## NAME

**tr** — transliterate bytes

## SYNOPSIS

**tr** [**-cts**] *from to*
**tr** [**-c**] **-s** *squeeze*
**tr** [**-c**] **-d** *delete*
**tr** [**-c**] **-ds** *delete squeeze*

## DESCRIPTION

**tr** reads the standard input stream byte-wise, performs two transformations, one after another, and writes the resulting bytes to the standard output stream:

1. transliteration of bytes found in one set to bytes at corresponding offsets in another, and
2. squeeze of runs of identical bytes which exist in a set into their first occurrence.

The final position of any given byte is considered as its canonical offset within the source set.

In all cases, sets are expanded *first*, before any other processing, according to the rules below.

If **-s** is specified, the set of bytes to squeeze is the latter specified, or the empty set otherwise.

If **-d** is specified, the transliteration step is performed from the *delete* set to the empty set; otherwise, if **-s** is specified *and* one set is provided, it's performed between the empty set and the empty set; otherwise it's performed from the *from* set to the *to* set — this is otherwise known as "translation".

During translation *to* may not be empty, and *from* must be of at least the same length as *to*; if it isn't:

| | |
|---|---|
| by default | duplicate the last byte of *to* until the lengths match; |
| with **-t** | truncate *from* to the length of *to*. |

## OPTIONS

| | |
|---|---|
| **-c**, **-C**, **--complement** | Invert (complement) the first set — make it contain all bytes except for the ones previously therein, sorted, as the last step of expansion. |
| **-d**, **--delete** | Map from the first string onto the empty set. |
| **-s**, **--squeeze-repeats** | Squeeze bytes from the latter specified set before output, such that all runs of any given byte within it are reduced to a length of *1*. This happens after transliteration. |
| **-t**, **--truncate-set1** | During translation, truncate the source string to the length of the target string, instead of expanding the latter, matching the AT&T UNIX behaviour instead of the Version 7 AT&T UNIX ("BSD") behaviour. Ignored in other modes. |

## SET EXPANSION

In all places where an arbitrary byte is allowed, it can instead be one of the following escapes:

| | |
|---|---|
| ****, **\a**, **\b**, **\t** **\n**, **\v**, **\f**, **\r** | The backslash, bell, backspace, tab, line feed, vertical tab, form feed, and carriage return characters, respectively. |
| *O*[*O*[*O*]] | Greedily-parsed octal byte of value *OOO*, ended prematurely only by a character not allowed in an octal number. |
| *c* | Literal *c*, without considering any other sequences. |

Additionally, the following expansions are recognised:

| | |
|---|---|
| *f*-*t* | Inclusive range of all characters between *f* and *t*, in ascending order. Recognised only if *t* is at least *f*. |
| **[:***class***:]** | Where *class* may be any of<br>**alnum** alphanumeric characters<br>**alpha** letters |

**blank** blanks
**cntrl** control characters
**digit** digits
**graph** graphic characters — printable except space
**lower** lower-case letters
**print** printable characters
**punct** punctuation characters
**space** whitespace
**upper** upper-case letters
**xdigit** hexadecimal digits

or a locale-specific character class (but see **STANDARDS**, below). These correspond directly to respective `is`*`class`*(3) functions, applied consecutively as a filter on all bytes, according to the current locale; this ordering is compatible with 4.4BSD.

IEEE Std 1003.1-2024 ("POSIX.1") only guarantees equivalent relative ordering for **[:lower:]** and **[:upper:]**, so reliance upon any given ordering of characters generated by character classes may not be portable.

**[=***e***=]** Just single-byte *e*, but see **STANDARDS**, below.

**[***c****], [***c*******len***]** If *len* is *0* or missing, repeat *c* until length of set matches length of the first set (not available when expanding the first set), otherwise repeat *c len* times. *len* is a decimal integer, or octal if it starts with **0**.

If enabled, **-c** occurs directly after processing these transformations.

## EXAMPLES

Extract all words (maximal runs of letters) from *form*:

```
$ cat form
Groceries for February:
    Bananas    3.5kg     $4.51
    Kiwis      2kg       $3.19     Call Siegfried to explain short!
    Bread                $20.21
$ tr -cs '[:alpha:]' '\n' < form     # Only compatible with the BSD!
$ tr -cs '[:alpha:]' '[\n*]' < form
Groceries
for
February
Bananas
kg
Kiwis
kg
Call
Siegfried
to
explain
short
Bread
```

Capitalise that same form:

```
$ tr "[:lower:]" "[:upper:]" < form
GROCERIES FOR FEBRUARY:
    BANANAS    3.5KG     $4.51
    KIWIS      2KG       $3.19     CALL SIEGFRIED TO EXPLAIN SHORT!
    BREAD                $20.21
```

## STANDARDS

Extends IEEE Std 1003.1-2024 (“POSIX.1”) for compatibility with AT&T UNIX: collation (**[=***e***=]** equivalence classes and **-C**) is unsupported — **[=***e***=]** is equivalent to *e* (and must be a single byte), and **-C** is equivalent to **-c**, which is compatible with, respectively, 4.4BSD and the GNU system (as well as the plurality of implementations). Nominally, **[=***e***=]** represents all *characters* with the same equivalence class as *e*, and **-C** selects the inverse set of characters, rather than bytes.

