IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

12405 ### 4.59.9 Consequences of Errors

12406 Default.

12407 ### 4.59.10 Rationale. *(This subclause is not a part of P1003.2)*

12408 **Examples, Usage**

12409 Since POSIX.1 {8} doesn't specify any output modes, they are not specified in this
12410 standard either. Implementations are expected to provide `stty` operands
12411 corresponding to all of the output modes they support.

12412 In many ways outside the scope of POSIX.2, `stty` is primarily used to tailor the
12413 user interface of the terminal, such as selecting the preferred ERASE and KILL
12414 characters. As an application programming utility, `stty` can be used within shell
12415 scripts to alter the terminal settings for the duration of the script. The `-g` flag is
12416 designed to facilitate the saving and restoring of terminal state from the shell
12417 level. For example, a program may:

```
12418     saveterm="$(stty -g)"   # save terminal state
12419     stty (new settings)     # set new state
12420     ...                     # ...
12421     stty $saveterm          # restore terminal state
```

12422 Since the format is unspecified, the saved value is not portable across systems.

12423 Since the `-a` format is so loosely specified, scripts that save and restore terminal
12424 settings should use the `-g` option.

12425 **History of Decisions Made**

12426 The original `stty` manual page was taken directly from System V and reflected
12427 the System V terminal driver *termio*. It has been modified to correspond to the
12428 POSIX.1 {8} terminal driver *termios*.

12429 The *termios* section states that individual disabling of control characters is an
12430 option {_POSIX_VDISABLE}. If enabled, two conventions currently exist for speci-
12431 fying this: System V uses `"^-"`, and BSD uses `undef`. Both are accepted by
12432 POSIX.2 `stty`. The other BSD convention of using the letter `u` was rejected
12433 because it conflicts with the actual letter `u`, which is an acceptable value for a con-
12434 trol character.

12435 Early drafts did not specify the mapping of ^*c* to control characters because the
12436 control characters were not specified in the POSIX Locale character set description
12437 file requirements. The control character set is now specified in 2.4.1, so the tradi-
12438 tional mapping is specified. Note that although the mapping corresponds to
12439 control-character key assignments on many terminals that use ISO/IEC 646 {1} (or
12440 ASCII) character encodings, the mapping specified here is to the control charac-
12441 ters, not their keyboard encodings.

12442 The combination options `raw` and `cooked` (`-raw`) were dropped from the standard
12443 because the exact values that should be set are not well understood or commonly

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

agreed on. In particular, *termios* has no explicit RAW bit, and the options that should be re-enabled (`–raw`) are not clear. General programming practice is to save the terminal state, change the settings for the duration of the program, and then reset the state. This is easy to do within a C program, however it is not possible for a single invocation of `stty` to restore the terminal state (`–raw`) without knowledge of the prior settings. Using the `–g` option and two calls to `stty`, a shell application could do this as described above. However, it is impossible to implement this as a single option. Also, it is not clear that changing word size and parity is appropriate. For example, requiring that `cooked` set `cs7` and `parenb` would be disastrous for users working with 8–bit international character sets. In general, these options are too ill-defined to be of any use.

Since *termios* supports separate speeds for input and output, two new options were added to specify each distinctly.

The `ixany` input mode was removed from Draft 8 on the basis that it could not be implemented on a POSIX.1 {8} system without extensions.

Some historical implementations use standard input to get and set terminal characteristics; others use standard output. Since input from a login TTY is usually restricted to the owner while output to a TTY is frequently open to the world, using standard input provides fewer chances of accidentally (or mischievously) altering the terminal settings of other users. Using standard input also allows `stty –a` and `stty –g` output to be redirected for later use. Therefore, usage of standard input is required by this standard.

The `tostop` option was omitted from early drafts through an oversight. It is the only option that requires job control to be effective, and thus could have gone into the UPE as a modification to `stty`, but since all other terminal control features are in the base standard, `tostop` was included as well.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12470 # 4.60 `tail` — Copy the last part of a file

12471 ## 4.60.1 Synopsis

12472 `tail [-f][ -c` *number* `| -n` *number* `][`*file*`]`

12473 *Obsolescent versions:*

12474 `tail -[`*number*`][c|l][f] [`*file*`]`

12475 `tail +[`*number*`][c|l][f] [`*file*`]`

12476 ## 4.60.2 Description

12477 The `tail` utility shall copy its input file to the standard output beginning at a
12478 designated place.

12479 Copying shall begin at the point in the file indicated by the `-c` *number* or
12480 `-n` *number* options (or the ±*number* portion of the argument to the obsolescent
12481 version). The option-argument *number* shall be counted in units of lines or bytes,
12482 according to the options `-n` and `-c` (or, in the obsolescent version, the appended
12483 option suffixes `l` or `c`).

