Reading: A Brief Introduction to Shell Variables

Learning Objectives

After completing this reading, you will be able to:

- Describe shell variables
- Create shell variables

What is a shell variable?

Shell variables offer a powerful way to store and later access or modify information such as numbers, character strings, and other data structures by name. Let's look at some basic examples to get the idea.

Consider the following example.

```
$ firstname=Jeff
$ echo $firstname
Jeff
```

The first line assigns the value `Jeff` to a new variable called `firstname`. The next line accesses and displays the value of the variable, using the `echo` command along with the special character `$` in front of the variable name to extract its value, which is the string `Jeff`.

Thus, we have created a new shell variable called `firstname` for which the value is `Jeff`.

This is the most basic way to create a shell variable and assign it to a value all in one step.

Reading user input into a shell variable at the command line

Here's another way to create a shell variable, using the `read` command. After entering

```
$ read lastname
```

on the command line, the shell waits for you to enter some text:

```
$ read lastname
Grossman
$
```

Now we can see that the value `Grossman` has just been stored in the variable `lastname` by the `read` command:

```
$ read lastname
Grossman
$ echo $lastname
Grossman
```

By the way, notice that you can echo the values of multiple variables at once.

```
$ echo $firstname $lastname
Jeff Grossman
```

As you will soon see, the `read` command is particularly useful in shell scripting. You can use it within a shell script to prompt users to input information, which is then stored in a shell variable and available for use by the shell script while it is running. You will also learn about **command line arguments**, which are values that can be passed to a script and automatically assigned to shell variables.

Summary

In this reading, you learned that:

- Shell variables store values and allow users to later access them by name
- You can create shell variables by declaring a shell variable and value or by using the `read` command

Authors

Jeff Grossman

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-05-04	1.4	Benny Li	Minor formatting changes
2023-04-27	1.3	D Teel-Friedman	Copy edit pass
2023-04-14	1.2	Nick Yi	ID Review
2023-03-22	1.1	Jeff Grossman	Added brief intro and minor edits
2022-12-23	1.0	Jeff Grossman	Created initial version of the reading

Examples of Pipes

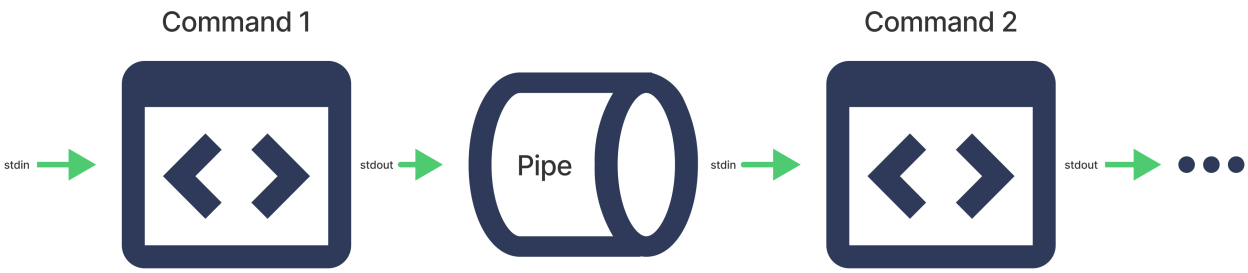
Learning Objectives

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 URLs

You can also use `curl` in combination with the `grep` command to extract components of URL data by piping the output of `curl` to `grep`.

Let's see how you can use this pattern to get the current price of Bitcoin (BTC) in USD.

First, find a public URL API. In this example, you will use one provided by CoinStats.

CoinStats provides a public API with no key required at `https://api.coinstats.app/public/v1/coins/bitcoin\?currency\=USD`, which returns some JSON about the current BTC price in USD.

You can see what this looks like by entering the above link in your browser.

Entering the following command returns the BTC price data, displayed as a JSON object:

```
$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency
{
  "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/"
    ]
  }
}
```

> **Note:** For the purpose of this reading, we've reformatted the output to make it easier to interpret. The actual output is a continuous stream of text.

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 `.`

Now that you have the `grep` statement that you need, you can pipe the BTC data to it using the `curl` command from above:

```
$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency
    grep -oE "\"price\":\s*[0-9]*?\.[0-9]*"
"price": 57907.78008618953
```

> **Tip:** The backslash `\` character used here after the pipe `|` allows you to write the expression on multiple lines.

Finally, to get *only* the value in the price field and drop the `"price"` label, you can use chaining to pipe the same output to another `grep`:

```
$ curl -s --location --request GET https://api.coinstats.app/public/v1/coins/bitcoin\?currency
    grep -oE "\"price\":\s*[0-9]*?\.[0-9]*" |\
```

```
    grep -oE "[0-9]*?\.[0-9]*"
57907.78008618953
```

This now displays only the numerical price without the label.

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

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-05-23	1.4	Benny Li	QA pass
2023-04-27	1.3	Nick Yi	QA pass
2023-04-14	1.2	Nick Yi	ID Review
2023-03-08	1.1	Jeff Grossman	Added new content
2021-11-09	1.0	Sam Prokopchuk	Initial version created

Reading: A Brief Introduction to Shell Variables

Learning Objectives

After completing this reading, you will be able to:

- Describe shell variables
- Create shell variables

What is a shell variable?

Shell variables offer a powerful way to store and later access or modify information such as numbers, character strings, and other data structures by name. Let's look at some basic examples to get the idea.

Consider the following example.

```
$ firstname=Jeff
$ echo $firstname
Jeff
```

The first line assigns the value `Jeff` to a new variable called `firstname`. The next line accesses and displays the value of the variable, using the `echo` command along with the special character `$` in front of the variable name to extract its value, which is the string `Jeff`.

Thus, we have created a new shell variable called `firstname` for which the value is `Jeff`.

This is the most basic way to create a shell variable and assign it to a value all in one step.

Reading user input into a shell variable at the command line

Here's another way to create a shell variable, using the `read` command. After entering

```
$ read lastname
```

on the command line, the shell waits for you to enter some text:

```
$ read lastname
Grossman
$
```

Now we can see that the value `Grossman` has just been stored in the variable `lastname` by the `read` command:

```
$ read lastname
Grossman
$ echo $lastname
Grossman
```

By the way, notice that you can echo the values of multiple variables at once.

```
$ echo $firstname $lastname
Jeff Grossman
```

As you will soon see, the `read` command is particularly useful in shell scripting. You can use it within a shell script to prompt users to input information, which is then stored in a shell variable and available for use by the shell script while it is running. You will also learn about **command line arguments**, which are values that can be passed to a script and automatically assigned to shell variables.

Summary

In this reading, you learned that:

- Shell variables store values and allow users to later access them by name
- You can create shell variables by declaring a shell variable and value or by using the `read` command

Authors

Jeff Grossman

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-05-04	1.4	Benny Li	Minor formatting changes
2023-04-27	1.3	D Teel-Friedman	Copy edit pass
2023-04-14	1.2	Nick Yi	ID Review
2023-03-22	1.1	Jeff Grossman	Added brief intro and minor edits
2022-12-23	1.0	Jeff Grossman	Created initial version of the reading