Input characters from `stdin` are mapped from `set1` to `set2` and the output is written to `stdout` (standard output). `set1` and `set2` are character classes or a set of characters. If the length of sets is unequal, `set2` is extended to the length of `set1` by repeating the last character, or else, if the length of `set2` is greater than that of `set1`, all the characters exceeding the length of `set1` are ignored from `set2`.

How to do it...

To perform translation of characters in the input from uppercase to lowercase, use the following command:

```
$ echo "HELLO WHO IS THIS" | tr 'A-Z' 'a-z'
```

`'A-Z'` and `'a-z'` are the sets. We can specify custom sets as needed by appending characters or character classes.

`'ABD-}'`, `'aA.,'`, `'a-ce-x'`, `'a-c0-9'`, and so on are valid sets. We can define sets easily. Instead of writing continuous character sequences, we can use the `'startchar-endchar'` format. It can also be combined with any other characters or character classes. If `startchar-endchar` is not a valid continuous character sequence, they are then taken as a set of three characters (for example, `startchar`, `-`, and `endchar`). You can also use special characters such as `'\t'`, `'\n'`, or any ASCII characters.

How it works...

By using `tr` with the concept of sets, we can map characters from one set to another set easily. Let's go through an example on how to use `tr` for encrypting and decrypting numeric characters:

```
$ echo 12345 | tr '0-9' '9876543210'
87654 #Encrypted


$ echo 87654 | tr '9876543210' '0-9'
12345 #Decrypted
```

Let's look at another interesting example.

ROT13 is a well-known encryption algorithm. In the ROT13 scheme, the same function is used to encrypt and decrypt text. The ROT13 scheme performs alphabetic rotation of characters for 13 characters. Let's perform ROT13 using `tr` as follows:

```
$ echo "tr came, tr saw, tr conquered." | tr 'a-zA-Z' 'n-za-mN-ZA-M'
```

The output will be:

```
ge pnzr, ge fnj, ge pbadhrerq.
```

By sending the encrypted text again to the same ROT13 function, we get:

```
$ echo ge pnzr, ge fnj, ge pbadhrerq. | tr 'a-zA-Z' 'n-za-mN-ZA-M'
```

The output will be:

```
tr came, tr saw, tr conquered.
```

`tr` can be used to convert tab characters into space as follows:

```
$ tr '\t' ' ' < file.txt
```

There's more...

We saw some basic translations using the `tr` command. Let's see what else can `tr` help us achieve.

Deleting characters using tr

`tr` has an option `-d` to delete a set of characters that appear on `stdin` by using the specified set of characters to be deleted as follows:

```
$ cat file.txt | tr -d  '[set1]'
#Only set1 is used, not set2
```

For example:

```
$ echo "Hello 123 world 456" | tr -d '0-9'
Hello world
# Removes the numbers from stdin and print
```

Complementing character set

We can use a set to complement `set1` by using the `-c` flag. `set2` is optional in the following command:

```
tr -c [set1] [set2]
```

The complement of `set1` means that it is the set having all the characters except characters in `set1`.

The best usage example is to delete all the characters from the input text except the ones specified in the complement set. For example:

```
$ echo hello 1 char 2 next 4 | tr -d -c '0-9 \n'
 1  2  4
```

Here, the complement set is the set containing all numerals, space characters, and newline characters. All other characters are removed since `-d` is used with `tr`.