12484 Tails relative to the end of the file may be saved in an internal buffer, and thus
12485 may be limited in length. Implementations shall ensure that such a buffer, if any,
12486 is no smaller than {LINE_MAX}*10 bytes.

12487 ## 4.60.3 Options

12488 The `tail` utility shall conform to the utility argument syntax guidelines
12489 described in standard described in 2.10.2, except that the obsolescent version
12490 accepts multicharacter options that can preceded by a plus sign.

12491 The following options shall be supported by the implementation in the nonob-
12492 solescent version:

12493 `-c` *number* The *number* option-argument shall be a decimal integer whose
12494 sign affects the location in the file, measured in bytes, to begin
12495 the copying:

12496

Sign	Copying Starts
+	Relative to the beginning of the file.
-	Relative to the end of the file.
none	Relative to the end of the file.

12497
12498
12499

12500 The origin for counting shall be 1; i.e., `-c +1` represents the first 1
12501 byte of the file, `-c -1` the last. 1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12502 **−f** If the input file is a regular file or if the *file* operand specifies a
12503 FIFO , do not terminate after the last line of the input file has
12504 been copied, but read and copy further bytes from the input file
12505 when they become available. If no *file* operand is specified and
12506 standard input is a pipe, the **−f** option shall be ignored. If the
12507 input file is not a FIFO , pipe, or regular file, it is unspecified
12508 whether or not the **−f** option shall be ignored.

12509 **−n** *number* This option shall be equivalent to **−c** *number* , except the starting
12510 location in the file shall be measured in lines instead of bytes. 1
12511 The origin for counting shall be 1; i.e., **-n +1** represents the first 1
12512 line of the file, **-n -1** the last. 1

12513 In the obsolescent version, an argument beginning with a **-** or **+** can be used as a
12514 single option. The argument ±*number* with the letter **c** specified as a suffix shall
12515 be equivalent to **−c** ±*number* ; ±*number* with the letter **l** specified as a suffix, or
12516 with neither **c** nor **l** as a suffix, shall be equivalent to **−n** ±*number* . If *number* is
12517 not specified in these forms, 10 shall be used. The letter **f** specified as a suffix
12518 shall be equivalent to specifying the **−f** option. If the **-[***number* **]c[f]** form is used
12519 and neither *number* nor the **f** suffix is specified, it shall be interpreted as the
12520 **−c** *number* option.

12521 In the nonobsolescent form, if neither **−c** nor **−n** is specified, **-n 10** shall be
12522 assumed.

12523 ### 4.60.4 Operands

12524 The following operand shall be supported by the implementation:

12525 *file* A pathname of an input file. If no *file* operands are specified, the
12526 standard input shall be used.

12527 ### 4.60.5 External Influences

12528 #### 4.60.5.1 Standard Input

12529 The standard input shall be used only if no *file* operands are specified. See Input
12530 Files.

12531 #### 4.60.5.2 Input Files

12532 If the **−c** option is specified, the input file can contain arbitrary data; otherwise,
12533 the input file shall be a text file.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12534 #### 4.60.5.3 Environment Variables

12535 The following environment variables shall affect the execution of `tail`:

12536 **LANG** This variable shall determine the locale to use for the
12537 locale categories when both **LC_ALL** and the correspond-
12538 ing environment variable (beginning with **LC_**) do not
12539 specify a locale. See 2.6.

12540 **LC_ALL** This variable shall determine the locale to be used to over-
12541 ride any values for locale categories specified by the set-
12542 tings of **LANG** or any environment variables beginning
12543 with **LC_**.

12544 **LC_CTYPE** This variable shall determine the locale for the interpreta-
12545 tion of sequences of bytes of text data as characters (e.g.,
12546 single- versus multibyte characters in arguments and
12547 input files).

