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 'file.1p' command

$ man file.1p

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

file ? determine file type

SYNOPSIS

file [-dh] [-M file] [-m file] file...

file -i [-h] file...

DESCRIPTION

The file utility shall perform a series of tests in sequence on each specified file in an attempt to classify it:

1. If file does not exist, cannot be read, or its file status could not be determined, the output shall indicate that the file was pro? cessed, but that its type could not be determined.
2. If the file is not a regular file, its file type shall be identi? fied. The file types directory, FIFO, socket, block special, and character special shall be identified as such. Other implementa? tion-defined file types may also be identified. If file is a sym? bolic link, by default the link shall be resolved and file shall test the type of file referenced by the symbolic link. (See the -h

and -i options below.)

3. If the length of file is zero, it shall be identified as an empty file.

4. The file utility shall examine an initial segment of file and shall make a guess at identifying its contents based on position-sensi? tive tests. (The answer is not guaranteed to be correct; see the -d, -M, and -m options below.)

5. The file utility shall examine file and make a guess at identifying its contents based on context-sensitive default system tests. (The answer is not guaranteed to be correct.)

6. The file shall be identified as a data file.

If file does not exist, cannot be read, or its file status could not be determined, the output shall indicate that the file was processed, but that its type could not be determined.

If file is a symbolic link, by default the link shall be resolved and file shall test the type of file referenced by the symbolic link.

OPTIONS

The file utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines, except that the order of the -m, -d, and -M options shall be significant.

The following options shall be supported by the implementation:

-d Apply any position-sensitive default system tests and con? text-sensitive default system tests to the file. This is the default if no -M or -m option is specified.

-h When a symbolic link is encountered, identify the file as a symbolic link. If -h is not specified and file is a symbolic link that refers to a nonexistent file, file shall identify the file as a symbolic link, as if -h had been specified.

-i If a file is a regular file, do not attempt to classify the type of the file further, but identify the file as specified in the STDOUT section.

-M file Specify the name of a file containing position-sensitive tests that shall be applied to a file in order to classify it

(see the EXTENDED DESCRIPTION). No position-sensitive default system tests nor context-sensitive default system tests shall be applied unless the -d option is also specified.

-m file Specify the name of a file containing position-sensitive tests that shall be applied to a file in order to classify it (see the EXTENDED DESCRIPTION).

If the -m option is specified without specifying the -d option or the -M option, position-sensitive default system tests shall be applied af? ter the position-sensitive tests specified by the -m option. If the -M option is specified with the -d option, the -m option, or both, or the -m option is specified with the -d option, the concatenation of the po? sition-sensitive tests specified by these options shall be applied in the order specified by the appearance of these options. If a -M or -m file option-argument is -, the results are unspecified.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be tested.

STDIN

The standard input shall be used if a file operand is '-' and the im? plementation treats the '-' as meaning standard input. Otherwise, the standard input shall not be used.

INPUT FILES

The file can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of file:

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 error and informative messages written to standard output.

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

ASYNCHRONOUS EVENTS

Default.

STDOUT

In the POSIX locale, the following format shall be used to identify each operand, file specified:

"%s: %s\n", <file>, <type>

The values for <type> are unspecified, except that in the POSIX locale, if file is identified as one of the types listed in the following ta? ble, <type> shall contain (but is not limited to) the corresponding string, unless the file is identified by a position-sensitive test specified by a -M or -m option. Each <space> shown in the strings shall be exactly one <space>.

Table 4-9: File Utility Output Strings

If file is:	<type> shall contain the string:	Notes?
Nonexistent	cannot open	? ?
?	?	? ?
?Block special	? block special	? 1 ?
?Character special	? character special	? 1 ?
?Directory	? directory	? 1 ?
?FIFO	? fifo	? 1 ?
?Socket	? socket	? 1 ?
?Symbolic link	? symbolic link to	? 1 ?

?Regular file ? regular file ? 1,2 ?

?Empty regular file ? empty ? 3 ?

?Regular file that cannot be read ? cannot open ? 3 ?

? ? ? ?

?Executable binary ? executable ? 3,4,6 ?

?ar archive library (see ar) ? archive ? 3,4,6 ?

?Extended cpio format (see pax) ? cpio archive ? 3,4,6 ?

?Extended tar format (see ustar in pax) ? tar archive ? 3,4,6 ?

? ? ? ?

?Shell script ? commands text ? 3,5,6 ?

?C-language source ? c program text ? 3,5,6 ?

?FORTRAN source ? fortran program text ? 3,5,6 ?

? ? ? ?

?Regular file whose type cannot be determined ? data ? 3 ?

??

