bash, howto, processsubstitution, process, substitution, parallelization

Bash process substitution

- https://medium.com/@joewalnes/handy-bash-feature-process-substitution-8eb6dce68133

Handy Bash feature: Process Substitution

Pretty much everything useful I ever learned about UNIX shells was not from reading books or manuals, but by peering over someone's shoulder and seeing something I'd never seen before. "Ooooh, what was that you just did?"

It was a few years ago I was working with my friend Jordan Samuels, when we wanted to compare two versions of a file at a particular URL. I knew what to do...

```
curl http://somesite/file1 > file1
curl http://somesite/file2 > file2
diff file1 file2
```

Simple right? Download *file1*, download *file2*, and *diff* them. 3 steps.

But before I could type it, Jordan grabbed the keyboard and typed some crazy voodoo I'd never seen before...

```
diff <(curl http://somesite/file1) <(curl http://somesite/file2)
```

Ooooh, what was that weird syntax? And why did it appear to run twice as fast as what I was expecting?

Process substitution!

Process substitution gives you similar capabilities to *piping*. Except *piping* only allows you to *pipe* the *output* from a **single** command into another. In the **diff** scenario, we need to *pipe* the output from **multiple** commands into another. And that's what *process substitution* allows us to do.

The syntax for using process substitution is this:

```
some-command <(another-command)
```

Where **some-command** accepts a *filename* (or multiple *filenames*) as **arguments**, and **another-command** writes output to **stdout**.

Wait a minute... how does that work? There are no filenames anywhere? Well, behind the scenes, when Bash sees the process substitution *<(...)*, it'll create a *temporary file descriptor* which it uses as the *filename* and *pipe* **output** from the other **process** into it.

When to use process substitution

If a regular old *pipe* will do, just do that. But there are a few scenarios when pipes won't cut it...

- If you need to feed **multiple outputs** into a **single program** (like the diff example above)
- If you need to feed **output from a program** into a program that **expects input files, but cannot read from stdin**
- If you find yourself **using temporary files**

Parallelization!

Apart from simplicity, another advantage of using process substitution is Bash will **automatically parallelize** your tasks. Returning to our first example...

```
diff <(curl http://somesite/file1) <(curl http://somesite/file2)
```

... Bash will **run both** those *curl* commands **in parallel**. Sweet huh?

Ok, that's it. Have a nice day.

Diving deeper

If you want to get into the nitty gritty details about how this stuff works, here's some more reading.

- http://tldp.org/LDP/abs/html/process-sub.html
- http://en.wikipedia.org/wiki/Process_substitution
- http://en.wikipedia.org/wiki/Named_pipe

~~DISCUSSION~~