12548 **LC_MESSAGES** This variable shall determine the language in which mes-
12549 sages should be written.

12550 #### 4.60.5.4 Asynchronous Events

12551 Default.

12552 ### 4.60.6 External Effects

12553 #### 4.60.6.1 Standard Output

12554 The designated portion of the input file shall be written to standard output.

12555 #### 4.60.6.2 Standard Error

12556 Used only for diagnostic messages.

12557 #### 4.60.6.3 Output Files

12558 None.

12559 ### 4.60.7 Extended Description

12560 None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12561 ### 4.60.8 Exit Status

12562 The `tail` utility shall exit with one of the following values:

12563 0 Successful completion.

12564 >0 An error occurred.

12565 ### 4.60.9 Consequences of Errors

12566 Default.

12567 ### 4.60.10 Rationale. *(This subclause is not a part of P1003.2)*

12568 **Usage, Examples**

12569 The nonobsolescent version of `tail` was created to allow conformance to the Util-
12570 ity Syntax Guidelines. The historical `-b` option was omitted because of the gen-
12571 eral nonportability of block-sized units of text. The `-c` option historically meant
12572 ''characters,'' but this standard indicates that it means ''bytes.'' This was selected
12573 to allow reasonable implementations when multibyte characters are possible; it
12574 was not named `-b` to avoid confusion with the historical `-b`.

12575 Note that the `-c` option should be used with caution when the input is a text file
12576 containing multibyte characters; it may produce output that does not start on a
12577 character boundary.

12578 The origin of counting both lines and bytes is 1, matching all widespread histori- 1
12579 cal implementations. 1

12580 The restriction on the internal buffer is a compromise between the historical
12581 System V implementation of 4K and the BSD 32K.

12582 The `-f` option can be used to monitor the growth of a file that is being written by
12583 some other process. For example, the command:

12584 ```
tail -f fred
```

12585 prints the last ten lines of the file `fred`, followed by any lines that are appended
12586 to `fred` between the time `tail` is initiated and killed. As another example, the
12587 command:

12588 ```
tail -f -c 15 fred
```

12589 prints the last 15 bytes of the file `fred`, followed by any bytes that are appended
12590 to `fred` between the time `tail` is initiated and killed.

12591 Although the input file to `tail` can be any type, the results need not be what
12592 would be expected on some character special device files or on file types not
12593 described by POSIX.1 {8}. Since the standard does not specify the block size used
12594 when doing input, `tail` need not read all of the data from devices that only per-
12595 form block transfers.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12596 **History of Decisions Made**

12597 The developers of the standard originally decided that `tail`, and its frequent
12598 companion, `head`, were useful mostly to interactive users, and not application
12599 programs. However, balloting input suggested that these utilities actually do find
12600 significant use in scripts, such as to write out portions of log files. The balloters
12601 also challenged the working group's assumption that clever use of `sed` could be an
12602 appropriate substitute for `tail`.

12603 The `-f` option has been implemented as a loop that sleeps for one second and
12604 copies any bytes that are available. This is sufficient, but if more efficient
12605 methods of determining when new data are available are developed, implementa-
12606 tions are encouraged to use them.

12607 Historical documentation says that `tail` ignores the `-f` option if the input file is a
12608 pipe (pipe and FIFO on systems that support FIFOs). On BSD-based systems, this
12609 has been true; on System V-based systems, this was true when input was taken
12610 from standard input, but behaved as on other files if a FIFO was named as the *file*
12611 operand. Since the `-f` option is not useful on pipes and all historical implementa-
12612 tions ignore `-f` if no *file* operand is specified and standard input is a pipe, POSIX.2
12613 requires this behavior. However, since the `-f` option is useful on a FIFO, POSIX.2
12614 also requires that if standard input is a FIFO or a FIFO is named, the `-f` option
12615 shall not be ignored. Although historical behavior does not ignore the `-f` option
12616 for other file types, this is unspecified so that implementations are allowed to
12617 ignore the `-f` option if it is known that the file cannot be extended.

