Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'od.1p' command

$ man od.1p

OD(1P) POSIX Programmer's Manual OD(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

od ? dump files in various formats

SYNOPSIS

od [-v] [-A address_base] [-j skip] [-N count] [-t type_string]... [file...]

od [-bcdosx] [file] [[+]offset[.][b]]

DESCRIPTION

The od utility shall write the contents of its input files to standard output in a user-specified format.

OPTIONS

The od utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines, except that the order of presentation of the -t options and the -bcdosx options is sig? nificant.

The following options shall be supported:

-A address_base

Specify the input offset base. See the EXTENDED DESCRIPTION

section. The application shall ensure that the address_base option-argument is a character. The characters 'd', 'o', and 'x' specify that the offset base shall be written in decimal, octal, or hexadecimal, respectively. The character 'n' speci? fies that the offset shall not be written.

-b Interpret bytes in octal. This shall be equivalent to -t o1.

-c Interpret bytes as characters specified by the current set? ting of the LC_CTYPE category. Certain non-graphic characters appear as C escapes: "NUL=\0", "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal numbers.

-d Interpret words (two-byte units) in unsigned decimal. This shall be equivalent to -t u2.

-j skip Jump over skip bytes from the beginning of the input. The od utility shall read or seek past the first skip bytes in the concatenated input files. If the combined input is not at least skip bytes long, the od utility shall write a diagnos? tic message to standard error and exit with a non-zero exit status.

By default, the skip option-argument shall be interpreted as a decimal number. With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number; otherwise, with a leading '0', the offset shall be interpreted as an octal num? ber. Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted as a multiple of 512, 1024, or 1048576 bytes, respectively. If the skip number is hexadeci? mal, any appended 'b' shall be considered to be the final hexadecimal digit.

-N count Format no more than count bytes of input. By default, count shall be interpreted as a decimal number. With a leading 0x or 0X, count shall be interpreted as a hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an octal number. If count bytes of input (after successfully skipping, if -j skip is specified) are not available, it

shall not be considered an error; the od utility shall format the input that is available.

-o Interpret words (two-byte units) in octal. This shall be equivalent to -t o2.

-s Interpret words (two-byte units) in signed decimal. This shall be equivalent to -t d2.

-t type_string

Specify one or more output types. See the EXTENDED DESCRIP? TION section. The application shall ensure that the type_string option-argument is a string specifying the types to be used when writing the input data. The string shall con? sist of the type specification characters a, c, d, f, o, u, and x, specifying named character, character, signed decimal, floating point, octal, unsigned decimal, and hexadecimal, re? spectively. The type specification characters d, f, o, u, and x can be followed by an optional unsigned decimal integer that specifies the number of bytes to be transformed by each instance of the output type. The type specification character f can be followed by an optional F, D, or L indicating that the conversion should be applied to an item of type float, double, or long double, respectively. The type specification characters d, o, u, and x can be followed by an optional C, S, I, or L indicating that the conversion should be applied to an item of type char, short, int, or long, respectively. Multiple types can be concatenated within the same type_string and multiple -t options can be specified. Output lines shall be written for each type specified in the order in which the type specification characters are specified.

-v Write all input data. Without the -v option, any number of groups of output lines, which would be identical to the imme? diately preceding group of output lines (except for the byte offsets), shall be replaced with a line containing only an <asterisk> ('*').

-x Interpret words (two-byte units) in hexadecimal. This shall be equivalent to -t x2.

Multiple types can be specified by using multiple -bcdostx options. Output lines are written for each type specified in the order in which the types are specified.

OPERANDS

The following operands shall be supported:

file A pathname of a file to be read. If no file operands are specified, the standard input shall be used.

If there are no more than two operands, none of the -A, -j, -N, -t, or -v options is specified, and either of the follow? ing is true: the first character of the last operand is a <plus-sign> ('+'), or there are two operands and the first character of the last operand is numeric; the last operand shall be interpreted as an offset operand on XSI-conformant systems. Under these conditions, the results are unspecified on systems that are not XSI-conformant systems.

[+]offset[.][b]

The offset operand specifies the offset in the file where dumping is to commence. This operand is normally interpreted as octal bytes. If '.' is appended, the offset shall be in? terpreted in decimal. If 'b' is appended, the offset shall be interpreted in units of 512 bytes.

