fBobs=\fR. If no conversion other than \fBsync\fR,\fB noerror\fR, and
\fBnotrunc\fR is specified, each input block is copied to the output as a
single block without aggregating short blocks.
.RE

.sp
.ne 2
.na
\fB\fBcbs=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Specifies the conversion block size for \fBblock\fR and \fBunblock\fR in bytes
by \fIn\fR (default is \fB0\fR). If \fBcbs=\fR is omitted or given a value of
```

\fB0\fR, using \fBblock\fR or \fBunblock\fR produces unspecified results.
.sp
This option is used only if \fBASCII\fR or \fBEBCDIC\fR conversion is specified. For the \fBascii\fR and \fBasciib\fR operands, the input is handled as described for the \fBunblock\fR operand except that characters are converted to \fBASCII\fR before the trailing \fBSPACE\fR characters are deleted. For the \fBebcdic\fR, \fBebcdicb\fR, \fBibm\fR, and \fBibmb\fR operands, the input is handled as described for the \fBblock\fR operand except that the characters are converted to \fBEBCDIC\fR or IBM \fBEBCDIC\fR after the trailing \fBSPACE\fR characters are added.
.RE

.sp
.ne 2
.na
\fB\fBfiles=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Copies and concatenates \fIn\fR input files before terminating (makes sense only where input is a magnetic tape or similar device).
.RE

.sp
.ne 2
.na
\fB\fBskip=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Skips \fIn\fR 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.
.RE

.sp
.ne 2
.na
\fB\fBiseek=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Seeks \fIn\fR blocks from beginning of input file before copying (appropriate for disk files, where \fBskip\fR can be incredibly slow).
.RE

.sp
.ne 2
.na
\fB\fBoseek=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Seeks \fIn\fR blocks from beginning of output file before copying.
.RE

.sp
.ne 2
.na
\fB\fBseek=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Skips \fIn\fR 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.
.RE

.sp
.ne 2
.na
\fB\fBcount=\fR\fIn\fR\fR
.ad
.sp .6
.RS 4n
Copies only \fIn\fR input blocks.
.RE

.sp
.ne 2
.na
\fB\fBconv=\fR\fIvalue\fR[\fB,\fR\fIvalue\fR.\|.\|.\|]\fR
.ad
.sp .6
.RS 4n
Where \fIvalue\fRs are comma-separated symbols from the following list:
.sp
.ne 2
.na
\fB\fBascii\fR\fR
.ad
.RS 11n
Converts \fBEBCDIC\fR to \fBASCII\fR.
.RE

.sp
.ne 2
.na
\fB\fBasciib\fR\fR
.ad
.RS 11n
Converts \fBEBCDIC\fR to \fBASCII\fR using \fBBSD\fR-compatible character translations.
.RE

.sp
.ne 2
.na
\fB\fBebcdic\fR\fR
.ad
.RS 11n
Converts \fBASCII\fR to \fBEBCDIC\fR. If converting fixed-length \fBASCII\fR records without NEWLINEs, sets up a pipeline with \fBdd conv=unblock\fR beforehand.
.RE

.sp
.ne 2
.na
\fB\fBebcdicb\fR\fR
.ad
.RS 11n
Converts \fBASCII\fR to \fBEBCDIC\fR using \fBBSD\fR-compatible character translations. If converting fixed-length \fBASCII\fR records without \fBNEWLINE\fRs, sets up a pipeline with \fBdd conv=unblock\fR beforehand.
.RE

.sp
.ne 2

```
.na
\fB\fBibm\fR\fR
.ad
.RS 11n
Slightly different map of \fBASCII\fR to \fBEBCDIC\fR. If converting
fixed-length \fBASCII\fR records without \fBNEWLINE\fRs, sets up a pipeline
with \fBdd conv=unblock\fR beforehand.
.RE

.sp
.ne 2
.na
\fB\fBibmb\fR\fR
.ad
.RS 11n
Slightly different map of \fBASCII\fR to \fBEBCDIC\fR using
\fBBSD\fR-compatible character translations. If converting fixed-length
\fBASCII\fR records without \fBNEWLINE\fRs, sets up a pipeline with \fBdd
conv=unblock\fR beforehand.
.RE