12618 An earlier draft had the synopsis line:

12619 `tail [ -c | -l ][-f][-n` *number* `][`*file*`]`

12620 This was changed to the current form based on comments and objections noting
12621 that `-c` was almost never used without specifying a number and there was no
12622 need to specify `-l` if `-n` *number* was given.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12623 ## 4.61 `tee` — Duplicate standard input

12624 ### 4.61.1 Synopsis

12625 `tee [-ai][`*file* `...]`

12626 ### 4.61.2 Description

12627 The `tee` utility shall copy standard input to standard output, making a copy in
12628 zero or more files. The `tee` utility shall not buffer output.

12629 The options determine if the specified files are overwritten or appended to.

12630 ### 4.61.3 Options

12631 The `tee` utility shall conform to the utility argument syntax guidelines described
12632 in 2.10.2.

12633 The following options shall be supported by the implementation:

12634 `-a` Append the output to the files rather than overwriting them.

12635 `-i` Ignore the SIGINT signal.

12636 ### 4.61.4 Operands

12637 The following operands shall be supported by the implementation:

12638 *file* A pathname of an output file. Implementations shall support pro-
12639 cessing of at least 13 *file* operands.

12640 ### 4.61.5 External Influences

12641 #### 4.61.5.1 Standard Input

12642 The standard input can be of any type.

12643 #### 4.61.5.2 Input Files

12644 None.

12645 #### 4.61.5.3 Environment Variables

12646 The following environment variables shall affect the execution of `tee`:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12647 **LANG** This variable shall determine the locale to use for the
12648 locale categories when both **LC_ALL** and the correspond-
12649 ing environment variable (beginning with **LC_**) do not
12650 specify a locale. See 2.6.

12651 **LC_ALL** This variable shall determine the locale to be used to over-
12652 ride any values for locale categories specified by the set-
12653 tings of **LANG** or any environment variables beginning
12654 with **LC_**.

12655 **LC_CTYPE** This variable shall determine the locale for the interpreta-
12656 tion of sequences of bytes of text data as characters (e.g.,
12657 single- versus multibyte characters in arguments).

12658 **LC_MESSAGES** This variable shall determine the language in which mes-
12659 sages should be written.

12660 #### 4.61.5.4 Asynchronous Events

12661 Default, except that if the `-i` option was specified, SIGINT shall be ignored.

12662 ### 4.61.6 External Effects

12663 #### 4.61.6.1 Standard Output

12664 The standard output shall be a copy of the standard input.

12665 #### 4.61.6.2 Standard Error

12666 Used only for diagnostic messages.

12667 #### 4.61.6.3 Output Files

12668 If any *file* operands are specified, the standard input shall be copied to each
12669 named file.

12670 ### 4.61.7 Extended Description

12671 None.

12672 ### 4.61.8 Exit Status

12673 0 The standard input was successfully copied to all output files.

12674 >0 An error occurred.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12675 ### 4.61.9 Consequences of Errors

12676 If a write to any successfully opened *file* operand fails, writes to other successfully
12677 opened *file* operands and standard output shall continue, but the exit status shall
12678 be nonzero. Otherwise, the default actions specified in 2.11.9 shall apply.

12679 ### 4.61.10 Rationale. *(This subclause is not a part of P1003.2)*

12680 **Examples, Usage**

12681 The `tee` utility is usually used in a pipeline, to make a copy of the output of some
12682 utility.

12683 The *file* operand is technically optional, but `tee` is no more useful than `cat` when
12684 none is specified.

12685 **History of Decisions Made**

12686 The buffering requirement means that `tee` is not allowed to use C Standard {7}
12687 fully-buffered or line-buffered writes, not that `tee` has to do one-byte reads fol-
12688 lowed by one-byte writes.

12689 It should be noted that early versions of BSD silently ignore any invalid options,
12690 and accept a single – as an alternative to `-i`. They also print the message

12691 `"tee: cannot access %s\n"`, <*pathname*>

12692 if unable to open a file.

12693 Historical implementations ignore write errors. This is explicitly not permitted
12694 by this standard.

12695 Some historical implementations use O_APPEND when providing append mode;
12696 others just *lseek* () to the end of file after opening the file without O_APPEND.
12697 This standard requires functionality equivalent to using O_APPEND ; see 2.9.1.4.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12698 # 4.62 `test` — Evaluate expression

12699 ## 4.62.1 Synopsis

12700 `test [`*expression* `]`

12701 `[ [`*expression* `] ]`

12702 ## 4.62.2 Description

12703 The `test` utility shall evaluate the *expression* and indicate the result of the 1
12704 evaluation by its exit status. An exit status of zero indicates that the expression 1
12705 evaluated as true and an exit status of 1 indicates that the expression evaluated 1
12706 as false. 1

12707 In the second form of the utility, which uses `[ ]`, rather than `test`, the square
12708 brackets shall be separate arguments.

