String Manipulation

In this lesson, you'll cover how to edit strings in bash, what extglobs are and how you can use them, and how to avoid common quoting problems with strings.

WE'LL COVER THE FOLLOWING

Since so much of working in bash is related to files and strings of text, the ability to manipulate strings is valuable.

While tools such as `sed`, `awk`, `perl` (and many others) are well worth learning, in this lesson I want to show you what is possible in bash - and it may be more than you think!

How Important is this Lesson?

This lesson is not essential. String manipulation in bash is only occasionally useful to know about, for example when the above-mentioned tools (`sed`, `awk`, `perl` *et al*) are not available.

The quoting tricks in this lesson are occasionally very useful, however.

String Length

String Length

One of the most common requirements when working with strings is to determine length:

```
A='12345678901234567890'
echo "${#A}"               # How long is 'A'?
echo "$#A"
```

Type the above code into the terminal in this lesson.

Terminal

Why did the **line 3** not 'work'?

Show Hint

String Editing

Bash provides a way to extract a substring from a string. The following example explains how to parse *n* characters starting from a particular position.

Work out what's going on here. You may need to consult the manual:

```
echo ${A:2}
echo ${A:2:3}
```

Type the above code into the terminal in this lesson.

You can replace sections of scripts using search and replace. The first part enclosed in `/` signs represents what's searched for, and the second what is replaced:

```
echo "${A/234/432}"
echo "${A//234/432}"
```

Type the above code into the terminal in this lesson.

What's going on in the second command above? How does it differ from the

first?

String Editing

Another commonly-required string operation is getting a substring.

This outputs everything from the third character in the string:

```
echo ${A:2}
```

Type the above code into the terminal in this lesson.

and this outputs two numbers from the fourth character in the string:

```
echo ${A:3:2}
```

Type the above code into the terminal in this lesson.

You can tell from this that the strings are 0-indexed, that is the first character is numbered zero, the second is numbered one, and so on.

You can also search and replace within strings:

```
B=0000000000
echo ${B/0/1}
```

Type the above code into the terminal in this lesson.

The simple pattern replaces only the first `0` with a `1`. If you want the pattern to go across all the matches in the string, then add another slash:

```
echo ${B//0/1}
```

Type the above code into the terminal in this lesson.

You can also convert lower case to upper case and vice versa. Using a single comma will turn the starting title case character into a lower case character:

```
C=AbCd
echo ${C,}
```

Type the above code into the terminal in this lesson.

whereas using two will lower-case the entire string:

```
echo ${C,,}
C=aAbCd
```

Type the above code into the terminal in this lesson.

Similarly, the caret (`^`) character will upper-case a string also:

```
echo ${C^}
echo ${C^^}
```

Type the above code into the terminal in this lesson.

And More...

There are many more string manipulation tricks you can use, but it is not worth trying to learn them all in one go.

If you want to study them in more detail and find out what's available (or ever find yourself wondering what can be done in bash with strings) then have a look at the man page under *Parameter Expansion*.

Extglobs and Removing Text

A more advanced means of working with strings is possible by using bash's **extglobs** functionality.

A word of warning here: although this functionality is useful to know, it is arguably less useful than learning the programs `sed` and `perl` for this purpose. It's also quite confusing to have this extra type of glob syntax to learn in addition to regular expressions (and there's even flavours of those too!). Feel free to skip this section if you feel it's too obscure.

```
shopt -s extglob
A="12345678901234567890"
B="  ${A}  "
```

Type the above code into the terminal in this lesson.

You've ensured that the `extglob` option is on, and created a new variable `B`

You've ensured that the `extglob` option is on, and created a new variable `B`, which is `A` with two spaces in front and behind. The pipe (`|`) character is used in the `echo` output to show where the spaces are.

```
echo "B      |${B}|"
echo "B#+( ) |${B#+( )}|"
echo "B#?( ) |${B#?( )}|"
echo "B#*( ) |${B#\*( )}|"
echo "B##+( )|${B##+( )}|"
echo "B##*( )|${B##*( )}|"
echo "B##?( )|${B##?( )}|"
```

Type the above code into the terminal in this lesson.

Using the bash man page, and experimenting, can you figure out the differences between the above extglobs? They are:

- `?( )`
- `+( )`
- `*( )`

Now try `%` instead of `#` above. What happens?

One potentially handy application of this is when having to remove leading zeroes from dates:

```
TODAY=$(date +%j)
TODAY=${TODAY##+(0)}
```

Type the above code into the terminal in this lesson.

Remember: `#` is to the left, and `%` is to the right on a (US) keyboard. Or 'hash' is before 'per cent' in the alphabet.

Quoting Hell

Quoting - as I'm sure you've seen - can get very complicated in bash very quickly.

Try this:

```
echo 'I really want to echo $HOME and I can't avoid it'
```

Type the above code into the terminal in this lesson.

Uh-oh. You're now stuck.

Can you see why?

Hit `CTRL-c` to get out of it.

So you might think to try this:

```
echo "I really want to echo $HOME and I can't avoid it"
```

Type the above code into the terminal in this lesson.

but this time the `$HOME` variable is evaluated and the output is not what is wanted.

Try this:

```
echo 'I really want to echo $HOME and I can'"'"'t avoid it'
```

Type the above code into the terminal in this lesson.

That works. Remember this trick as it can save you a lot of time!

String Manipulation Quiz

1 What will this output?

```
echo ${#A}
```

What You Learned

This lesson taught you about string management using pure bash. This is a relatively rare field to cover, but worth having seen in case you come across it. You learned:

- How to edit strings using pure bash
- How to determine string length
- What *extglobs* are and how you can use them
- How to avoid common quoting problems

What Next

In the next lesson you will look at bash's **autocomplete** functionality.

Exercises

1) Learn how to do all of the above things in `sed` too. This will take some research and time.

2) Learn how to do all of the above in `perl`. This will also take some research and time!

3) Construct a useful `echo` that has a double-quoted string with a single-quoted string inside, eg one that outputs:

```
He said 'I thought she'd said "bash was easy when $s are involved" but tha
t can’t be true!'
```