The \fBascii\fR (or \fBasciib\fR), \fBebcdic\fR (or \fBebcdicb\fR), and
\fBibm\fR (or \fBibmb\fR) values are mutually exclusive.
.sp
.ne 2
.na
\fB\fBblock\fR\fR
.ad
.RS 11n
Treats the input as a sequence of \fBNEWLINE\fR-terminated or
\fBEOF\fR-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 \fBNEWLINE\fR character is removed from the
input line. \fBSPACE\fR 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.
.RE

.sp
.ne 2
.na
\fB\fBunblock\fR\fR
.ad
.RS 11n
Converts fixed-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 \fBSPACE\fR
characters, and append a \fBNEWLINE\fR character.
.RE

The  \fBblock\fR and \fBunblock\fR values are mutually exclusive.
.sp
.ne 2
.na
\fB\fBlcase\fR\fR
.ad
.RS 9n
Maps upper-case characters specified by the \fBLC_CTYPE\fR keyword
\fBtolower\fR to the corresponding lower-case character. Characters for which
no mapping is specified are not modified by this conversion.
.RE

.sp
.ne 2
.na
```



```
\fB\fBucase\fR\fR
.ad
.RS 9n
Maps lower-case characters specified by the \fBLC_CTYPE\fR keyword
\fBtoupper\fR to the corresponding upper-case character. Characters for which
no mapping is specified are not modified by this conversion.
.RE

The \fBlcase\fR and \fBucase\fR symbols are mutually exclusive.
.sp
.ne 2
.na
\fB\fBswab\fR\fR
.ad
.RS 11n
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.
.RE

.sp
.ne 2
.na
\fB\fBnoerror\fR\fR
.ad
.RS 11n
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
\fBsync\fR conversion is specified, the missing input is replaced with null
bytes and processed normally. Otherwise, the input block will be omitted from
the output.
.RE

.sp
.ne 2
.na
\fB\fBnotrunc\fR\fR
.ad
.RS 11n
Does not truncate the output file. Preserves blocks in the output file not
explicitly written by this invocation of \fBdd\fR. (See also the preceding
\fBof=\fR\fIfile\fR operand.)
.RE

.sp
.ne 2
.na
\fB\fBsync\fR\fR
.ad
.RS 11n
Pads every input block to the size of the \fBibs=\fR buffer, appending null
bytes. (If either \fBblock\fR or \fBunblock\fR is also specified, appends
\fBSPACE\fR characters, rather than null bytes.)
.RE

.RE

.sp
.LP
If operands other than \fBconv=\fR are specified more than once, the last
specified \fBoperand=\fR\fIvalue\fR is used.
.sp
.LP
For the \fBbs=\fR, \fBcbs=\fR, \fBibs=\fR, and \fBobs=\fR operands, the
application must supply an expression specifying a size in bytes. The
expression, \fBexpr\fR, can be:
```

```
.RS +4
.TP
1.
a positive decimal number
.RE
.RS +4
.TP
2.
a positive decimal number followed by \fBk\fR, specifying multiplication by
1024
.RE
.RS +4
.TP
3.
a positive decimal number followed by \fBM\fR, specifying multiplication by
1024*1024
.RE
.RS +4
.TP
4.
a positive decimal number followed by \fBG\fR, specifying multiplication by
1024*1024*1024
.RE
.RS +4
.TP
5.
a positive decimal number followed by \fBT\fR, specifying multiplication by
1024*1024*1024*1024
.RE
.RS +4
.TP
6.
a positive decimal number followed by \fBP\fR, specifying multiplication by
1024*1024*1024*1024*1024
.RE
.RS +4
.TP
7.
a positive decimal number followed by \fBE\fR, specifying multiplication by
1024*1024*1024*1024*1024*1024
.RE
.RS +4
.TP
8.
a positive decimal number followed by \fBZ\fR, specifying multiplication by
1024*1024*1024*1024*1024*1024*1024
.RE
.RS +4
.TP
9.
a positive decimal number followed by \fBb\fR, specifying multiplication by
512
.RE
.RS +4
.TP
10.
a positive decimal number followed by \fBk\fR or \fBb\fR)
two or more positive decimal numbers (with or without \fBk\fR or \fBb\fR)
separated by \fBx\fR, specifying the product of the indicated values.
.RE
.sp
.LP
All of the operands will be processed before any input is read.
.SH USAGE
.sp
.LP
```



```
See \fBlargefile\fR(5) for the description of the behavior of \fBdd\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH EXAMPLES
.LP
\fBExample 1 \fRCopying from one tape drive to another
.sp
.LP
The following example copies from tape drive \fB0\fR to tape drive \fB1\fR,
using a common historical device naming convention.

