LINFO

The tr Command

The *tr* command is used to *translate* specified characters into other characters or to delete them.

In contrast to many command line programs, tr does not accept file names as *arguments* (i.e., input data). Instead, it only accepts inputs via *standard input*, (i.e., from the keyboard) or from the output of other programs via redirection.

The general syntax of tr is

```
tr [options] set1 [set2]
```

The items in the square brackets are optional. tr requires at least one argument and accepts a maximum of two. The first, designated *set1*, lists the characters in the text to be replaced or removed. The second, *set2*, lists the characters that are to be substituted for the characters listed in the first argument.

When used without any options, tr will replace the characters provided in *set1* with those provided in *set1*. Thus, for example, every instance of the letter *a* in text typed in at the keyboard can be replaced with the letter *b* by entering the following command, pressing the ENTER key to start a new line, typing the text, and then pressing the ENTER key again:

```
tr a b
```

This substitution can repeated for additional text by typing it in and then pressing the ENTER key again. It can be terminated by simultaneously pressing the CONTROL and *c* keys.

Although tr cannot accept the names of files as arguments, it can nevertheless be used to modify copies of their contents. All that is necessary is to use the *input redirection operator*, which is represented by a leftward pointing angular bracket, before the name of the file whose text is to be modified. For example, the following would redirect a copy of a file named *file1* to tr, which would display its contents on the monitor screen with every instance of an *a* replaced with a *b*:

```
tr a b < file1
```

It is typically more convenient to have the modified output written to a file rather than being displayed on the screen, and this can be easily accomplished by using the *output redirection operator*, which is represented by a rightward pointing angular bracket. Thus, for example, the following would replace the letter *c* with the letter *d* in the text from file1 and write it to a file named *file2*:

```
tr c d < file1 > file2
```

If file2 does not already exist, it will be created automatically by the output redirection operator. If it already exists, it will be overwritten.

Care should be taken to avoid attempting to use the output redirection operator to write to the same file from which the text is being read, as this will erase all of the text in that file, including that outputted by tr. Thus, for example, the following should *not* be used:

```
tr c d < file1 > file1
```

What can be done, however, is to use the *append redirection operator*, which is represented by two successive rightward pointing angular brackets, to add the modified text to the end of the existing text in the input file. Thus, the above example could be modified to append the revised text to file1 as follows:

```
tr c d < file1 >> file1
```

An alternative way to obtain a copy of the content of a file is to use a command such as *cat* to read it and then send its output to tr via a *pipe*, which is represented by the vertical bar character. For example, the following sends a copy of the text in a file named *file3* to tr for translation before writing the result to another file named *file4*:

```
cat file3 | tr c d > file4
```

tr can do much more than just translate a single letter. It can also translate any number of specified characters into other characters. In such case, each of the two sets needs to be enclosed in square brackets, and these brackets, in turn, need to be enclosed in quotation marks. Either single of double quotation marks can be used.

One way of performing such translation of multiple characters is by listing the characters that are to be changed in the first argument and and listing their respective replacements in order in the second argument. For example, the following would replace all instances of the letters *e*, *f* and *g* in a file named *file5* with the letters *x*, *y* and *z*, respectively, and write the output to another file named *file6*:

```
cat file5 | tr '[efg]' '[xyz]' > file6
```

tr can also replace characters in a specified range by their counterparts in another specified range, which can be more convenient when numerous characters are involved. A range is indicated by inserting a hyphen between the first and last characters, placing all of this in square brackets and then enclosing the brackets in quotation marks. Thus, the above example could be rewritten as follows:

```
cat file5 | tr '[e-g]' '[x-z]' > file6
```

Likewise, the following would replace every upper case letter in a file named *file7* by its lower case counterpart and write the result to a file called *file8*:

```
cat file7 | tr '[A-Z]' '[a-z]' > file8
```

tr can also be used with *control characters*. Also referred to as *non-printing characters*, these are bit sequences that represent basic formatting instructions, etc., and each begins with a backslash. They are *b* for backspace, *f* for form feed, *n* for new line, *r* for return, *t* for horizontal tab, *v* for vertical tab, *a* for buzzer sound, \\ for backslash and *NNN* for octal value NNN (consisting of one to three digits).

tr also has a number of standardized arguments that are often more convenient to use instead of making lists or ranges of characters to be translated. Each consists of a word or abbreviation surrounded by colons and then enclosed in a set of square brackets. For example, all of the letters of the alphabet are represented by [:alpha:], all numerals are represented by [:digit:], all alphanumeric characters are represented by [:alnum:], all horizontal white spaces are represented by [:blank:], all horizontal or vertical white spaces are represented by [:space:], all printable characters exclusive of spaces are represented by [:graph:], all control characters are represented by [:cntrl:], all lower case letters are represented by [:lower:], all upper case letters are represented by [:upper:], all punctuation characters are represented by [:punct:], all printable characters including spaces are represented by [:print:], and all hexadecimal digits are represented by [:xdigit:].

Thus, for example, the previous example could be rewritten as

```
cat file7 | tr [:upper:] [:lower:] > file8
```

Probably the most useful of tr's several options is -s, which replaces each instance of a sequence of repeated characters listed in set1 with a single instance of the character specified in set2. This *squeeze* option is commonly used to replace each sequence of multiple blank spaces in text with a single blank space. For example, the following will remove all instances of successive blank spaces from a copy of the text in a file named *file9* and write its output to a new file named *file10*:

```
tr -s ' ' ' ' < file9 > file10
```

The *-d* option is used to delete every instance of the *string* (i.e., sequence of characters) specified in set1. Thus, for example, the following would remove every instance of the word *soft* from a copy of the text in a file named *file11* and write the modified text to a file named *file12*:

```
cat file11 | tr -d 'soft' > file12
```

The quotation marks are necessary for tr to treat the argument as a string. If they are not used, everything in the argument is instead treated as individual characters. Thus, if the above example were rewritten without the quotation marks, it would remove every instance of the letters *s*, *o*, *f* and *t*. Interestingly, the quotation marks cannot be used to treat arguments as strings when not using the -d option.

Among the few remaining options is *-c*, which causes tr to work on the *complement* of the specified characters, that is, on the characters that are *not* in the given set.

tr contains much of most basic functionality of the command line program *sed*, which is used to perform basic editing on *streams* of text supplied by a pipe. However, it often advantageous to use tr instead of sed because the former is simpler and requires less typing and because it is easier to incorporate into scripts.

Created October 28, 2006.