STDIN

The standard input shall be used if no file operands are specified, and shall be used if a file operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of od:

LANG Provide a default value for the internationalization vari?

ables that are unset or null. (See the Base Definitions vol?
ume of POSIX.1?2017, Section 8.2, Internationalization Vari?
ables for the precedence of internationalization variables
used to determine the values of locale categories.)

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

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er?
ror.

LC_NUMERIC
Determine the locale for selecting the radix character used when writing floating-point formatted output.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The od utility shall copy sequentially each input file to standard out?
put, transforming the input data according to the output types speci?
fied by the -t option or the -bcdosx options. If no output type is specified, the default output shall be as if -t oS had been specified.

The number of bytes transformed by the output type specifier c may be

variable depending on the LC_CTYPE category.

The default number of bytes transformed by output type specifiers d, f, o, u, and x corresponds to the various C-language types as follows. If the c99 compiler is present on the system, these specifiers shall cor? respond to the sizes used by default in that compiler. Otherwise, these sizes may vary among systems that conform to POSIX.1?2008.

* For the type specifier characters d, o, u, and x, the default num? ber of bytes shall correspond to the size of the underlying imple? mentation's basic integer type. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-lan? guage types char, short, int, and long. These numbers can also be specified by an application as the characters 'C', 'S', 'I', and 'L', respectively. The implementation shall also support the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The implementation shall support the decimal value corre? sponding to the C-language type long long. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corre? sponding type is stored in memory on the system.
* For the type specifier character f, the default number of bytes shall correspond to the number of bytes in the underlying implemen? tation's basic double precision floating-point data type. The im? plementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-lan? guage types float, double, and long double. These numbers can also be specified by an application as the characters 'F', 'D', and 'L', respectively.

The type specifier character a specifies that bytes shall be inter? preted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification.

Bytes with the values listed in the following table shall be written

using the corresponding names for those characters.

Table: Named Characters in od

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

Note: The "\012" value may be written either as lf or nl.

The type specifier character c specifies that bytes shall be interpreted as characters specified by the current setting of the LC_CTYPE locale category. Characters listed in the table in the Base Definitions volume of POSIX.1?2017, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as the corresponding escape sequences, except that <backslash> shall be written as a single <backslash> and a NUL shall be written as '\0'. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence "**" shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the -j skip or -N count option is specified along with the c type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined

as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more <blank> characters.

If, as a result of the specification of the -N option or end-of-file being reached on the last input file, input data only partially satis? fies an output type, the input shall be extended sufficiently with null bytes to write the last byte of the input.

Unless -A n is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the first character of the output line, and shall be followed by one or more <blank> characters. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any <blank> characters.

If no -A option is specified, the input offset base is unspecified.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

XSI-conformant applications are warned not to use filenames starting with '+' or a first operand starting with a numeric character so that

the old functionality can be maintained by implementations, unless they specify one of the -A, -j, or -N options. To guarantee that one of these filenames is always interpreted as a filename, an application could always specify the address base format with the -A option.

EXAMPLES

If a file containing 128 bytes with decimal values zero to 127, in in? creasing order, is supplied as standard input to the command:

```
od -A d -t a
```

on an implementation using an input block size of 16 bytes, the stan? dard output, independent of the current locale setting, would be simi? lar to:

```
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000016 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000032  sp   !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
0000048   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000064   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000080   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000096   `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000112   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del
0000128
```

Note that this volume of POSIX.1?2017 allows nl or lf to be used as the name for the ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character lf (line feed), but traditional imple? mentations have referred to this character as newline (nl) and the POSIX locale character set symbolic name for the corresponding charac? ter is a <newline>.

The command:

```
od -A o -t o2x2x -N 18
```

on a system with 32-bit words and an implementation using an input block size of 16 bytes could write 18 bytes in approximately the fol? lowing format:

```
0000000 032056 031440 041123 042040 052516 044530 020043 031464
          342e   3320   4253   4420   554e   4958   2023   3334
```

342e3320 42534420 554e4958 20233334

0000020 032472

353a

353a0000

0000022

The command:

od -A d -t f -t o4 -t x4 -N 24 -j 0x15

on a system with 64-bit doubles (for example, IEEE Std 754?1985 double precision floating-point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:

0000000 1.00000000000000e+00 1.57350000000000e+01

07774000000 00000000000 10013674121 35341217270

3ff00000 00000000 402f3851 eb851eb8

0000016 1.40668230000000e+02

10030312542 04370303230

40619562 23e18698

0000024

RATIONALE

The od utility went through several names in early proposals, including hd, xd, and most recently hexdump. There were several objections to all of these based on the following reasons:

* The hd and xd names conflicted with historical utilities that be? haved differently.
* The hexdump description was much more complex than needed for a simple dump utility.
* The od utility has been available on all historical implementations and there was no need to create a new name for a utility so similar to the historical od utility.