**[:***class***:]** may be any character class supported by the current locale (POSIX notes this to be an X/Open Systems Interfaces (XSI) extension), but this support is exorbitantly sparse (at time of writing there are *two* **charclass** (cf. `wctype`(3)) definitions in all of glibc — "`jspace;jhira;jkata;jkanji;jdigit`" in `ja_JP` and "`hangul;hanja`" in `ko_KR`), and all multi-byte results are silently discarded, so the usefulness of this is dubious at best. The GNU system (and the plurality of implementations) only supports the literal *class*es out-lined in the table above.

IEEE Std 1003.1-2008 (“POSIX.1”) forbids redudant source transliteration bytes and, hence, **[***c****]** and **[***c*******len***]** in the first set — this implementation, like AT&T UNIX, allows them (given *len* > **0**), canonicalising the final occurrence of any given byte.

IEEE Std 1003.1-2008 (“POSIX.1”), in the case of *f***-***t* with *f* > *t* allows the resulting range to either be empty or refused. This implementation’s refusal behaviour matches the GNU system.

**-t** is an extension, originating from the GNU system.

**** at the end of a set is unspecified by POSIX — this implementation treats it as a literal ‘****’ for compatibility with 4.4BSD.

IEEE Std 1003.1-2008 (“POSIX.1”) doesn’t specify what happens for *****O*[*O*[*O*]] escapes larger than a byte (**\377**). The GNU system allows them with a warning, and parses them as **\0***OO* followed by *O*. 4.3BSD–Reno discards them. AT&T UNIXes allow them, masking down to the byte. This implementation forbids them.

## HISTORY

### Research UNIX

Appears in Version 4 AT&T UNIX as `tr`(I):

> tr – transliterate

with a **SYNOPSIS** of

> **tr** [ **–cds** ] [ string1 [ string2 ] ]

packed full of juicy semantics, most of which as present-day, except:

- **-c** is as present-day (except the generated set starts at **1**, not **0**).
- **[***c*******len***]** is as present-day, but in **[***c******] (including for *len* = *0*) *len* "is taken to be huge" (**1000**); the recommendation is still for padding *string2*.
- **[***f***-***t***]**, not *f***-***t*, generates characters in the [*f*, *t*] range (but *f* > *t* is refused).
- The default set length behaviour is the same as present-day **-t**, but documented from the opposite perspective:

  > If *string2* is short, it is padded with corresponding characters from *string1.*

- A single **** at the end of either string is equivalent to *****0* (i.e. is discarded);
- conversely, **** escapes are ANDed with **255**.
- Both *string*s are initialised to the empty string, and only updated if a string is passed; altogether, this means that **tr** [**-cs**] [*string1*] is allowed, but equivalent to **cat**,

but it wouldn’t be a C program if it didn’t have NUL-related issues:

- all NULs in the input are always removed,
- *****O*[*O*[*O*]] escapes that resolve to **0** (NUL) immediately terminate the containing *string*,

(these are noted in the **BUGS** section simply as "Won’t handle ascii NUL.").

Version 5 AT&T UNIX renders the **BUGS** more reasonably as

> Won’t handle ascii NUL in *string1* or *string2;* always deletes NUL from input.

Version 6 AT&T UNIX removes the description of what happens if the sets are different lengths entirely.

Version 7 AT&T UNIX replaces **[***c****]**, **[***c*******len***]** with the present-day ("BSD") final-*to*-byte extension and **[***f***-***t***]** with *f*-*t* that is just not recognised as a generator if *f* > *t*. The description also becomes

tr – translate characters

and ascii – ASCII.

### The BSD

Naturally, 3BSD inherits Version 7 AT&T UNIX **tr**.

4BSD adds expand(1) to **SEE ALSO**.

4.3BSD early-exits **1** if its writes to the standard output stream fail.