.sp
.in +2
.nf
example% \fBdd if=/dev/rmt/0h  of=/dev/rmt/1h\fR
.fi
.in -2
.sp

.LP
\fBExample 2 \fRStripping the first 10 bytes from standard input
.sp
.LP
The following example strips the first 10 bytes from standard input:

.sp
.in +2
.nf
example% \fBdd ibs=10  skip=1\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRReading a tape into an ASCII file
.sp
.LP
This example reads an \fBEBCDIC\fR tape blocked ten 80-byte \fBEBCDIC\fR card
images per block into the \fBASCII\fR file \fBx\fR:

.sp
.in +2
.nf
example% \fBdd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase\fR
.fi
.in -2
.sp

.LP
\fBExample 4 \fRUsing conv=sync to write to tape
.sp
.LP
The following example uses \fBconv=sync\fR when writing to a tape:

.sp
.in +2
.nf
example% \fBtar cvf - . | compress | dd obs=1024k of=/dev/rmt/0 conv=sync\fR
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBdd\fR: \fBLANG\fR, \fBLC_ALL\fR,
```

```
523 \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
524 .SH EXIT STATUS
525 .sp
526 .LP
527 The following exit values are returned:
528 .sp
529 .ne 2
530 .na
531 \fB\fB0\fR\fR
532 .ad
533 .RS 6n
534 The input file was copied successfully.
535 .RE

537 .sp
538 .ne 2
539 .na
540 \fB\fB>0\fR\fR
541 .ad
542 .RS 6n
543 An error occurred.
544 .RE

546 .sp
547 .LP
548 If an input error is detected and the \fBnoerror\fR conversion has not been
549 specified, any partial output block will be written to the output file, a
550 diagnostic message will be written, and the copy operation will be
551 discontinued. If some other error is detected, a diagnostic message will be
552 written and the copy operation will be discontinued.
553 .SH ATTRIBUTES
554 .sp
555 .LP
556 See \fBattributes\fR(5) for descriptions of the following attributes:
557 .sp

559 .sp
560 .TS
561 box;
562 c | c
563 l | l .
564 ATTRIBUTE TYPE  ATTRIBUTE VALUE
565 _
566 Interface Stability     Standard
567 .TE

569 .SH SEE ALSO
570 .sp
571 .LP
572 \fBcp\fR(1), \fBsed\fR(1), \fBtr\fR(1), \fBattributes\fR(5), \fBenviron\fR(5),
573 \fBlargefile\fR(5), \fBstandards\fR(5)
574 .SH DIAGNOSTICS
575 .sp
576 .ne 2
577 .na
578 \fB\fBf+p records in(out)\fR\fR
579 .ad
580 .RS 23n
581 numbers of full and partial blocks read(written)
582 .RE

584 .SH NOTES
585 .sp
586 .LP
587 Do not use \fBdd\fR to copy files between file systems having different block
588 sizes.
```



```
589 .sp
590 .LP
591 Using a  blocked device to copy a file will result in extra nulls being added
592 to the file to pad the final block to the block boundary.
593 .sp
594 .LP
595 When  \fBdd\fR reads from a pipe, using the  \fBibs=X\fR and  \fBobs=Y\fR
596 operands, the output will always be blocked in chunks of size Y. When
597 \fBbs=Z\fR is used, the output blocks will be whatever was available to be read
598 from the pipe at the time.
599 .sp
600 .LP
601 When using \fBdd\fR to copy files to a tape device, the file size must be a
602 multiple of the device sector size (for example, 512 Kbyte).  To  copy files of
603 arbitrary size to a tape device, use  \fBtar\fR(1) or  \fBcpio\fR(1).
604 .sp
605 .LP
606 For \fBSIGINT\fR, \fBdd\fR writes status information to standard error before
607 exiting. It takes the standard action for all other signals.
```