12709 ## 4.62.3 Options

12710 The `test` utility shall not recognize the `--` argument in the manner specified by
12711 utility syntax guideline 10 in 2.10.2.

12712 Implementations shall not support any options. 1

12713 ## 4.62.4 Operands

12714 All operators and elements of primaries shall be presented as separate arguments 2
12715 to the `test` utility.

12716 The following primaries can be used to construct *expression* :

12717 `–b` *file* True if *file* exists and is a block special file.

12718 `–c` *file* True if *file* exists and is a character special file.

12719 `–d` *file* True if *file* exists and is a directory.

12720 `–e` *file* True if *file* exists.

12721 `–f` *file* True if *file* exists and is a regular file.

12722 `–g` *file* True if *file* exists and its set group ID flag is set.

12723 `–n` *string* True if the length of *string* is nonzero.

12724 `–p` *file* True if *file* is a named pipe (FIFO).

12725 `–r` *file* True if *file* exists and is readable.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12726 `-s` *file* True if *file* exists and has a size greater than zero.

12727 `-t` *file_descriptor*
12728 12729 True if the file whose file descriptor number is *file_descriptor* is open and is associated with a terminal.

12730 `-u` *file* True if *file* exists and its set-user- ID flag is set.

12731 12732 12733 `-w` *file* True if *file* exists and is writable. True shall indicate only that the write flag is on. The *file* shall not be writable on a read-only file system even if this test indicates true.

12734 12735 12736 `-x` *file* True if *file* exists and is executable. True shall indicate only that the execute flag is on. If *file* is a directory, true indicates that *file* can be searched.

12737 `-z` *string* True if the length of string *string* is zero.

12738 *string* True if the string *string* is not the null string.

12739 *s1* `=` *s2* True if the strings *s1* and *s2* are identical.

12740 *s1* `!=` *s2* True if the strings *s1* and *s2* are not identical.

12741 *n1* `-eq` *n2* True if the integers *n1* and *n2* are algebraically equal.

12742 *n1* `-ne` *n2* True if the integers *n1* and *n2* are not algebraically equal.

12743 12744 *n1* `-gt` *n2* True if the integer *n1* is algebraically greater than the integer *n2*.

12745 12746 *n1* `-ge` *n2* True if the integer *n1* is algebraically greater than or equal to the integer *n2*.

12747 *n1* `-lt` *n2* True if the integer *n1* is algebraically less than the integer *n2*.

12748 12749 *n1* `-le` *n2* True if the integer *n1* is algebraically less than or equal to the integer *n2*.

12750 12751 A primary can be preceded by the `!` operator to complement its test, as described below. 1 1

12752 The primaries with two elements of the form: 2

12753 *-primary_operator primary_operand* 2

12754 12755 are known as *unary primaries.* The primaries with three elements in either of the two forms: 2 2

12756 *primary_operand -primary_operator primary_operand* 2

12757 *primary_operand primary_operator primary_operand* 2

12758 12759 12760 12761 12762 are known as *binary primaries.* Additional implementation-defined operators and *primary_operator* s may be provided by implementations. They shall be of the form *-operator* where the first character of *operator* is not a digit. The additional implementation-defined operators "`(`" and "`)`" may also be provided by implementations. 2 2 2 2 2

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The algorithm for determining the precedence of the operators and the return value that shall be generated is based on the number of arguments presented to `test`. (However, when using the `[...]` form, the right-bracket final argument shall not be counted in this algorithm.) In the following list, `$1`, `$2`, `$3`, and `$4` represent the arguments presented to `test`.

0 arguments:

Exit false (1).

1 argument:

Exit true (0) if `$1` is not null; otherwise, exit false.

2 arguments:

— If `$1` is `!`, exit true if `$2` is null, false if `$2` is not null.

