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 'test.1p' command

$ man test.1p

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

test ? evaluate expression

SYNOPSIS

test [expression]

[[expression]]

DESCRIPTION

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

In the second form of the utility, where the utility name used is [rather than test, the application shall ensure that the closing square bracket is a separate argument. The test and [utilities may be imple? mented as a single linked utility which examines the basename of the zeroth command line argument to determine whether to behave as the test or [variant. Applications using the exec() family of functions to exe? cute these utilities shall ensure that the argument passed in arg0 or

argv[0] is '[' when executing the [utility and has a basename of "test" when executing the test utility.

OPTIONS

The test utility shall not recognize the "--" argument in the manner specified by Guideline 10 in the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

No options shall be supported.

OPERANDS

The application shall ensure that all operators and elements of pri? maries are presented as separate arguments to the test utility.

The following primaries can be used to construct expression:

-b pathname

True if pathname resolves to an existing directory entry for a block special file. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that is not a block special file.

-c pathname

True if pathname resolves to an existing directory entry for a character special file. False if pathname cannot be re? solved, or if pathname resolves to an existing directory en? try for a file that is not a character special file.

-d pathname

True if pathname resolves to an existing directory entry for a directory. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that is not a directory.

-e pathname

True if pathname resolves to an existing directory entry. False if pathname cannot be resolved.

-f pathname

True if pathname resolves to an existing directory entry for a regular file. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file

that is not a regular file.

-g pathname

True if pathname resolves to an existing directory entry for a file that has its set-group-ID flag set. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that does not have its set-group-ID flag set.

-h pathname

True if pathname resolves to an existing directory entry for a symbolic link. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that is not a symbolic link. If the final component of path?name is a symbolic link, that symbolic link is not followed.

-L pathname

True if pathname resolves to an existing directory entry for a symbolic link. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that is not a symbolic link. If the final component of path?name is a symbolic link, that symbolic link is not followed.

-n string True if the length of string is non-zero; otherwise, false.

-p pathname

True if pathname resolves to an existing directory entry for a FIFO. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that is not a FIFO.

-r pathname

True if pathname resolves to an existing directory entry for a file for which permission to read from the file will be granted, as defined in Section 1.1.1.4, File Read, Write, and Creation. False if pathname cannot be resolved, or if path?name resolves to an existing directory entry for a file for which permission to read from the file will not be granted.

-S pathname

True if pathname resolves to an existing directory entry for a socket. False if pathname cannot be resolved, or if path? name resolves to an existing directory entry for a file that is not a socket.

-s pathname

True if pathname resolves to an existing directory entry for a file that has a size greater than zero. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that does not have a size greater than zero.

-t file_descriptor

True if file descriptor number file_descriptor is open and is associated with a terminal. False if file_descriptor is not a valid file descriptor number, or if file descriptor number file_descriptor is not open, or if it is open but is not as? sociated with a terminal.

-u pathname

True if pathname resolves to an existing directory entry for a file that has its set-user-ID flag set. False if pathname cannot be resolved, or if pathname resolves to an existing directory entry for a file that does not have its set-user-ID flag set.

-w pathname

True if pathname resolves to an existing directory entry for a file for which permission to write to the file will be granted, as defined in Section 1.1.1.4, File Read, Write, and Creation. False if pathname cannot be resolved, or if path? name resolves to an existing directory entry for a file for which permission to write to the file will not be granted.

-x pathname

True if pathname resolves to an existing directory entry for a file for which permission to execute the file (or search it, if it is a directory) will be granted, as defined in Sec?

tion 1.1.1.4, File Read, Write, and Creation. False if path?
name cannot be resolved, or if pathname resolves to an exist?
ing directory entry for a file for which permission to exe?
cute (or search) the file will not be granted.