Notes:

1. This is a file type test.
2. This test is applied only if the -i option is specified.
3. This test is applied only if the -i option is not speci?
   fied.
4. This is a position-sensitive default system test.
5. This is a context-sensitive default system test.
6. Position-sensitive default system tests and context-sen?
   sitive default system tests are not applied if the -M op?
   tion is specified unless the -d option is also specified.

In the POSIX locale, if file is identified as a symbolic link (see the -h option), the following alternative output format shall be used:

"%s: %s %s\n", <file>, <type>, <contents of link>"

If the file named by the file operand does not exist, cannot be read, or the type of the file named by the file operand cannot be determined, this shall not be considered an error that affects the exit status.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

A file specified as an option-argument to the -m or -M options shall contain one position-sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single '>' char? acter shall be applied.

Each line shall be composed of the following four <tab>-separated fields. (Implementations may allow any combination of one or more white-space characters other than <newline> to act as field separa? tors.)

offset An unsigned number (optionally preceded by a single '>' char? acter) specifying the offset, in bytes, of the value in the file that is to be compared against the value field of the line. If the file is shorter than the specified offset, the test shall fail.

If the offset begins with the character '>', the test con? tained in the line shall not be applied to the file unless the test on the last line for which the offset did not begin with a '>' was successful. By default, the offset shall be interpreted as an unsigned decimal number. With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal num? ber; otherwise, with a leading 0, the offset shall be inter? preted as an octal number.

type The type of the value in the file to be tested. The type shall consist of the type specification characters d, s, and u, specifying signed decimal, string, and unsigned decimal, respectively.

The type string shall be interpreted as the bytes from the file starting at the specified offset and including the same number of bytes specified by the value field. If insufficient

bytes remain in the file past the offset to match the value field, the test shall fail.

The type specification characters d and u can be followed by an optional unsigned decimal integer that specifies the num? ber of bytes represented by the type. The type specification characters d and u can be followed by an optional C, S, I, or L, indicating that the value is of type char, short, int, or long, respectively.

The default number of bytes represented by the type speci? fiers d, f, and u shall correspond to their respective C-lan? guage types as follows. If the system claims conformance to the C-Language Development Utilities option, those specifiers shall correspond to the default sizes used in the c99 util? ity. Otherwise, the default sizes shall be implementation-de? fined.

For the type specifier characters d and u, the default number of bytes shall correspond to the size of a basic integer type of the implementation. 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-language types char, short, int, or long. These num? bers can also be specified by an application as the charac? ters C, S, I, and L, respectively. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the cor? responding type is stored in memory on the system.

All type specifiers, except for s, can be followed by a mask specifier of the form &number. The mask value shall be AND'ed with the value of the input file before the comparison with the value field of the line is made. By default, the mask shall be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask shall be interpreted as an un? signed hexadecimal number; otherwise, with a leading 0, the

mask shall be interpreted as an unsigned octal number.

The strings byte, short, long, and string shall also be sup? ported as type fields, being interpreted as dC, dS, dL, and s, respectively.

value The value to be compared with the value from the file.

If the specifier from the type field is s or string, then in? terpret the value as a string. Otherwise, interpret it as a number. If the value is a string, then the test shall succeed only when a string value exactly matches the bytes from the file.

If the value is a string, it can contain the following se? quences:

\character The <backslash>-escape sequences as specified in the Base Definitions volume of POSIX.1?2017, Ta? ble 5-1, Escape Sequences and Associated Actions ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').

In addition, the escape sequence '\ ' (the <back? slash> character followed by a <space> character) shall be recognized to represent a <space> char? acter. The results of using any other character, other than an octal digit, following the <back? slash> are unspecified.

\octal Octal sequences that can be used to represent characters with specific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three oc? tal-digit characters (01234567).

By default, any value that is not a string shall be inter? preted as a signed decimal number. Any such value, with a leading 0x or 0X, shall be interpreted as an unsigned hexa? decimal number; otherwise, with a leading zero, the value shall be interpreted as an unsigned octal number.

If the value is not a string, it can be preceded by a charac?

ter indicating the comparison to be performed. Permissible characters and the comparisons they specify are as follows:

= The test shall succeed if the value from the file equals the value field.

< The test shall succeed if the value from the file is less than the value field.

> The test shall succeed if the value from the file is greater than the value field.

& The test shall succeed if all of the set bits in the value field are set in the value from the file.

^ The test shall succeed if at least one of the set bits in the value field is not set in the value from the file.

x The test shall succeed if the file is large enough to contain a value of the type specified starting at the offset specified.

message The message to be printed if the test succeeds. The message shall be interpreted using the notation for the printf for? matting specification; see printf. If the value field was a string, then the value from the file shall be the argument for the printf formatting specification; otherwise, the value from the file shall be the argument.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

The file utility can only be required to guess at many of the file types because only exhaustive testing can determine some types with certainty. For example, binary data on some implementations might match

the initial segment of an executable or a tar archive.

