NAME

bdd – Bit stream copy and modify

SYNOPSIS

bdd [OPTIONS]

DESCRIPTION

bdd modifies information in bit streams (non byte-aligned).

By default, bdd reads bytes from standard input and writes them to standard output. With options, the behaviour can be widely modified to handle inputs and outputs with accuracy of one bit.

bdd considers input and output as streams of bit units. A bit unit is a repeatable units of constant amount of bits. Those units can be in the middle of the stream and the input stream can be infinite (like /dev/zero).

Every unit is divided to bit-defined fields and fields are interpreted into a tuple (analogy of relational database). Tuples may have integers, floats and strings.

Tuple can be modified and packed again bit by bit into a unit. Units are written into the output bit stream.

There can be anothern input stream, merge stream. Units in the merge stream are output after every main input stream unit. Merge stream does not support division into fields.

The name bdd comes from binary-dd because it is able to do much of the operations anyone would need in dd to handle bit streams.

OPTIONS

Input and output file selection:

The default is to use standard input and standard output.

--input-file=FILENAME

Selects input file to used as input instead pipe input. Can not be used with --input-zeros --input-ones, --input-random or --input-counter. --input-integers and --input-tuples take their input from input file. Filename can also be "-" to describe pipe input (default).

--output-file=FILENAME

Selects output file to used as output instead pipe output. Filename can also be "-" to describe pipe output (default). Outuput file must not exist, bdd does not overwrite an existing file.

Input unit selection

A unit is a constant bit amount that repeats in the input:

--input-unit=BITS

Breaks input to units of BITS. If input is less than even amount of units, the most significant bits of the last unit are taken from the input file and the least significant bits are filled with zeroes. Input unit defaults to 8. If --input-pattern is defined, the combined bit size of the --input-pattern defines the input unit. You must not define both --input-unit and --input-pattern.

--input-skip-bits=BITS

Skip BITS bits from the beginning of the input.

--input-skip-units=UNITS

Skip UNITS units from the beginning of the input. If both skip-bits and skip-units are defined, they are done both. --skip has the same effect but sometimes it is more natural to use --input-skip-units to find units and use --skip for units. --input-skip-units is potentially more efficient because it is able to use seek to find the right place in the input file (look --input-use-seek).

--input-gap=BITS

After reading a unit, skip BITS bits.

--input-pregap=BITS

Before reading a unit, skip BITS bits. Can always be implemented by adding BITS to both --input-skip-bits and --input-gap, but using pregap is much clearer in some cases.

--input-assert-aligned

Gives error message if the input is not full units. Note that with --count you can cleanly stop reading non-aligned input.

--input-use-seek

Use seek in input to do --input-skip-bits, --input-skip-units and first --input-pregap. Default is to read them and discard the result. Does not work if input is a pipe.

--input-reverse-bytes

Every time when a byte is read from input, bits are reversed. 0x01 becomes 0x80. Look --input-little-endian for more information.

--input-reverse-unit

Every time when a unit is read, bits are reversed. Look --input-little-endian for more information.

--input-little-endian

By default, the input is assumed to be in network byte order or in big-endian byte order. However, the natural byte order for most processors is little-endian byte order. 16-bit big-endian integer 1 in byte stream is 0x00 0x01 and low-endian 0x01 0x00. In some systems, even if integers are low-endian, floating point numbers and strings are in big-endian order.

Unit is translated as an integer assuming that the least significant bit is the first one. To implement this as generic as possible, this is actually a synonym for combination --input-reverse-bytes and --input-reverse-unit. If we read 16-bit integer 1, in low-endian input stream it is 0x01 0x00. With --input-reverse-bytes it turns to 0x80 0x00 and with --input-reverse-unit it turns to 0x00 0x01. By these two steps, the operation has a meaningful interpretation even in non-byte-aligned cases.

After --input-little-endian, the content is usually read as a big-endian number in --input-pattern!

Bypass bit stream input (fake inputs)

Instead of the file input, the input can be one of these fake inputs. You can use either file input or one of these fake inputs. You must use --count to tell how many units the fake input has (except for --input-integers).

