To provide a command execution sequence as shown, `xargs` has an option `-I`. By using `-I`, we can specify a replacement string that will be replaced while `xargs` expands. When `-I` is used with `xargs`, it will execute as one command execution per argument.

Let's do it as follows:

```
$ cat args.txt | xargs -I {} ./cecho.sh -p {} -l
-p arg1 -l #
-p arg2 -l #
-p arg3 -l #
```

`-I {}` specifies the replacement string. For each of the arguments supplied for the command, the `{}` string will be replaced with arguments read through `stdin`.

When used with `-I`, the command is executed in a loop. When there are three arguments the command is executed three times along with the command `{}`. Each time `{}` is replaced with arguments one by one.

Using xargs with find

`xargs` and `find` are best friends. They can be combined to perform tasks easily. Usually, people combine them in the wrong way. For example:

```
$ find . -type f -name "*.txt"  -print | xargs rm -f
```

This is dangerous. It may sometimes cause removal of unnecessary files. Here, we cannot predict the delimiting character (whether it is `'\n'` or `' '`) for the output of the `find` command. Many of the filenames may contain a space character (`' '`) and hence, `xargs` may misinterpret it as a delimiter (for example, `"hell text.txt"` is misinterpreted by `xargs` as `"hell"` and `"text.txt"`).

Hence, we must use `-print0` along with `find` to produce an output with a delimited character null (`'\0'`) whenever we use the `find` output as the `xargs` input.

Let's use `find` to match and list of all the `.txt` files and remove them using `xargs`:

```
$ find . -type f -name "*.txt" -print0 | xargs -0 rm -f
```

This removes all `.txt` files. `xargs -0` interprets that the delimiting character is `\0`.

Counting the number of lines of C code in a source code directory

This is a task most programmers do, that is, counting all C program files for **Lines of Code** (**LOC**). The code for this task is as follows:

```
$ find source_code_dir_path -type f -name "*.c" -print0 | xargs -0 wc -l
```

If you want more statistics about your source code, there is a utility called **SLOCCount**, which is very useful. Modern GNU/Linux distributions usually have packages or you can get it from `http://www.dwheeler.com/sloccount/`.

While and subshell trick with stdin

`xargs` is restricted to providing arguments in limited ways to supply arguments. Also, `xargs` cannot supply arguments to multiple sets of commands. For executing commands with collected arguments from the standard input, we have a very flexible method. A subshell with a `while` loop can be used to read arguments and execute commands in a trickier way as follows:

```
$ cat files.txt  | ( while read arg; do cat $arg; done )
# Equivalent to cat files.txt | xargs -I {} cat {}
```

Here, by replacing `cat $arg` with any number of commands using a `while` loop, we can perform many command actions with the same arguments. We can also pass the output to other commands without using pipes. Subshell `( )` tricks can be used in a variety of problematic environments. When enclosed within subshell operators, it acts as a single unit with multiple commands inside, like so:

```
$ cmd0 | ( cmd1;cmd2;cmd3) | cmd4
```

If `cmd1` is `cd /`, within the subshell, the path of the working directory changes. However, this change resides inside the subshell only. `cmd4` will not see the directory change.

Translating with tr

`tr` is a small and beautiful command in the Unix command-warrior toolkit. It is one of the important commands frequently used to craft beautiful one-liner commands. It can be used to perform substitution of characters, deletion of the characters, and squeezing of repeated characters from the standard input. It is often called **translate**, since it can translate a set of characters to another set. In this recipe we will see how to use `tr` to perform basic translation between sets.

Getting ready

`tr` accepts input only through `stdin` (standard input) and cannot accept input through command-line arguments. It has the following invocation format:

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