-z string True if the length of string string is zero; otherwise,
false.

string True if the string string is not the null string; otherwise,
false.

s1 = s2 True if the strings s1 and s2 are identical; otherwise,
false.

s1 != s2 True if the strings s1 and s2 are not identical; otherwise,
false.

n1 -eq n2 True if the integers n1 and n2 are algebraically equal; oth?
erwise, false.

n1 -ne n2 True if the integers n1 and n2 are not algebraically equal;
otherwise, false.

n1 -gt n2 True if the integer n1 is algebraically greater than the in?
teger n2; otherwise, false.

n1 -ge n2 True if the integer n1 is algebraically greater than or equal
to the integer n2; otherwise, false.

n1 -lt n2 True if the integer n1 is algebraically less than the integer
n2; otherwise, false.

n1 -le n2 True if the integer n1 is algebraically less than or equal to
the integer n2; otherwise, false.

expression1 -a expression2
True if both expression1 and expression2 are true; otherwise,
false. The -a binary primary is left associative. It has a
higher precedence than -o.

expression1 -o expression2
True if either expression1 or expression2 is true; otherwise,
false. The -o binary primary is left associative.

With the exception of the -h pathname and -L pathname primaries, if a
pathname argument is a symbolic link, test shall evaluate the expres?

sion by resolving the symbolic link and using the file referenced by the link.

These primaries can be combined with the following operators:

! expression

True if expression is false. False if expression is true.

(expression)

True if expression is true. False if expression is false. The parentheses can be used to alter the normal precedence and associativity.

The primaries with two elements of the form:

-primary_operator primary_operand

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

primary_operand -primary_operator primary_operand

primary_operand primary_operator primary_operand

are known as binary primaries. Additional implementation-defined operators and primary_operators may be provided by implementations. They shall be of the form -operator where the first character of operator is not a digit.

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-square-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.

* If $1 is '(' and $3 is ')', perform the unary test of $2. On systems that do not support the XSI option, the results are unspecified if $1 is '(' and $3 is ')'.

* Otherwise, produce unspecified results.

4 arguments:

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

* If $1 is '(' and $4 is ')', perform the two-argument test of $2 and $3. On systems that do not support the XSI option, the results are unspecified if $1 is '(' and $4 is ')'.

* Otherwise, the results are unspecified.

>4 arguments:

The results are unspecified.

On XSI-conformant systems, combinations of primaries and operators shall be evaluated using the precedence and asso? ciativity rules described previously. In addition, the string comparison binary primaries '=' and "!=" shall have a higher precedence than any unary primary.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of test:

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).

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

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

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 expression evaluated to true.

1 expression evaluated to false or expression was missing.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

The XSI extensions specifying the -a and -o binary primaries and the '(' and ')' operators have been marked obsolescent. (Many expressions using them are ambiguously defined by the grammar depending on the spe? cific expressions being evaluated.) Scripts using these expressions should be converted to the forms given below. Even though many imple? mentations will continue to support these obsolescent forms, scripts should be extremely careful when dealing with user-supplied input that could be confused with these and other primaries and operators. Unless the application developer 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 maximal portability 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 ef? fect grouping.

Parentheses must be escaped when using sh; for example:

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

This command is not always portable even on XSI-conformant systems de? pending on the expressions specified by expr1, expr2, and expr3. 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 some historical systems. Unexpected re? sults would occur if such a string expression were used and $1 expanded to '!', '(', or a known unary primary. Better constructs are:

test -n "$1"

test -z "$1"

respectively.

Historical systems have also been unreliable given the common con? struct:

test "$response" = "expected string"

One of the following is 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 unary primary. If expected string starts with '-', '(', '!', or even '=', the first form should be used instead. Using the preceding rules without the XSI marked extensions, any of the three comparison forms is reliable, given any input. (However, note that the strings are quoted in all cases.)