Note that the table indicates that the output contains the stated string. Systems may add text before or after the string. For executables, as an example, the machine architecture and various facts about how the file was link-edited may be included. Note also that on systems that recognize shell script files starting with "#!" as executable files, these may be identified as executable binary files rather than as shell scripts.

EXAMPLES

Determine whether an argument is a binary executable file:

```
file -- "$1" | grep -q ':.*executable' &&
    printf "%s is executable.\n$1"
```

RATIONALE

The -f option was omitted because the same effect can (and should) be obtained using the xargs utility.

Historical versions of the file utility attempt to identify the following types of files: symbolic link, directory, character special, block special, socket, tar archive, cpio archive, SCCS archive, archive library, empty, compress output, pack output, binary data, C source, FORTRAN source, assembler source, nroff/troff/eqn/tbl source troff output, shell script, C shell script, English text, ASCII text, various executables, APL workspace, compiled terminfo entries, and CURSES screen images. Only those types that are reasonably well specified in POSIX or are directly related to POSIX utilities are listed in the table.

Historical systems have used a ``magic file'' named /etc/magic to help identify file types. Because it is generally useful for users and scripts to be able to identify special file types, the -m flag and a portable format for user-created magic files has been specified. No requirement is made that an implementation of file use this method of identifying files, only that users be permitted to add their own classifying tests.

In addition, three options have been added to historical practice. The -d flag has been added to permit users to cause their tests to follow

any default system tests. The -i flag has been added to permit users to test portably for regular files in shell scripts. The -M flag has been added to permit users to ignore any default system tests.

The POSIX.1?2008 description of default system tests and the interac?tion between the -d, -M, and -m options did not clearly indicate that there were two types of ``default system tests''. The ``position-sensi?tive tests'' determine file types by looking for certain string or bi?nary values at specific offsets in the file being examined. These posi?tion-sensitive tests were implemented in historical systems using the magic file described above. Some of these tests are now built into the file utility itself on some implementations so the output can provide more detail than can be provided by magic files. For example, a magic file can easily identify a core file on most implementations, but can?not name the program file that dropped the core. A magic file could produce output such as:

    /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1

but by building the test into the file utility, you could get output such as:

    /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'

These extended built-in tests are still to be treated as position-sen?sitive default system tests even if they are not listed in /etc/magic or any other magic file.

The context-sensitive default system tests were always built into the file utility. These tests looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and other computer lan?guage source files, and even plain text files. With the addition of the -m and -M options the distinction between position-sensitive and con?text-sensitive default system tests became important because the order of testing is important. The context-sensitive system default tests should never be applied before any position-sensitive tests even if the -d option is specified before a -m option or -M option due to the high probability that the context-sensitive system default tests will incor?rectly identify arbitrary text files as text files before position-sen?

sitive tests specified by the -m or -M option would be applied to give a more accurate identification.

Leaving the meaning of -M - and -m - unspecified allows an existing prototype of these options to continue to work in a backwards-compati? ble manner. (In that implementation, -M - was roughly equivalent to -d in POSIX.1?2008.)

The historical -c option was omitted as not particularly useful to users or portable shell scripts. In addition, a reasonable implementa? tion of the file utility would report any errors found each time the magic file is read.

The historical format of the magic file was the same as that specified by the Rationale in the ISO POSIX?2:1993 standard for the offset, value, and message fields; however, it used less precise type fields than the format specified by the current normative text. The new type field values are a superset of the historical ones.

The following is an example magic file:

```
0  short     070707            cpio archive
0  short     0143561           Byte-swapped cpio archive
0  string    070707            ASCII cpio archive
0  long      0177555           Very old archive
0  short     0177545           Old archive
0  short     017437            Old packed data
0  string    \037\036          Packed data
0  string    \377\037          Compacted data
0  string    \037\235          Compressed data
>2 byte&0x80 >0                Block compressed
>2 byte&0x1f x                 %d bits
0  string    \032\001          Compiled Terminfo Entry
0  short     0433              Curses screen image
0  short     0434              Curses screen image
0  string    <ar>              System V Release 1 archive
0  string    !<arch>\n__.SYMDEF  Archive random library
0  string    !<arch>           Archive
```

```
0  string    ARF_BEGARF          PHIGS clear text archive
0  long      0x137A2950          Scalable OpenFont binary
0  long      0x137A2951          Encrypted scalable OpenFont binary
```

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

None.

SEE ALSO

ar, ls, pax, printf

The Base Definitions volume of POSIX.1?2017, Table 5-1, Escape Se? quences and Associated Actions, Chapter 8, Environment Variables, Sec? tion 12.2, Utility Syntax Guidelines

COPYRIGHT