

Process Substitution

In this lesson, you will learn about process substitution, a handy way to save time and make scripts more concise and elegant. What you'll learn will also allow you to capture the output of a command into a variable so that you can use it in other contexts or access it later.

WE'LL COVER THE FOLLOWING ^

- How Important is this Lesson?
- Simple Process Substitution
- The `<()` operator
- The `>()` operator
- What You Learned
- What Next?
- Exercises

How Important is this Lesson?

I spent years reading and writing bash before I understood this concept, so this lesson can be skipped. However, since I learned about **process substitution**, I use it on the command line almost every day, so I recommend you learn it at some point.

Simple Process Substitution

Type this in to set files up for this lesson:

```
mkdir a
mkdir b
touch a/1 a/2 # Creates files 1 and 2 in folder a
touch b/2 b/3 # Creates files 2 and 3 in folder b
ls a
ls b
```



Type the above code into the terminal in this lesson.

You've created two folders with slightly different contents.

Now let's say that you want to diff the output of `ls a` and `ls b` (a trivial but usefully simple example here). How would you do it?

Note: if you are not familiar with the `diff` command, you can find an introduction to it [here](#).

You might do it like this:

```
ls a > aout      # List the files in the folder 'a' and redirect output to the 'aout' file
ls b > bout      # List the files in the folder 'b' and redirect output to the 'bout' file
diff aout bout   # Diff the two outputs
rm aout bout     # Clean up the temporary output files
```

Type the above code into the terminal in this lesson.

That works, and there's nothing wrong with it, but typing all that out and cleaning up the files is a bit cumbersome. There's a much neater way that exposes a very useful technique.

Type this in:

```
diff <(ls a) <(ls b)
```

Type the above code into the terminal in this lesson.

That's neater, isn't it?

So what's going on?

The `<()` operator

The `<()` operator is conceptually similar to the `$()` we saw earlier. In the same way that `$()` substitutes the *output* of the process contained within it into the command, eg:

```
echo $(lsa)
```

Type the above code into the terminal in this lesson.

the `<()` operator substitutes a *file containing the output* of the process contained within it. You might need to stop and think about this for a second.

That means that this line:

```
diff <(ls a) <(ls b)
```

effectively becomes the command to diff two files, equivalent to the files `aout` and `bout` in the lines you typed in above.

So *wherever you would normally put a filename*, you can use the `<()` operator to save some time by dropping these in rather than creating files.

The `>()` operator

Can you guess what this does? It's similar to the `<()` but for me it was a lot trickier to grasp, and much more rarely seen (so feel free to skip).

See if you can work out from this line what it does:

```
tar cvf >(cat > out.tar) /tmp
```



Type the above code into the terminal in this lesson.

As with the `<()` operator, this replaces a file in a command. This time, rather than sending the output to the file, it takes *input* from the command that would normally go to that file reference, and feeds that input to the command in the operator.

Let's take a step back and think about that, because it can be hard to follow.

Normally you'd write something like this:

```
rm -f out.tar  
tar cvf out.tar /tmp
```



Type the above code into the terminal in this lesson.

The command is read into bash, expanded out, and the tar command accepts two arguments: a file and a folder (`out.tar` and the `/tmp`). It tars up the

contents of the `/tmp` folder and places it in the `out.tar` file.

The difference in the previous command is that the contents that would normally be inputted into the file are instead fed into the command `cat > out.tar`.

Obviously, in this case that command is pointless - in both cases you end up with a file called `out.tar` that is a tar file.

Let's say, however, that you wanted to use a different compression scheme for your tar file. You could type this:

```
tar cvf >(gzip > out.tar.gz) /tmp
```



Type the above code into the terminal in this lesson.

which would gzip the tarfile and place it into the `out.tar.gz` file.

It can reasonably be pointed out that most versions of tar offer a gzip flag (`-z`). that does this for you. However, some versions don't (especially on minimal Linux distributions like busybox), so this can be a neat way of getting round that.

I have never had a need to use this mechanism in real life, but I've written things like this before, which are less neat (but good enough):

```
tar cvf out.tar /tmp
gzip out.tar
```



Type the above code into the terminal in this lesson.

What You Learned

- What the `<()` operator is
- What the `>()` operator is
- How to use these operators
- How `<()` differs from the `$()` operator
- How `>()` works

What Next?

Next you will cover **subshells**, and **grouping** commands more generally.

Exercises

- 1) Look for examples of where these operators are used on the web, and figure out what they're doing.
- 2) Look through your bash history on other machines where you have been using bash (by typing `history`) and see where you could have used these operators.
- 3) Construct a command that uses `$()`, `<()`, and `>()`.