Because the string comparison binary primaries, '=' and "!=", have a higher precedence than any unary primary in the greater than 4 argument case, unexpected results can occur if arguments are not properly pre? pared. For example, in:

test -d $1 -o -d $2

If $1 evaluates to a possible directory name of '=', the first three arguments are considered a string comparison, which shall cause a syn? tax error when the second -d is encountered. One of the following forms prevents this; the second is preferred:

test \(-d "$1" \) -o \(-d "$2" \)

test -d "$1" || test -d "$2"

Also in the greater than 4 argument case:

test "$1" = "bat" -a "$2" = "ball"

syntax errors occur if $1 evaluates to '(' or '!'. One of the follow?

ing forms prevents this; the third is preferred:

test "X$1" = "Xbat" -a "X$2" = "Xball"

test "$1" = "bat" && test "$2" = "ball"

test "X$1" = "Xbat" && test "X$2" = "Xball"

Note that none of the following examples are permitted by the syntax

described:

[-f file]

[-f file]

[-f file]

[-f file

test -f file]

In the first two cases, if a utility named [?f exists, that utility

would be invoked, and not test. In the remaining cases, the brackets

are mismatched, and the behavior is unspecified. However:

test !]

does have a defined meaning, and must exit with status 1. Similarly:

test]

must exit with status 0.

EXAMPLES

1. Exit if there are not two or three arguments (two variations):

   if [$# -ne 2] && [$# -ne 3]; then exit 1; fi

   if [$# -lt 2] || [$# -gt 3]; then exit 1; fi

2. Perform a mkdir if a directory does not exist:

   test ! -d tempdir && mkdir tempdir

3. Wait for a file to become non-readable:

   while test -r thefile

   do

   sleep 30

   done

   echo '"thefile" is no longer readable'

4. Perform a command if the argument is one of three strings (two variations):

```
if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
then
    command
fi
case "$1" in
    pear|grape|apple) command ;;
esac
```

RATIONALE

The KornShell-derived conditional command (double bracket [[]]) was re? moved from the shell command language description in an early proposal. Objections were raised that the real problem is misuse of the test com? mand ([), and putting it into the shell is the wrong way to fix the problem. Instead, proper documentation and a new shell reserved word (!) are sufficient.

Tests that require multiple test operations can be done at the shell level using individual invocations of the test command and shell logi? cals, rather than using the error-prone -o flag of test.

XSI-conformant systems support more than four arguments.

XSI-conformant systems support the combining of primaries with the fol? lowing constructs:

expression1 -a expression2

True if both expression1 and expression2 are true.

expression1 -o expression2

True if at least one of expression1 and expression2 are true.

(expression)

True if expression is true.

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

* The unary primaries have higher precedence than the algebraic bi? nary primaries.
* The unary primaries have lower precedence than the string binary

primaries.

* 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 as? sociativity.

The BSD and System V versions of -f are not the same. The BSD defini? tion 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 volume of POSIX.1?2017 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 por? table 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 reg? ular file. An early proposal 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.

The following options were not included in this volume of POSIX.1?2017, although they are provided by some implementations. These operands should not be used by new implementations for other purposes:

-k file True if file exists and its sticky bit is set.

-C file True if file is a contiguous file.

-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 (in? cluding System V), and the functionality is provided by the shell (see Section 2.6.2, Parameter Expansion.

-l string The length of the string string.

The -b, -c, -g, -p, -u, and -x operands are derived from the SVID; his? torical BSD does not provide them. The -k operand is derived from Sys? tem V; historical BSD does not provide it.

On historical BSD systems, test -w directory always returned false be? cause test tried to open the directory for writing, which always fails.

Some additional primaries newly invented or from the KornShell appeared in an early proposal 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 implementa? tions of the sh utility.

The -t file_descriptor primary is shown with a mandatory argument be? cause the grammar is ambiguous if it can be omitted. Historical imple? mentations have allowed it to be omitted, providing a default of 1.

It is noted that '[' is not part of the portable filename character set; however, since it is required to be encoded by a single byte, and is part of the portable character set, the name of this utility forms a character string across all supported locales.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.1.1.4, File Read, Write, and Creation, find

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

COPYRIGHT