Examples of Pipes

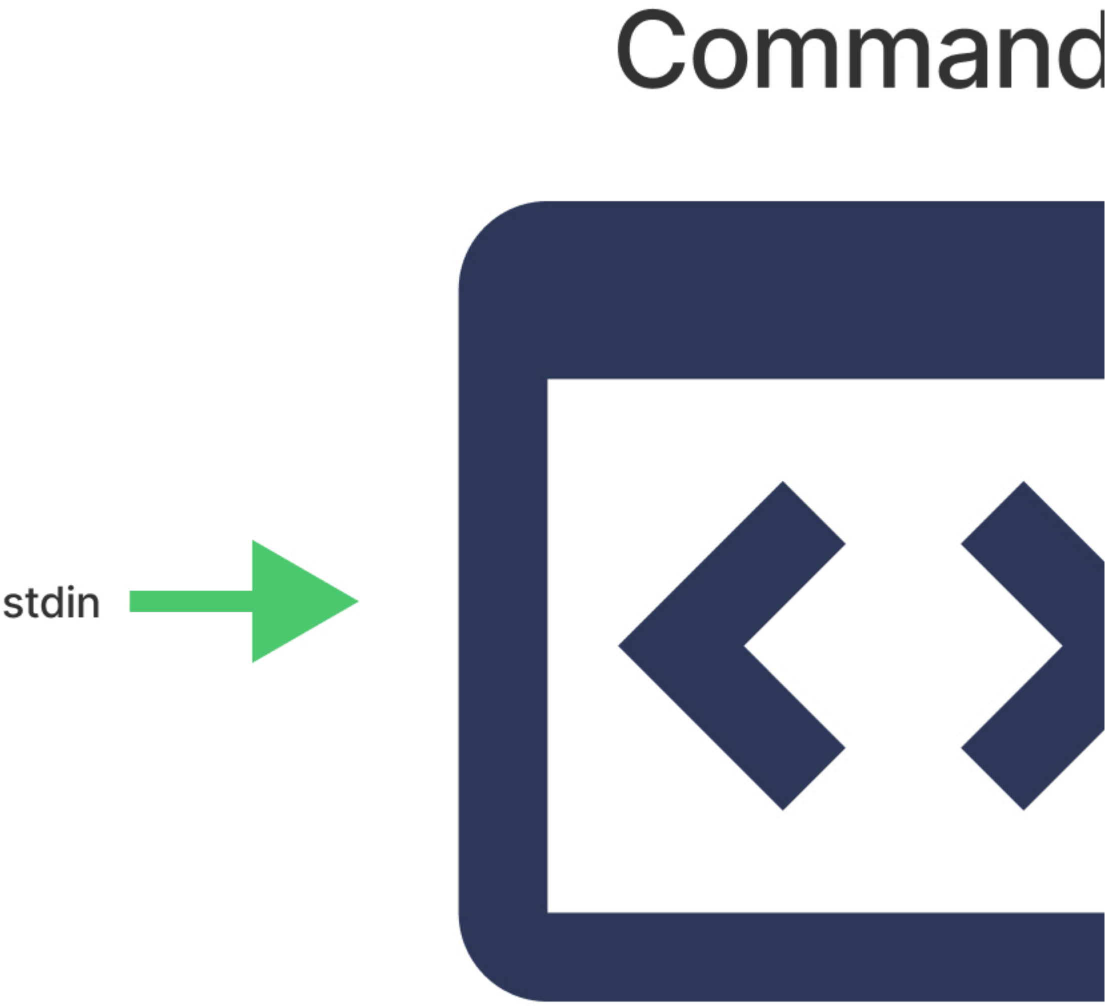
Learning Objectives

After completing this reading, you will be able to:

- Describe pipes
- Use pipes to combine commands when working with strings and text file contents
- Use pipes to extract information from URLs

What are pipes?

Put simply, pipes are commands in Linux which allow you to use the output of one command as the input of another.



Pipes | use the following format:

```
[command 1] | [command 2] | [command 3] ... | [command n]
```

There is no limit to the number of times you can chain pipes in a row!

In this lab, you'll take a closer look at how you can use pipes and filters to solve basic data processing problems.

Pipe examples

Combining commands

Let's start with a commonly used example. Recall the following commands:

- `sort` - sorts the lines of text in a file and displays the result
- `uniq` - prints a text file with any consecutive, repeated lines collapsed to a single line

With the help of the pipe operator, you can combine these commands to print all the unique lines in a file.