Values of fake inputs are truncated to the input unit size. For example, if you use --input-counter with input unit size 8 and output unit size 9, the topmost bit of the output is always zero.

If a fake input is used, the standard input is not opened at all for input, and standard input can be used with

--merge-file.

None of the fake inputs does support reversing bits, initial skips or gap operations. --input-random, --input-counter and --input-integers support --skip (also others allow it, but it does nothing as it would not affect the output).

--input-zeros
: All the input units are zeros.

--input-ones
: Like --input-zeros but all the bits of the input unit are ones.

--input-random
: Like --input-zeros but all the input bits are random (from /dev/urandom).

--input-counter
: Like --input-zeros, but input is units, every one with one bigger value than the previous. Counter value is cut to input unit size. Can not be reversed but --skip can be used to start from a specific number. If --skip=N, the first counter value will be N.

--input-integers
: Instead of a bit stream, input is text, 10-base unsigned integers that are separated with newline characters. Whitespace around number is allowed and empty lines are skipped. Can not be used with any other unit selection argument but can be used with --skip and --count to read only a part of the file.

Unpack a unit to a tuple

The input unit is split into fields according the --input-pattern. Fields are possibly interpreted as typed values (signed integers, floats or strings).

If there is no --output-pattern, the first field of the tuple is output as the sole unit. It must be a positive integer to be interpreted as a unit. If there is no --input-pattern, --output-pattern can still be used but there is only one field available and it is the input unit as whole.

--input-pattern=PAT
: Interpret every unit as fields as described in the PAT. PAT consists of patterns that match some bits in the input unit. A pattern has first the number of the bits matched and then a character that gives those bits the interpretation. For example,

```
--input-pattern=1U7U
```

divides a 8-bit unit into two integers, the first bit forms one field and seven next bits form another field. U means that the fields are interpreted as unsigned integers.

```
x - -  - skip these bits without output
U - i  - field is unsigned integer (msb first)
u - i  - field is unsigned integer (lsb first)
M - ii - reads bits as 2's complement signed integer,
         but stores sign and magnitude separately as
         two positive integer fields
m - ii - same as M, lsb first
```

M and m generate two fields from one number. The sign is output first as one integer and the magnitude second as another integer. The sign is zero for positive numbers and one for negative numbers. Note that magnitude alone needs as many bits as the original number (look --cut-maxint).

The following type characters are only for rare, advanced use. The main use is that they can give a human interpretation of the input stream using --output-tuples.

If the need is only rearrange bits, you should use only U pattern. The following type characters give some functionality that is sometimes necessary and it makes sense to output them to the bit stream:

```
S - i - field is 2's complement signed integer (msb)
s - i  - field is 2's complement signed integer (lsb)
F - d  - IEEE-754 32-bit floating point number
f - d  - same, lsb first
D - d  - IEEE-754 64-bit floating point number
d - d  - same, lsb first
c - s  - field is unicode UTF-8 character string
C - s  - same, lsb first
```

S and s should be used with caution. If negative value is used in making a unit, it will cause an error. The only way to use negative values correctly is --output-pattern=nS, --abs or --output-tuples.

Float can be written to output using output-patterns F, f, D and d which means that it can be used to make conversion from 32-bit float to 64-bit float and vice versa. Other uses of floats in the output stream is not possible. If the idea is only to re-arrange bits, you can use type character U more effciently to handle the unit as whole without internal interpretation for the field.

Most likely, you want to use upper case variants of the patterns (bitwise big-endian representation). If you want to interpret bytewise low-endian numbers, you likely want to use --input-low-endian and still use uppercase patterns. Lowercase patterns are for the (rare) case that the value in bit stream is exactly in the bitwise (not bytewise) opposite order, least significant bit first.

Rule to remember: uppercase letters are big-endian, lowercase letters are low-endian.

Floating point number length must match the bit length. D must be always written as pattern 64D.

Also, the string must be right amount of bytes. Some byte strings can not be output as a valid UTF-8 string.

Bypass unit and field selection

--input-tuples

Instead of a bit stream, input is a text file containing tuples. Tuple is a line of comma-separated values like:

```
123, -55, 678.5, "abba"
```

Tuples are separated with newline. Whitespace around the comma is allowed. If --input-tuples is used, unit or field selection arguments have no effect but --count, --skip and tuple manipulation options can be used.

