
*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 'tr.1p' command*

***$ man tr.1p***

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

tr ? translate characters

SYNOPSIS

tr [-c|-C] [-s] string1 string2

tr -s [-c|-C] string1

tr -d [-c|-C] string1

tr -ds [-c|-C] string1 string2

DESCRIPTION

The tr utility shall copy the standard input to the standard output with substitution or deletion of selected characters. The options spec? ified and the string1 and string2 operands shall control translations that occur while copying characters and single-character collating ele? ments.

OPTIONS

The tr utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-c Complement the set of values specified by string1. See the EXTENDED DESCRIPTION section.

-C Complement the set of characters specified by string1. See the EXTENDED DESCRIPTION section.

-d Delete all occurrences of input characters that are specified by string1.

-s Replace instances of repeated characters with a single char? acter, as described in the EXTENDED DESCRIPTION section.

OPERANDS

The following operands shall be supported:

string1, string2

Translation control strings. Each string shall represent a set of characters to be converted into an array of characters used for the translation. For a detailed description of how the strings are interpreted, see the EXTENDED DESCRIPTION section.

STDIN

The standard input can be any type of file.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of tr:

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_COLLATE

Determine the locale for the behavior of range expressions and equivalence classes.

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 the behav? ior of character classes.

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

The tr output shall be identical to the input, with the exception of the specified transformations.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The operands string1 and string2 (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the con? structs result in multi-character collating elements, tr shall exclude, without a diagnostic, those multi-character elements from the resulting array.

character Any character not described by one of the conventions below shall represent itself.

\octal Octal sequences can be used to represent characters with spe? cific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the ar?

ray. Multi-byte characters require multiple, concatenated es?
cape sequences of this type, including the leading <back?
slash> for each byte.

\character
The <backslash>-escape sequences in the Base Definitions vol?
ume of POSIX.1?2017, Table 5-1, Escape Sequences and Associ?
ated Actions ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v')
shall be supported. The results of using any other character,
other than an octal digit, following the <backslash> are un?
specified. Also, if there is no character following the
<backslash>, the results are unspecified.

c-c In the POSIX locale, this construct shall represent the range
of collating elements between the range endpoints (as long as
neither endpoint is an octal sequence of the form \octal),
inclusive, as defined by the collation sequence. The charac?
ters or collating elements in the range shall be placed in
the array in ascending collation sequence. If the second end?
point precedes the starting endpoint in the collation se?
quence, it is unspecified whether the range of collating ele?
ments is empty, or this construct is treated as invalid. In
locales other than the POSIX locale, this construct has un?
specified behavior.
If either or both of the range endpoints are octal sequences
of the form \octal, this shall represent the range of spe?
cific coded values between the two range endpoints, inclu?
sive.

[:class:] Represents all characters belonging to the defined character
class, as defined by the current setting of the LC_CTYPE lo?
cale category. The following character class names shall be
accepted when specified in string1:
alnum blank digit lower punct upper
alpha cntrl graph print space xdigit
In addition, character class expressions of the form [:name:]

shall be recognized in those locales where the name keyword has been given a charclass definition in the LC_CTYPE cate? gory.

When both the -d and -s options are specified, any of the character class names shall be accepted in string2. Other? wise, only character class names lower or upper are valid in string2 and then only if the corresponding character class (upper and lower, respectively) is specified in the same rel? ative position in string1. Such a specification shall be in? terpreted as a request for case conversion. When [:lower:] appears in string1 and [:upper:] appears in string2, the ar? rays shall contain the characters from the toupper mapping in the LC_CTYPE category of the current locale. When [:upper:] appears in string1 and [:lower:] appears in string2, the ar? rays shall contain the characters from the tolower mapping in the LC_CTYPE category of the current locale. The first char? acter from each mapping pair shall be in the array for string1 and the second character from each mapping pair shall be in the array for string2 in the same relative position.

Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.

If the name specified for class does not define a valid char? acter class in the current locale, the behavior is undefined.

[=equiv=] Represents all characters or collating elements belonging to the same equivalence class as equiv, as defined by the cur? rent setting of the LC_COLLATE locale category. An equiva? lence class expression shall be allowed only in string1, or in string2 when it is being used by the combined -d and -s options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.

[x*n] Represents n repeated occurrences of the character x. Be? cause this expression is used to map multiple characters to

one, it is only valid when it occurs in string2. If n is omitted or is zero, it shall be interpreted as large enough to extend the string2-based sequence to the length of the string1-based sequence. If n has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be inter? preted as a decimal value.

When the -d option is not specified:

* If string2 is present, each input character found in the array specified by string1 shall be replaced by the character in the same relative position in the array specified by string2. If the array specified by string2 is shorter that the one specified by string1, or if a character occurs more than once in string1, the results are unspecified.
* If the -C option is specified, the complements of the characters specified by string1 (the set of all characters in the current character set, as defined by the current setting of LC_CTYPE, ex? cept for those actually specified in the string1 operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of LC_COLLATE.
* If the -c option is specified, the complement of the values speci? fied by string1 shall be placed in the array in ascending order by binary value.
* Because the order in which characters specified by character class expressions or equivalence class expressions is undefined, such ex? pressions should only be used if the intent is to map several char? acters into one. An exception is case conversion, as described pre? viously.

When the -d option is specified:

* Input characters found in the array specified by string1 shall be deleted.
* When the -C option is specified with -d, all characters except those specified by string1 shall be deleted. The contents of string2 are ignored, unless the -s option is also specified.

