IEEE P1003.2a Draft 8 – December 1991

5.14 `file` — Determine file type

5.14.1 Synopsis

`file` *file* . . .

5.14.2 Description

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

(1) If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, block special, and character special shall be identified as such. Other implementation-defined file types may also be identified.

(2) If the file is a regular file, and:

   (a) The file is zero-length, it shall be identified as an empty file.

   (b) The file is not zero-length, `file` shall examine an initial segment of the file and shall make a guess at identifying its contents or whether it is an executable binary file. (The answer is not guaranteed to be correct.)

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.

5.14.3 Options

None.

5.14.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to be tested.

5.14.5 External Influences

5.14.5.1 Standard Input

None.

5.14.5.2 Input Files

The *file* can be any file type.

5.14.5.3 Environment Variables

The following environment variables shall affect the execution of `file`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.14.5.4 Asynchronous Events

Default.

5.14.6 External Effects

5.14.6.1 Standard Output

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 Table 5-100, *<type>* shall contain (but is not limited to) the corresponding string. Each space shown in the strings shall be exactly one `<space>` character.

If the file named by the *file* operand does not exist, cannot be read, or the status of the file cannot be determined, the string `cannot open` shall be included as part of the *<type>* field, but this shall not be considered an error that affects the exit status.

Table 5-100 – `file` Output Strings

If *file* is a	<*type*> shall contain the string
directory	`directory`
FIFO	`fifo`
block special	`block special`
character special	`character special`
executable binary	`executable`
empty regular file	`empty`
`ar` archive library (see 6.1)	`archive`
extended `cpio` format (see POSIX.1 {8} 10.1.2)	`cpio archive`
extended `tar` format (see POSIX.1 {8} 10.1.1)	`tar archive`
shell script	`commands text`
C-language source	`c program text`
FORTRAN source	`fortran program text`

5.14.6.2 Standard Error

Used only for diagnostic messages.

5.14.6.3 Output Files

None.

5.14.7 Extended Description

None.

5.14.8 Exit Status

The `file` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

5.14.9 Consequences of Errors

Default.

5.14.10 Rationale. *(This subclause is not a part of P1003.2a)*

Usage, Examples

Historical systems have used a ‘‘magic file,’’ generally named `/etc/magic`, to help identify file types from their initial segments. Although the general format of the file is well known, it is not portable for more than a few file types; none of the interesting executable-guessing capabilities could be portable without a list of how all systems format their binary files. However, in the interest of some informal interoperability, the format is presented here for informative purposes only.

The file consists of four `<blank>`-separated fields:

offset A number specifying the offset, in bytes, into the file that is to be tested.

type The type of the data to be tested. The possible values for the type include `string`, `short`, `long`, and other data types from the C Standard {7} or other standards. Some types may be followed by a mask specifier of the form &*number* , which is ANDed with the value before any comparisons are done. The mask specifier is octal if preceded by a `0`, hexadecimal if preceded by a `0x`, and decimal otherwise.

value The value to be compared with the value from the file. If the value is a string, it may contain escape characters. If the value is numeric, it may be preceded by a character indicating the operation to be performed. Allowable values include:

- `=` The value from the file must equal the specified value.
- `<` The value from the file must be less than the specified value.
- `>` The value from the file must be greater than the specified value.
- `&` All bits in the specified value must be set in the value from the file.
- `^` At least one of the bits in the specified value must not be set in the value from the file.
- `x` Any value will match.

message The message to be printed if the comparison succeeds. If the string contains a *printf* () format specification, the value from the file is printed, using the message as the format string.

Some file formats contain additional information that is to be printed along with the file type. A line that begins with the character > indicates additional tests and messages to be printed. If the test on the line preceding the first line with a > succeeds, the test specified in all subsequent lines beginning with > are performed, and the messages printed if the tests succeed. The next line that does not begin with a > terminates this action.

Some example entries:

```
0     string   070707   ASCII cpio archive
0     string   !<arch>  portable archive
0     long     0550     executable
>12   long     >0       not stripped
```

History of Decisions Made

The `file` utility can only be required to ‘‘guess’’ at many of the file types because only exhaustive testing can determine some types with certitude. For example, binary data on some systems might match the initial segment of an executable or a `tar` archive. An exhaustive test might require that an executable be executed, or a C program be compiled, before certifying that the guess was correct. This is unreasonable. Since most historical systems generally provide very good guesses, `file` is in POSIX.2 with only the requirement that a best effort be made and that the output contain certain identifiable strings when the guess is made. Even on a very half-hearted system, the mandated output will be no less useful than that from `ls -l`.

The historical `-c` and `-m` options were omitted because the concept of a ‘‘magic file’’ is not portable.

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.2 utilities are listed in the table.

Implementations that support symbolic links are encouraged to use the string `symbolic link` to identify them.

5.15 `jobs` — Display status of jobs in the current session

5.15.1 Synopsis

`jobs` [`−l` | `−p`] [*job_id* ...]

5.15.2 Description

The `jobs` utility shall display the status of jobs that were started in the current shell environment; see 3.12.

When `jobs` reports the termination status of a job, the shell shall remove its pro- 7
cess ID from the list of those ‘‘known in the current shell execution environment’’;
see 3.9.3.1.

5.15.3 Options

The `jobs` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−l` (The letter ell.) Provide more information about each job listed. This information shall include the job number, current job, process group ID, state, and the command that formed the job.

`−p` Display only the process IDs for the process group leaders of the selected jobs.

By default, the `jobs` utility shall display the status of all stopped jobs, running 7
background jobs, and all jobs whose status has changed and have not been 7
reported by the shell. 7

5.15.4 Operands

The following operand shall be supported by the implementation:

job_id Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the status information for all jobs shall be displayed. The format of *job_id* is described in 2.2.2.206.

5.15.5 External Influences