Suppose you have the file `pets.txt` with the following contents:

```
$ cat pets.txt
goldfish
dog
cat
parrot
dog
goldfish
goldfish
```

If you *only* use `sort` on `pets.txt`, you get:

```
$ sort pets.txt
cat
dog
dog
goldfish
goldfish
goldfish
parrot
```

The file is sorted, but there are duplicated lines of "dog" and "goldfish".

On the other hand, if you *only* use `uniq`, you get:

```
$ uniq pets.txt
goldfish
dog
cat
parrot
dog
goldfish
```

This time, you removed consecutive duplicates, but non-consecutive duplicates of "dog" and "goldfish" remain.

But by combining the two commands in the correct order - by first using `sort` then `uniq` - you get back:

```
$ sort pets.txt | uniq
cat
dog
goldfish
parrot
```

Since `sort` sorts all identical items consecutively, and `uniq` removes all consecutive duplicates, combining the commands prints only the unique lines from `pets.txt`!

Applying a command to strings and files

Some commands such as `tr` only accept *standard input* - normally text entered from your keyboard - but not strings or filenames.

- `tr` (translate) - replaces characters in input text

  ```
  tr [OPTIONS] [target characters] [replacement characters]
  ```

In cases like this, you can use piping to apply the command to strings and file contents.

With strings, you can use `echo` in combination with `tr` to replace all the vowels in a string with underscores `_`:

```
$ echo "Linux and shell scripting are awesome\!" | tr "aeiou" "_"
L_n_x _nd sh_ll scr_pt_ng _r_ _w_s_m_!
```

To perform the complement of the operation from the previous example - or to replace all the *consonants* (any letter that is not a vowel) with an underscore - you can use the `-c` option:

```
$ echo "Linux and shell scripting are awesome\!" | tr -c "aeiou" "_"
_i_u__a_____e______i__i___a_e_a_e_o_e_
```

With files, you can use `cat` in combination with `tr` to change all of the text in a file to uppercase as follows:

```
$ cat pets.txt | tr "[a-z]" "[A-Z]"
GOLDFISH
DOG
CAT
PARROT
DOG
GOLDFISH
GOLDFISH
```

The possibilities are endless! For example, you could add `uniq` to the above pipeline to only return unique lines in the file, like so:

```
$ sort pets.txt | uniq | tr "[a-z]" "[A-Z]"
CAT
DOG
GOLDFISH
PARROT
```

Extracting information from JSON Files:

Let's see how you can use this json file to get the current price of Bitcoin (BTC) in USD, by using grep command.

```
{
  "coin": {
    "id": "bitcoin",
    "icon": "https://static.coinstats.app/coins/Bitcoin6l39t.png",
    "name": "Bitcoin",
    "symbol": "BTC",
    "rank": 1,
    "price": 57907.78008618953,
    "priceBtc": 1,
    "volume": 48430621052.9856,
    "marketCap": 1093175428640.1146,
    "availableSupply": 18877868,
    "totalSupply": 21000000,
    "priceChange1h": -0.19,
    "priceChange1d": -0.4,
    "priceChange1w": -9.36,
    "websiteUrl": "http://www.bitcoin.org",
    "twitterUrl": "https://twitter.com/bitcoin",
    "exp": [
      "https://blockchair.com/bitcoin/",
      "https://btc.com/",
      "https://btc.tokenview.com/"
    ]
  }
}
```

Copy the above output in a file and name it as `Bitcoinprice.txt`.

The JSON field you want to grab here is `"price": [numbers].[numbers]"`. To get this, you can use the following `grep` command to extract it from the JSON text:

```
grep -oE "\"price\"\s*:\s*[0-9]*\.?[0-9]*"
```

Let's break down the details of this statement:

- `-o` tells `grep` to *only* return the matching portion
- `-E` tells `grep` to be able to use extended regex symbols such as `?`
- `\"price\"` matches the string `"price"`
- `\s*` matches any number (including 0) of whitespace (`\s`) characters
- `:` matches `:`
- `[0-9]*` matches any number of digits (from 0 to 9)
- `\.?` optionally matches a `.`

Use the cat command to get the output of the JSON file and pipe it with the grep command to get the required output.

```
cat Bitcoinprice.txt | grep -oE "\"price\"\s*:\s*[0-9]*\.?[0-9]*"
```

You can also extract information directly from URLs and retreive any specific data using such grep commands.

▶ Click here to see the process of extracting information directly from URLs and retreiving specific data:

Summary

In this reading, you learned that:

- Pipes are commands in Linux which allow you to use the output of one command as the input of another
- You can combine commands such as `sort` and `uniq` to organize strings and text file contents
- You can pipe the output of a `curl` command to `grep` to extract components of URL data

Authors

Jeff Grossman
Sam Prokopchuk