* When the -c option is specified with -d, all values except those specified by string1 shall be deleted. The contents of string2 shall be ignored, unless the -s option is also specified.
* The same string cannot be used for both the -d and the -s option; when both options are specified, both string1 (used for deletion) and string2 (used for squeezing) shall be required.

When the -s option is specified, after any deletions or translations have taken place, repeated sequences of the same character shall be re? placed by one occurrence of the same character, if the character is found in the array specified by the last operand. If the last operand contains a character class, such as the following example:

tr -s '[:space:]'

the last operand's array shall contain all of the characters in that character class. However, in a case conversion, as described previ? ously, such as:

tr -s '[:upper:]' '[:lower:]'

the last operand's array shall contain only those characters defined as the second characters in each of the toupper or tolower character pairs, as appropriate.

An empty string used for string1 or string2 produces undefined results.

EXIT STATUS

The following exit values shall be returned:

0 All input was processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

If necessary, string1 and string2 can be quoted to avoid pattern match? ing by the shell.

If an ordinary digit (representing itself) is to follow an octal se? quence, the octal sequence must use the full three digits to avoid am? biguity.

When string2 is shorter than string1, a difference results between his? torical System V and BSD systems. A BSD system pads string2 with the last character found in string2. Thus, it is possible to do the fol? lowing:

```
tr 0123456789 d
```

which would translate all digits to the letter 'd'. Since this area is specifically unspecified in this volume of POSIX.1?2017, both the BSD and System V behaviors are allowed, but a conforming application cannot rely on the BSD behavior. It would have to code the example in the fol? lowing way:

```
tr 0123456789 '[d*]'
```

It should be noted that, despite similarities in appearance, the string operands used by tr are not regular expressions.

Unlike some historical implementations, this definition of the tr util? ity correctly processes NUL characters in its input stream. NUL charac? ters can be stripped by using:

```
tr -d '\000'
```

EXAMPLES

1. The following example creates a list of all words in file1 one per line in file2, where a word is taken to be a maximal string of let? ters.

   ```
   tr -cs "[:alpha:]" "[\n*]" <file1 >file2
   ```

2. The next example translates all lowercase characters in file1 to uppercase and writes the results to standard output.

   ```
   tr "[:lower:]" "[:upper:]" <file1
   ```

3. This example uses an equivalence class to identify accented vari? ants of the base character 'e' in file1, which are stripped of dia? critical marks and written to file2.

   ```
   tr "[=e=]" "[e*]" <file1 >file2
   ```

RATIONALE

In some early proposals, an explicit option -n was added to disable the historical behavior of stripping NUL characters from the input. It was considered that automatically stripping NUL characters from the input

was not correct functionality. However, the removal of -n in a later proposal does not remove the requirement that tr correctly process NUL characters in its input stream. NUL characters can be stripped by using tr -d '\000'.

Historical implementations of tr differ widely in syntax and behavior. For example, the BSD version has not needed the bracket characters for the repetition sequence. The tr utility syntax is based more closely on the System V and XPG3 model while attempting to accommodate historical BSD implementations. In the case of the short string2 padding, the de? cision was to unspecify the behavior and preserve System V and XPG3 scripts, which might find difficulty with the BSD method. The assump? tion was made that BSD users of tr have to make accommodations to meet the syntax defined here. Since it is possible to use the repetition se? quence to duplicate the desired behavior, whereas there is no simple way to achieve the System V method, this was the correct, if not desir? able, approach.

The use of octal values to specify control characters, while having historical precedents, is not portable. The introduction of escape se? quences for control characters should provide the necessary portabil? ity. It is recognized that this may cause some historical scripts to break.

An early proposal included support for multi-character collating ele? ments. It was pointed out that, while tr does employ some syntactical elements from REs, the aim of tr is quite different; ranges, for exam? ple, do not have a similar meaning (``any of the chars in the range matches'', versus ``translate each character in the range to the output counterpart''). As a result, the previously included support for multi-character collating elements has been removed. What remains are ranges in current collation order (to support, for example, accented charac? ters), character classes, and equivalence classes.

In XPG3 the [:class:] and [=equiv=] conventions are shown with double brackets, as in RE syntax. However, tr does not implement RE princi? ples; it just borrows part of the syntax. Consequently, [:class:] and

[=equiv=] should be regarded as syntactical elements on a par with [x*n], which is not an RE bracket expression.

The standard developers will consider changes to tr that allow it to translate characters between different character encodings, or they will consider providing a new utility to accomplish this.

On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
tr '[a-z]' '[A-Z]'
```

However, BSD-based systems did not require the brackets, and this con? vention is used here to avoid breaking large numbers of BSD scripts:

```
tr a-z A-Z
```

The preceding System V script will continue to work because the brack? ets, treated as regular characters, are translated to themselves. How? ever, any System V script that relied on "a?z" representing the three characters 'a', '-', and 'z' have to be rewritten as "az-".

The ISO POSIX?2:1993 standard had a -c option that behaved similarly to the -C option, but did not supply functionality equivalent to the -c option specified in POSIX.1?2008.

The earlier version also said that octal sequences referred to collat? ing elements and could be placed adjacent to each other to specify multi-byte characters. However, it was noted that this caused ambigui? ties because tr would not be able to tell whether adjacent octal se? quences were intending to specify multi-byte characters or multiple single byte characters. POSIX.1?2008 specifies that octal sequences al? ways refer to single byte binary values when used to specify an end? point of a range of collating elements.

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

sed

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