Using tr and dos2unix

THE TRANSLATOR

The *tr* tool is a real wizard. This simple command lets you replace strings in text files. Whether you are replacing letters or just removing whitespace, you will be amazed at *tr*'s versatility. **BY HEIKE JURZIK**

The *tr* command replaces characters in text files. The command reads the standard input and sends the results to standard output. But of course, you can use the familiar operators to redirect both streams. In fact, *tr* really shines in combination with other shell commands.

Simple Replacements

The *tr* command expects two strings as arguments and replaces all the occurrences of the first argument in a text with the second argument. This may sound complicated, but let's look at a simple example. The following command replaces the "e"s in "Petronella" with "a"s:

```
$ echo Petronella | tr 'e' 'a'
Patronalla
```

Of course you can specify the strings you want to replace within a file. *tr '1' '2' < test.txt* will replace any occurrences of "1" in the file *test.txt* with "2" and send the result to standard output. Note that the characters within the arguments you pass to the utility are handled as though they were in separate fields. Each character in the first argument is replaced by its counterpart in the second argument. For example, *tr 'abc' 'xyz'* will replace "a" with "x", "b" with "y", and "c" with "z.. If the second string is shorter than the first string, *tr* fills the gap with the last character in the second string.

For example, you can enter *tr 'abc' 'z' < test.txt* without fazing *tr*. The utility simply replaces all occurrences of "a," "b," and "c" with "z." However, *tr* fails if you try to replace, "ä" with "ae." The fields are not the same length, and "ä" is just replaced with "a." You might prefer to use *sed* for these more complex replacement tasks

Case Sensitive

tr can be extremely useful if you need to swap lower- and uppercase letters. The best approach is two define both arguments as arrays of lower and capital letters, for example:

```
tr 'a-z' 'A-Z' < test.txt
```

If you prefer, you can simply reference the cases as follows:

```
tr [:lower:] [:upper:]
< test.txt
```

Everything but...

The *tr* command has a few parameters that give you more granular control. For example, you can use the *-d* option to delete things, as in,

```
tr -d '0-9' < test.txt
```

which sends all the numbers in a text to the happy hunting grounds. A combination of this option and *-c* gives you an even neater way of removing superfluous content. Let's assume you want to remove everything apart from space characters, uppercase characters, and lowercase letters; you can use *-c* to tell *tr* what not to delete:

```
tr -c -d 'A-Z a-z' < test.txt
```

In combination with *-s*, *tr* allows you to reduce the volume of a file – a useful ability if you have, say, a logfile full of whitespace. The *-s* option expects either one or two arguments. For example, *tr -s ' ' < test.txt* removes all the whitespace from a file. But if you simply need to remove double spaces, or tabs, and insert a simple blank instead, you can supply *tr -s* with two arguments:

```
tr -s [:blank:] ' ' < test.txt
```

In this case, *tr* will replace contiguous blanks or tabs with simple blanks.

Migrating between Worlds

If you often have to exchange files between Windows and Linux systems, you will note that strange characters can occur at ends of lines. For example, if you try to open an ASCII created in Windows with the Vim editor, you will notice a weird looking bunch of *^M* characters. The reason for this is quite simple: the two systems use different symbols for the end-of-line character. Whereas Windows uses *\r\n* to denote a line break, Linux leaves off the *\r* and simply keeps the *\n*.

tr can help you exchange ASCII files between these two systems. Calling

```
tr -d '\r' < wintext > linuxtext
```

will remove the extra *\r* at the end of each line and convert the text file. The *-d* flag tells *tr* to remove the unwanted character. The < operator then tells the command to parse *wintext*, and > sends the "clean" results to an output file called *linuxtext*.

Alternative Converters

The *dos2unix* and *unix2dos* tools are also useful if you need to convert back and forth between Linux and Windows. To convert a Windows text file to the right format for Linux, simply enter:

```
$ dos2unix -n wintext ↵
linuxtext
dos2unix: converting file ↵
wintext to file linuxtext ↵
in UNIX format ...
```

This command uses the *-n* option, which allows you to specify both an input file and a new output file. The manpage has details on other useful tips and tricks. The *-k* flag keeps the original timestamp, and *-o* writes the changes directly to the original file. *unix2dos* does the same thing, but in the other direction:

```
unix2dos -n linuxtext wintext
```

On some systems, these utilities are symbolic links to the *fromdos* and *todos* programs, respectively. These programs have a slightly different syntax and different parameters. You should use the *-b* option to create a backup of the original file, even if you are not creating a new file. And you might like to note the *-a* flag, which removes all carriage returns when stipulated with *fromdos* (and not only those preceded by a line feed). If you specify *-a* with the *todos* command, it will convert any line feeds into CR-LF pairs. The default behavior in this case would be to convert only those line feeds that are not preceded by a carriage return.

The two examples of *dos2unix* and *unix2dos* shown previously could thus look like the following:

```
cat wintext | fromdos -a ↵
> linuxtext
cat linuxtext | todos -a ↵
> wintext
```

Good Combinations

tr really shines when you use it in combination with other shell commands. For example, imagine you discover a large number of files with blanks in their filenames on your disk, and you want to replace the blanks with underscores. A *for* loop, the *mv* command, and *tr* will help you change the underscores to blanks:

```
$ for i in *; do mv -v "$i" ↵
`echo $i | tr ' ' '_'`; done
'file with blank'  -> ↵
'file_with_blank'
'file with blank 2' -> ↵
'file_with_blank_2'
```

If we translate the preceding command into plain English, the command would be: for all the files in the current directory do the following: move the files visibly for the user to files whose names are the results of the *tr* replacement operations. ■

GLOSSARY

Line break: The syntax for line breaks on computers was based on the way typewriters work. There is a control character for the line feed, and another for the carriage return. Different operating systems have different approaches to handling line breaks. Whereas Linux uses a simple line feed (\n = "new line"), DOS/Windows adds a carriage return (\r = "return").

THE AUTHOR

Heike Jurzik studied German, Computer Science and English at the University of Cologne, Germany. She discovered Linux in 1996 and has been fascinated with the scope of the Linux command line ever since. In her leisure time you might find Heike hanging out at Irish folk sessions or visiting Ireland.