The original reasons for not standardizing historical od were also fairly widespread. Those reasons are given below along with rationale explaining why the standard developers believe that this version does not suffer from the indicated problem:

* The BSD and System V versions of od have diverged, and the inter?

section of features provided by both does not meet the needs of the user community. In fact, the System V version only provides a mech? anism for dumping octal bytes and shorts, signed and unsigned deci? mal shorts, hexadecimal shorts, and ASCII characters. BSD added the ability to dump floats, doubles, named ASCII characters, and octal, signed decimal, unsigned decimal, and hexadecimal longs. The ver? sion presented here provides more normalized forms for dumping bytes, shorts, ints, and longs in octal, signed decimal, unsigned decimal, and hexadecimal; float, double, and long double; and named ASCII as well as current locale characters.

* It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of the stan? dard developers. The historical default od output is the specified default output of this utility. None of the option letters chosen for this version of od conflict with any of the options to histori? cal versions of od.

* On systems with different sizes for short, int, and long, there was no way to ask for dumps of ints, even in the BSD version. Because of the way options are named, the name space could not be extended to solve these problems. This is why the -t option was added (with type specifiers more closely matched to the printf() formats used in the rest of this volume of POSIX.1?2017) and the optional field sizes were added to the d, f, o, u, and x type specifiers. It is also one of the reasons why the historical practice was not man? dated as a required obsolescent form of od. (Although the old ver? sions of od are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The a, c, f, o, and x types match the meaning of the corre? sponding format characters in the historical implementations of od except for the default sizes of the fields converted. The d format is signed in this volume of POSIX.1?2017 to match the printf() no? tation. (Historical versions of od used d as a synonym for u in this version. The System V implementation uses s for signed deci?

mal; BSD uses i for signed decimal and s for null-terminated strings.) Other than d and u, all of the type specifiers match for? mat characters in the historical BSD version of od.

The sizes of the C-language types char, short, int, long, float, double, and long double are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes shorts, 2 bytes ints, and 4 bytes longs, while another compiler (or an option to the same compiler) uses 2 bytes shorts, 4 bytes ints, and 4 bytes longs.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values re? ported by invocations of the getconf utility when called with sys? tem_var operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types char, short, int, and long, respectively.

There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of POSIX.1?2017 or this volume of POSIX.1?2017. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types float, double, and long double, respectively. If the optional c99 utility is provided by the imple? mentation and used as specified by this volume of POSIX.1?2017, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guaran? tee that the od utility is able to interpret binary data output by such a program correctly.

This volume of POSIX.1?2017 requires that the numeric values of these lengths be recognized by the od utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of unsigned long data elements using od -t uL.

* The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The -A option now specifies the address base and the -S option specifies a starting offset.

* It would be difficult to break the dependence on US ASCII to achieve an internationalized utility. It does not seem to be any harder for od to dump characters in the current locale than it is for the ed or sed l commands. The c type specifier does this with? out difficulty and is completely compatible with the historical im? plementations of the c format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The a type specifier (from the BSD a format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by pax could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of their base codeset.

The use of "**" as an indication of continuation of a multi-byte char? acter in c specifier output was chosen based on seeing an implementa? tion that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

An early proposal used -S and -n, respectively, for the -j and -N op? tions eventually selected. These were changed to avoid conflicts with historical implementations.

The original standard specified -t o2 as the default when no output type was given. This was changed to -t oS (the length of a short) to accommodate a supercomputer implementation that historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at the same time to address an histor? ical implementation that had no two-byte data types in its C compiler.

The use of a basic integer data type is intended to allow the implemen? tation to choose a word size commonly used by applications on that ar? chitecture.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this ver? sion.

FUTURE DIRECTIONS

All option and operand interfaces marked XSI may be removed in a future version.

SEE ALSO

c99, sed

The Base Definitions volume of POSIX.1?2017, Chapter 5, File Format No? tation, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

COPYRIGHT