How to give a comma-separated list as arguments to the next command

Ask Question

I have a script `s1` that outputs a list of numbers separated with ',' e.g. `1,2,3,4`. Now I want to give these numbers to script `s2` as arguments, so that s2 will be run on each of them and output its result in a separate line. For example, if s2 multiplies numbers by two, this would be the result I'm looking for:

```
2
4
6
8
```

What I'm doing right now is:

```
s1 | xargs -d "," | xargs -n1 s2
```

But I feel like I'm doing it in such a foolish way! So my question is:

What is the proper way of doing it?

My problem with my solution is that it's calling xargs twice and iterating over the input twice which is not reasonable to my eyes of course by means of performance! The answer `xargs -d "," -n1` seems nice, but I'm not sure if it's only iterating once. If it does, please verify that in your answer, and I'll accept it. By the way, I'd rather not use Perl since it still is iterating twice and also Perl may not exist on many systems.

pipe xargs arguments

1	If it's working why call it foolish. If execution time is important then that's another matter else leave here – George Udosen Dec 18 '17 at 15:11
2	Try this `s1 | xargs -d "," -n1 s2` – George Udosen Dec 18 '17 at 15:13
1	I suspect some of the issue is misunderstanding the impact of iterating. Iterating over the elements in something like an associative array is bad because of the expense of walking that data structure, but "iterating" in general is not inherently bad. Specifically, reading data line-by-line as it comes in on STDIN, isn't a huge performance problem. The performance issue here is more the cost of spawning a new process and setting up the pipeline. As long as you're not doing that frequently (as in a loop), worrying about the performance of xargs is probably an example of premature optimization. – dannysauer Dec 20 '17 at 2:31

3 Answers

This should equally work as well:

```
s1 | xargs -d "," -n1 s2
```

Test case:

```
printf 1,2,3,4 | xargs -d ',' -n1 echo
```

Result:

```
1
2
3
4
```

If `s1` outputs that list followed by a newline character, you'd want to remove it as otherwise the last call would be with `4\n` instead of `4` :

```
s1 | tr -d '\n' | xargs -d , -n1 s2
```

If `s2` can accept multiple arguments, you could do:

```
(IFS=,; ./s2 $(./s1))
```

which temporarily overrides IFS to be a comma, all in a subshell, so that `s2` sees the output of `s1` broken up by commas. The subshell is a short-hand way to change IFS without saving the previous value or resetting it.

A previous version of this answer was incorrect, probably due to a leftover IFS setting, corrupting the results. Thanks to ilkkachu for pointing out my mistake.

To manually loop over the outputs and provide them to individually to `s2` , here demonstrating the saving & resetting of IFS:

```
oIFS="$IFS"
IFS=,
for output in $(./s1); do ./s2 "$output"; done
IFS="$oIFS"
```

or run the IFS bits in a subshell as before:

```
(
IFS=,
for output in $(./s1); do ./s2 "$output"; done
)
```

edited Dec 19 '17 at 12:38

ilkkachu
55k 7 82 150

answered Dec 18 '17 at 17:34

Jeff Schaller
37.9k 10 53 123

1 Are you sure about that first one? `bash -c 'IFS=, printf "%s\n" $(echo 1,2,3)'` prints `1,2,3` on my system, i.e. there's no splitting. – ilkkachu Dec 18 '17 at 22:11

I'll re-test in a bit, but I suspect different behavior based on builtins vs external programs. – Jeff Schaller Dec 18 '17 at 22:29

same with `/usr/bin/printf` and `/bin/echo` – ilkkachu Dec 18 '17 at 22:31

1 @ilkkachu You are correct, word splitting occurs before IFS is re-assigned in this case `(IFS=,; printf "%s\n" $(echo 1,2,3))` , on the other hand, should work. – undercat Dec 18 '17 at 23:01

Try this:

```
s1 | perl -pe 's/,/\n/g' | xargs -n1 s2
```

(The space...) – Peter Mortensen Dec 18 '17 at 19:30

2 Why not simply `tr ',' '\n'` ? No need to invoke something as (relatively) heavy as Perl and regular expressions. – David Foerster Dec 18 '17 at 20:38