4.3BSD–Reno sees a complete rewrite, with an opinion:

```
/*
 * the original tr was amazingly tolerant of the command line.
 * Neither -c or -s have any effect unless there are two strings.
 * Extra arguments are silently ignored.  Bag this noise, they
 * should all be errors.
 */
```

preceding explicit conditional **cat** emulation and a fix: the "universe of characters" includes NUL, in both the input file and the sets (as **\0**, to which an unadorned backslash (one at the end) is now equivalent). This implementation is largely compatible, except it explicitly treats the *to* set with **-c** as only having the final character (and, per normal, extending it to the end), calling it "As reasonable as anything else.". A fringe problem though it may be, **printf** '\0\1aq' | **tr -c** a-c ABC produces "CCaC" instead of the expected "ABaC" — this is undocumented, and "Should really be an error.". Larger-than-**255** *O*[*O*[*O*]] escapes simply overflow the translation table indices (or values), and are, essentially, eaten (or truncated).

4.4BSD re-writes **tr** again with a strict reading of IEEE Std 1003.2-1992 ("POSIX.2"), fixing the **-c**-final-character-only behaviour, but allowing **[:***class***:]** and **[=***e***=]** in either argument in all cases. Of course, for **[=***e***=]**:

```
/*
 * English doesn't have any equivalence classes, so for now
 * we just syntax check and grab the character.
 */
```

**[***c****]**, **[***c*******len***]** are forbidden in the first set. **** at the end of a set is a literal ****. The sets are stored as *int*s, so larger-than-**255** *O*[*O*[*O*]] escapes overflow the indices even further, and are truncated later.

### System V

AT&T System III UNIX inherits Version 6 AT&T UNIX **tr**, but writes the "Bad string" error to the standard error stream instead.

AT&T System V Release 1 UNIX exits **1** in that case.

AT&T System V Release 4 UNIX includes 4.2BSD **tr** in /usr/ucb.

### Standards

X/Open Portability Guide Issue 2 ("XPG2") includes Version 6 AT&T UNIX tr(I) verbatim.

X/Open Portability Guide Issue 3 ("XPG3") adds **[[:***class***:]]** (as present-day **[:***class***:]**), **[[=***e***=]]** (as present-day **[=***e***=]**), and **[[.***cs***.]]** ("collating symbol" — "multi-character collating elements must be represented as collating symbols to distinguish them from single character collating elements"), matching notationally to EREs (regex(7)). They're all marked UN, for "Possibly *unsupportable* feature". Not surprisingly, nothing supported it. It also defines the order for all expressions in **[]**s to match the collation sequence (and the "universe of characters" for **-c**). This, on the other hand, is marked IN, for "Internationalised functionality", defined as optional.

IEEE Std 1003.2-1992 ("POSIX.2") specifies mostly present-day semantics (**-sd**, *O*[*O*[*O*]], all \-control-character escapes (since specifying them by value is non-portable), **[=***e***=]**, unspecified ordering of **[:***class***:]** and **[=***e***=]** except across **[:lower:]** and **[:upper:]** in the same relative positions, proper

NUL handling (since always removing NULs was considered less "correct functionality" than breaking compatibility: **tr -d '\000'** is recommended instead to restore the old behaviour)); and some differing ones: **-c** is defined in terms of characters, **[***c******]**, **[***c********len***]**, the sets, and ***f*****-***t*, in terms of "characters or collating symbols", with the latter being output in ascending collation sequence order. The double-bracket X/Open Portability Guide Issue 3 ("XPG3") syntax was reduced to single brackets, because, and this is a separate **APPLICATION USAGE** paragraph, **tr** sets aren't, even remotely, regular expressions.

Version 3 of the Single UNIX Specification ("SUSv3") backpedals, returning to present-day **-c** (renaming the former **-c** flag to **-C**, as it remains today), and lays forth the obvious issue very plainly:

> 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 IEEE Std 1003.1-2001. This meant that historical practice of being able to specify **tr -d** \200-\377 (which would delete all bytes with the top bit set) would have no effect because, in the C locale, bytes with the values octal 200 to octal 377 are not characters.

It also allows arbitrary locale-dependent **charclass**es in **[:***class***:]** and explicitly doesn't specify what happens if either set is empty.

IEEE Std 1003.1-2008 ("POSIX.1") reconciles the existing trailing-\ behaviours by explicitly making it unspecified, and, when translating, mirrors **sed**'s **y** operator by declaring as unspecified the behaviour for redundant characters in *from* (despite all implementations agreeing the final occurrence is canonical). The paragraph quoted in full above is also first fixed to **tr -cd** \000-\177, then removed since in IEEE Std 1003.1-2008 ("POSIX.1") all bytes are valid characters in the **C** (**POSIX**) locale.