— If `$1` is a unary primary, exit true if the unary test is true, false if the unary test is false.

— Otherwise, produce unspecified results.

3 arguments:

— If `$2` is a binary primary, perform the binary test of `$1` and `$3`.

— If `$1` is `!`, negate the two-argument test of `$2` and `$3`.

— Otherwise, produce unspecified results.

4 arguments:

— If `$1` is `!`, negate the three-argument test of `$2`, `$3`, and `$4`.

— Otherwise, the results are unspecified.

>4 arguments:

The results are unspecified.

4.62.5 External Influences

4.62.5.1 Standard Input

None.

4.62.5.2 Input Files

None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.62.5.3 Environment Variables

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

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 locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

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

4.62.5.4 Asynchronous Events

Default.

4.62.6 External Effects

4.62.6.1 Standard Output

None.

4.62.6.2 Standard Error

Used only for diagnostic messages.

4.62.6.3 Output Files

None.

4.62.7 Extended Description

None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12817 ### 4.62.8 Exit Status

12818 The `test` utility shall exit with one of the following values:

12819 0 *expression* evaluated to true.

12820 1 *expression* evaluated to false or *expression* was missing.

12821 >1 An error occurred.

12822 ### 4.62.9 Consequences of Errors

12823 Default.

12824 ### 4.62.10 Rationale. *(This subclause is not a part of P1003.2)*

12825 **Examples, Usage**

12826 *Editor's Note: The rationale has been rearranged quite a bit. Only new, not* 1
12827 *moved, text has been diffmarked.* 1

12828 Historical systems have supported more than four arguments, but there has been 1
12829 a fundamental disagreement between BSD and System V on certain combinations 1
12830 of arguments. Since no accommodation could be reached between the two ver- 1
12831 sions of `test` without breaking numerous applications, the version of `test` in 1
12832 POSIX. 2 specifies only the relatively simple tests and relies on the syntax of the 1
12833 shell command language for the construction of more complex expressions. Using 1
12834 the POSIX. 2 rules produces completely reliable, portable scripts, which is not 1
12835 always possible using either of the historical forms. Some of the historical 1
12836 behavior is described here to aid conversion of scripts with complex `test` expres- 1
12837 sions. 1

12838 Both BSD and System V support the combining of primaries with the following 1
12839 constructs: 1

12840 *expression1* `-a` *expression2* True if both *expression1* and *expression2* are 1
12841 true. 1

12842 *expression1* `-o` *expression2* True if at least one of *expression1* and *expres-* 1
12843 *sion2* are true. 1

12844 `(` *expression* `)` True if *expression* is true. 1

12845 In evaluating these more complex combined expressions, the following precedence 1
12846 rules are used: 1

12847 — The unary primaries have higher precedence than the algebraic binary pri- 1
12848 maries. 1

12849 — On BSD systems, the unary primaries have higher precedence than the 1
12850 string binary primaries. On System V systems, the unary primaries have 1
12851 lower precedence than the string binary primaries. 1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— The unary and binary primaries have higher precedence than the unary *string* primary.

— The `!` operator has higher precedence than the `-a` operator and the `-a` operator has higher precedence than the `-o` operator.

— The `-a` and `-o` operators are left associative.

— The parentheses can be used to alter the normal precedence and associativity.

The following guidance is offered for the use of the historical expressions:

— Scripts should be careful when dealing with user-supplied input that could be confused with primaries and operators. Unless the application writer knows all the cases that produce input to the script, invocations like:

```
test "$1" -a "$2"
```

should be written as:

```
test "$1" && test "$2"
```

to avoid problems if a user-supplied values such as `$1` set to `!` and `$2` set to the null string. That is, in cases where portability between implementations based on BSD and System V systems is of concern, replace:

```
test expr1 -a expr2
```

with:

```
test expr1 && test expr2
```

and replace:

```
test expr1 -o expr2
```

with:

```
test expr1 || test expr2
```

but note that, in `test`, `-a` has higher precedence than `-o` while `&&` and `||` have equal precedence in the shell.