Allowed types and their internal representations are:

Integers (signed bignums)
Floats (translated as IEEE 754 64-bit floating point numbers)
Strings (UTF-8)

Manipulate tuple

Fields are represented with field numbers (starting from 0). Currently, the same option can not be used several times.

--rearrange=FIELDLIST

Selects fields from input tuple for output tuple. Comma-separated list of field numbers starting from 0 (field 0 is the first field in the tuple). Negative field number refers to the field counting from the end of the tuple (-1 is the last field). The same field number can exist several times duplicating the input field. Not all input fields need to be in the output tuple.

--cut-maxint=FIELD,MAXINT

Cuts the maximum positive or negative integer value in the field to maxint. The values bigger than maxint get the value maxint and values smaller than -maxint get the value -maxint. Field number can be negative when the position is counted from the end of the tuple.

One of the reasons to use this command is that the maximum negative number of 8-bit 2's complement number is -128. If we cut the magnitude of the number with pattern type 'm', we have to have 8 bits to represent magnitude in the output bit stream. If we want to output the result back to a 8-bit stream with separate sign and magnitude, we have to strip values -128 to -127 using --cut-maxint=1,127.

--remove-right=FIELD,BITS

If we want to output other bits from the field than the rightmost ones, we can remove BITS rightmost bits from the field.

--xor=FIELD,BITS

Turns the righmost n BITS from 0 to 1 and vice versa. For example, sign of a number read using pattern "M" can be changed using --xor=0,1.

--abs=FIELD,0

Removes sign from the value of the field. Typically used with field that is read using pattern S and the result can be output to pattern U.

--sign=FIELD,0

Removes magnitude of the field. The result is 1 if the value was negative and 0 if the value was positive.

Skip a part of tuples

--skip=N

N first units are skipped.

--count=N

Only N units are processed. For example, if --skip=1 and --count=1, only the second unit is processed.

Pack a tuple to a unit

--output-pattern='PAT'

Packs output tuple to output bit unit. Fields must be in order, use --rearrange to arrange fields to the order of output pattern. Types are the same as the ones with the input pattern with the following exceptions:

z - fill with zero
o - fill with one
r - fill with random (from /dev/urandom)
x - can not be used in the output pattern

Also in the output-pattern, lowercase letters turn the bit order reversed. Look --input-pattern for more information.

z, o and r do not consume fields from the tuple. m and M consume two fields from the tuple and convert them to one 2's complement bit unit.

If signed value is too big to a signed field, the magnitude is first truncated to the right size. 4-bit signed output of -17 is '1111', because 17 is first truncated to 4 bits giving '0001' or 1 and the bit representation of -1 is '1111'.

If tuples are used (--input-pattern or --input-tuples), but there is is no --output-pattern option, the first field in the tuple is used directly as the output unit. The first field of the tuple must be unsigned integer if it is used as an output unit.

Collect output data stream

--output-unit=BITS

Packs the output unit to BITS bits. If --output-pattern is defined, output unit size comes from the total size of the output pattern and --output-unit must not be defined. If there is neither --output-unit nor --output-pattern, the default output unit size is 8 bits.

The last output byte is filled with zeros, if needed.

--output-little-endian

Unit is written to the output stream least significant byte first. The default is the network byte order or big-endian byte order where the most significant byte is written first. Actually --output-little-endian is a combination of --output-reverse-unit and --output-reverse-bytes and as such, those options can not be used with --output-little-endian. Look --input-little-endian for more discussion.

--output-reverse-unit

Reverses bits in the output unit. Look --output-little-endian for more information.

--output-reverse-bytes

Reverses bits of every written byte. Look --output-little-endian for more information.

Merge output with another data stream:

Units of the merge file are written to the output after every normal output unit. Output is finished when the normal output is finished. Last merge unit is written after the last output unit so that as many merge units as normal units are written. It is an error if the merge file does not have enough units but it can contain more units than the normal input file and the extra units are ignored.

Merge unit is defined like the normal input unit. Merge unit is written to the output with the same unit size as it has in input. Merge unit size can be different than the normal output unit. Output of the merge unit is done with the same --reverse-bytes and --reverse-unit options as the normal output.

--merge-file=FILENAME
: Opens the merge file. If FILENAME is "-", mergefile is standard input. If mergefile is standard input, input file must not be the same.

--merge-unit=BITS
: Breaks merge input to units of BITS.

--merge-skip-bits=BITS
: Skip BITS bits from the beginning of the merge input

--merge-skip-units=UNITS
: Skip UNITS units from the beginning of the merge input.

--merge-copy-first=BITS
: Copy BITS bits as-is from the beginning of the merge file before writing the first unit. If both skip (bits and/or units) and --merge-copy-first are defined, skip is done first.

--merge-gap=BITS
: After writing a unit, BITS bits are skipped from the merge file.

--merge-pregap=BITS
: Before reading a unit, skip BITS bits. Can always be implemented by adding BITS to --merge-skip-bits and --merge-gap, but using pregap is much clearer in some cases.

--merge-assert-aligned
: Warn if the whole merge input was not used or the merge input did not end to the byte border.

--merge-use-seek
: Seek is used in skipping in merge input instead of reading skipped bytes.

--merge-little-endian
: Merge file is read with both --merge-reverse-bytes and --merge-reverse-unit on.

--merge-reverse-bytes
: Every byte read from merge file is reversed.

--merge-reverse-unit
: Every unit read from merge file is reversed.

Show output as human readable

--output-tuples
: Replace bit output with human-readable output of tuples. This output is readable with --input-tuples

--output-integers

Replace bit output with human-readable output of units as integers This output is readable with --input-integers

If merge file is used, the main unit and the merge unit is printed separately as two integers (possibly with different unit size).

The integer is truncated to the unit size but reversing bits does not affect to the integer output.

--output-hex

Replace bit output with human-readable output of units as hex numbers.

Units are written as they would be written to the bit stream except that --output-little-endian, --output-reverse-bytes and --output-reverse-unit do not have effect to the hex output.

Output unit size affects the number as it affects in the bit output. If output unit size is 3 bits, the number 8 is printed as hex 0. The hex number has as many digits as needed to represent the full unit but topmost zero bits are not necessarily a part of the output unit. Units from the merge file are written with their own output unit so that they may have different amount of digits.

Output is divided to lines so that in one line there is 1, 2, 4, 8, 16 or 32 units separated with spaces. The line length is always between 40 and 80 characters.

--output-bits

Replace bit output with human-readable output of units as bits. Look --output-hex for more information.

EXAMPLES

bdd < foo > bar

Just copies file foo to bar, one byte at time.

bdd --input-unit=3

Extracts 3-bit stream values from standard input and expands values
to bytes in standard output. If the file is not aligned to 3 bits,
the missing least significant bits for the last unit are zeros.

bdd --input-unit=1 --output-pattern=1U1z

Output stream is input stream with zero added after every bit.
Four two-bit units are packed to one byte.

bdd --input-pattern=8U8U --rearrange=1,0 --output-pattern=8U8U

Swap two consecutive bytes.

echo -en '1,2,30,7,7' | bdd --input-tuples --output-pattern='2U3U3U' | od -t o1

Build octal numbers from fields (prints 123 377)

bdd --input-counter --count=16 --output-hex

Prints:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

bdd --input-counter --count=16 --input-unit=12 --output-unit=12 --merge-file=/etc/passwd --output-hex

Prints first 16 bytes of /etc/passwd file as hex preceeded by a three-digit hex address of the byte:

```
000 72 001 6f 002 6f 003 74 004 3a 005 78 006 3a 007 30
008 3a 009 30 00a 3a 00b 72 00c 6f 00d 6f 00e 74 00f 3a
```

SEE ALSO

dd(1)

BUGS

bdd is pretty slow.

Implementation does not allow duplicate manipulation options (--xor etc.).

Syntax for --abs and --sign is painful.

--input-tuples translates even quoted numbers to numbers, not strings.

In many cases, warning of bad input repeats for each unit.

In the next version, it is very likely that --output-unit will default to --input-unit, not to 8 bits.

AUTHOR

Esa Turtiainen (esa<at>turtiainen<dot>eu)