Parentheses or braces can be used in the shell command language to effect grouping. Historical `test` implementations also support parentheses, but they must be escaped when using `sh`; for example:

```
test \( expr1 -a expr2 \) -o expr3
```

This command is not always portable. The following form can be used instead:

```
( test expr1 && test expr2 ) || test expr3
```

— The two commands:

```
test "$1"
test ! "$1"
```

could not be used reliably on historical systems. Unexpected results would

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

occur if such a *string* expression were used and `$1` expanded to `!`, `(`, or a known unary primary. Better constructs were:

```
test -n "$1"
test -z "$1"
```

respectively. These suggested replacements have always worked on historical BSD-based implementations, and work on historical System V-based implementations as long as `$1` does not expand to `=` or `!=`. Using the POSIX.2 rules, any of the four forms shown will work for any possible value of `$1`.

— Historical systems were also unreliable given the common construct:

```
test "$response" = "expected string"
```

One of the following was a more reliable form:

```
test "X$response" = "Xexpected string"
test "expected string" = "$response"
```

Note that the second form assumes that `expected string` could not be confused with any any unary primary. If `expected string` starts with `-`, `(`, `!`, or even `=`, the first form should be used instead. Using the POSIX.2 rules, any of the three comparison forms is reliable, given any input. (However, note that the strings are quoted in all cases.)

The BSD and System V versions of `-f` are not the same. The BSD definition was:

`-f` *file* True if *file* exists and is not a directory.

The *SVID* version (true if the file exists and is a regular file) was chosen for this standard because its use is consistent with the `-b`, `-c`, `-d`, and `-p` operands (*file* exists and is a specific file type).

The `-e` primary, possessing similar functionality to that provided by the C-shell, was added because it provides the only way for a shell script to find out if a file exists without trying to open the file. (Since implementations are allowed to add additional file types, a portable script cannot use:

```
test -b foo -o -c foo -o -d foo -o -f foo -o -p foo
```

to find out if `foo` is an existing file.) On historical BSD systems, the existence of a file could be determined by:

```
test -f foo -o -d foo
```

but there was no easy way to determine that an existing file was a regular file. An earlier draft used the KornShell `-a` primary (with the same meaning), but this was changed to `-e` because there were concerns about the high probability of humans confusing the `-a` primary with the `-a` binary operator.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12925 **History of Decisions Made**

12926 The –a and –o binary operators and the grouping parentheses were omitted from 1
12927 POSIX.2 due to a difference between existing implementations of the `test` utility 1
12928 in the precedence of the binary primaries `=` and `!=` compared to the unary pri- 1
12929 maries –b, –c, –d, –f, –g, –n, –p, –r, –s, –t, –u, –w, –x, and –z. On BSD, Ver- 1
12930 sion 7, PWB, and 32V systems the unary primaries have higher precedence than
12931 the binary operators; on System III and System V implementations, the binary
12932 operators `=` and `!=` have higher precedence. The change was apparently made for
12933 System III so that the construct:

12934 ```
test "$1" = "$2"
```

12935 could be made to work even if `$1` started with `-`. It is believed that this change
12936 was a mistake because:

12937 — It is not a complete solution; if `$1` expands to `(` or `!`, it still will not work.

12938 — It makes it impossible to use the unary primaries –n and –z to test for a
12939 null string if there is any chance that the string will expand to `=`.

12940 — More importantly, there was the well known workaround of specifying:

12941 ```
test X"$1" = X"$2"
```

12942 that always worked.

12943 Unfortunately, when the `=` and `!=` binary primaries were given precedence over
12944 the unary primaries, there was no workaround provided for scripts that wanted to
12945 reliably specify something like:

12946 ```
test -n "$1"
```

12947 because if `$1` expands to `=`, it gives a syntax error.

12948 There was some discussion of outlawing the System V behavior and requiring the 1
12949 more logical precedence that originated in its predecessors and remains in BSD- 1
12950 based systems. However, there are simply too many historical applications that 1
12951 would break if System V were required to make this change; this number dwarfed 1
12952 the number of scripts using combination logic that would then no longer be 1
12953 strictly portable. 1

12954 POSIX.2 requires that if `test` is called with one, two, three, or four operands it 1
12955 correctly interprets the expression even if there is an alternate syntax tree that 1
12956 could lead to a syntax error. It eliminates the requirement that many string com- 1
12957 parisons be protected with leading characters, such as 1

12958 ```
test X"$1" = X"$2"
```
1

12959 and allows the single-argument *string* form to be used with all possible inputs. 1

12960 The following examples show some of the changes that are required to be made to
12961 make historical BSD and System V-based implementations of `test` conform to
12962 this standard:

12963 `test -d =`	POSIX.2	True if there is a directory named `=`

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

	BSD	True if there is a directory named =
	System V	Syntax error; = needs two operands
`test -d = -f`	POSIX.2	False
	BSD	Syntax error; it expects –a or –o after -d =
	System V	False

Implementations are prohibited from extending `test` with options because it would make the '`test` *string* '' case ambiguous for inputs that might match an extended option. Implementations can add primaries and operators, as indicated.

The following options were not included in POSIX.2, although they are provided by some historical implementations, since these facilities and concepts are not supported by POSIX.1 {8}, nor defined in POSIX.2. These operands should not be used by new implementations for other purposes.

`–h` *file* True if *file* exists and is a symbolic link.

`–k` *file* True if *file* exists and its sticky bit is set.

`–L` *file* True if *file* is a symbolic link.

`–C` *file* True if *file* is a contiguous file.

`–S` *file* True if *file* is a socket.

`–V` *file* True if *file* is a version file.

The following option was not included because it was undocumented in most implementations, has been removed from some implementations (including System V), and the functionality is provided by the shell (see 3.6.2).

`–l` *string* The length of the string *string* .

The `–b`, `–c`, `–g`, `–p`, `–u`, and `–x` operands are derived from the *SVID* ; historical BSD does not provide them. The `–k` operand is derived from System V; historical BSD does not provide it.

On historical BSD systems, `test –w` *directory* always returned false because `test` tried to open the directory for writing, which always fails.

Some additional primaries newly invented or from the KornShell appeared in an earlier draft as part of the Conditional Command (`[[ ]]`): *s1* > *s2*, *s1* < *s2*, *str* = *pattern*, *str* `!=` *pattern*, *f1* `–nt` *f2*, *f1* `–ot` *f2*, and *f1* `–ef` *f2*. They were not carried forward into the `test` utility when the Conditional Command was removed from the shell because they have not been included in the `test` utility built into historical implementations of the `sh` utility.

The `–t` *file_descriptor* primary is shown with a mandatory argument because the grammar is ambiguous if it can be omitted. Historical implementations have allowed it to be omitted, providing a default of 1.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13000 ## 4.63 `touch` — Change file access and modification times

13001 ### 4.63.1 Synopsis

13002 `touch [-acm][ -r` *ref_file* `| -t` *time* `]` *file* ...

13003 *Obsolescent Version* :

13004 `touch [-acm][`*date_time* `]` *file* ...

13005 ### 4.63.2 Description

13006 The `touch` utility shall change the modification and/or access times of files. The
13007 modification time is equivalent to the value of the *st_mtime* member of the *stat*
13008 structure for a file, as described in POSIX .1 {8};the access time is equivalent to the
13009 value of *st_atime* .

13010 The time used can be specified by the `-t` *time* option-argument, the correspond-
13011 ing time field(s) of the file referenced by the `-r` *ref_file* option-argument, or the
13012 *date_time* operand, as specified in the following subclauses. If none of these are
13013 specified, `touch` shall use the current time [the value returned by the equivalent
13014 of the POSIX .1 {8}*time* () function].

13015 For each *file* operand, `touch` shall perform actions equivalent to the following
13016 functions defined in POSIX .1 {8}:

13017 (1) If *file* does not exist, a *creat* () function call is made with the *file* operand
13018 used as the *path* argument and the value of the bitwise inclusive OR of
13019 S_IRUSR , S_IWUSR , S_IRGRP , S_IWGRP , S_IROTH , and S_IWOTH used
13020 as the *mode* argument.

13021 (2) The *utime* () function is called with the following arguments:

13022 (a) The *file* operand is used as the *path* argument.

13023 (b) The *utimbuf* structure members *actime* and *modtime* are deter-
13024 mined as described under 4.63.3.

13025 ### 4.63.3 Options

13026 The `touch` utility shall conform to the utility argument syntax guidelines
13027 described in 2.10.2.

13028 The following options shall be supported by the implementation:

13029 `-a` Change the access time of *file*. Do not change the modification
13030 time unless `-m` is also specified.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.