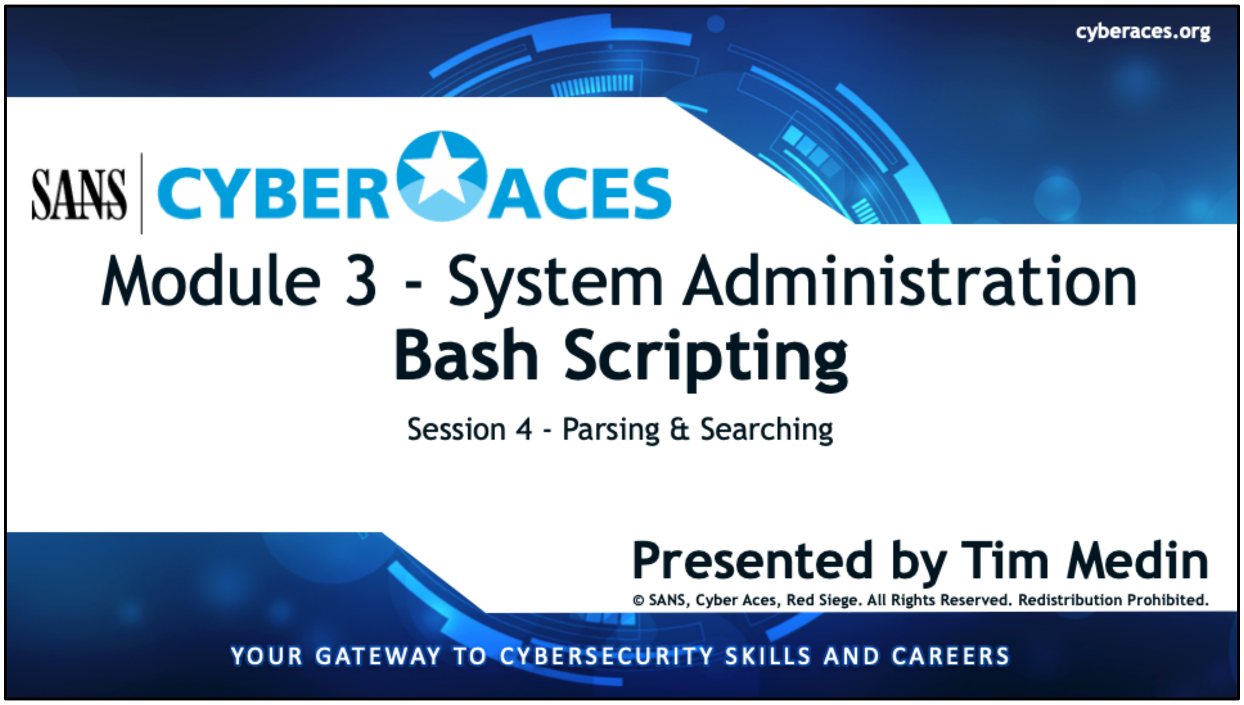

cyberaces.org

SANS | CYBER ACES

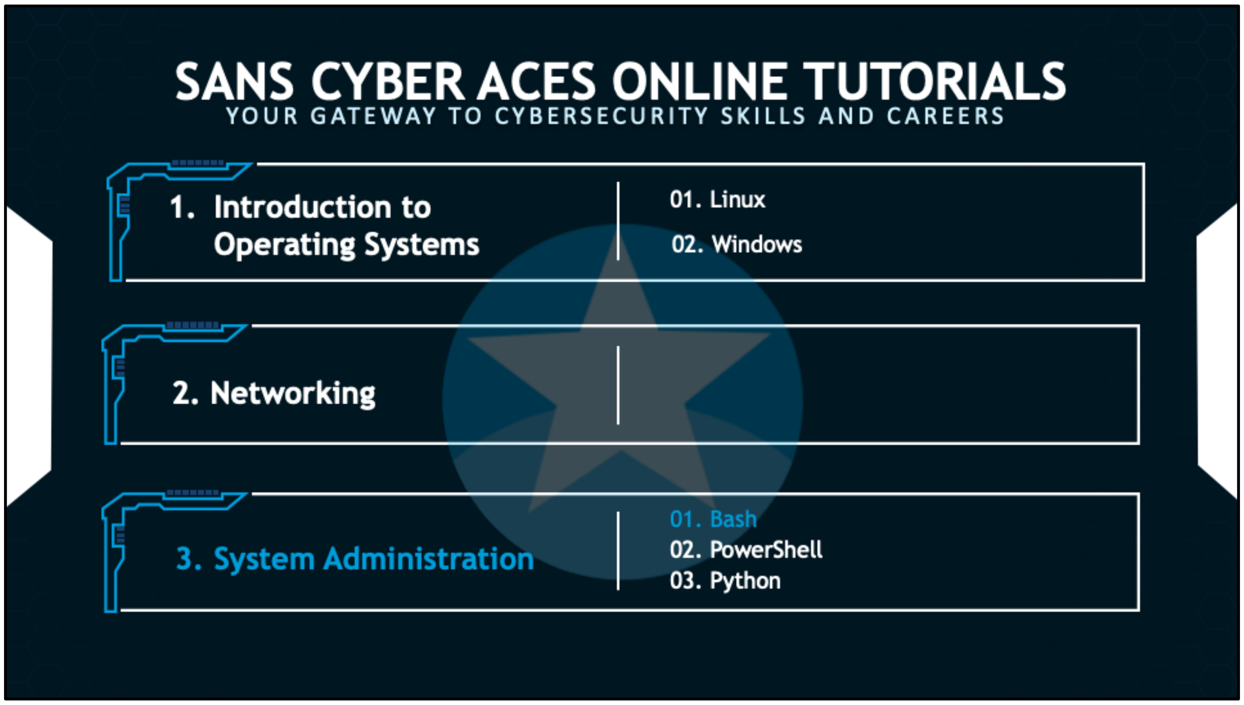
Module 3 - System Administration Bash Scripting

Session 4 - Parsing & Searching

Presented by Tim Medin

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

Welcome to Cyber Aces, Module 3! This module provides an introduction to Bash Scripting. In this session we will be discussion parsing and searching with Bash.



SANS CYBER ACES ONLINE TUTORIALS

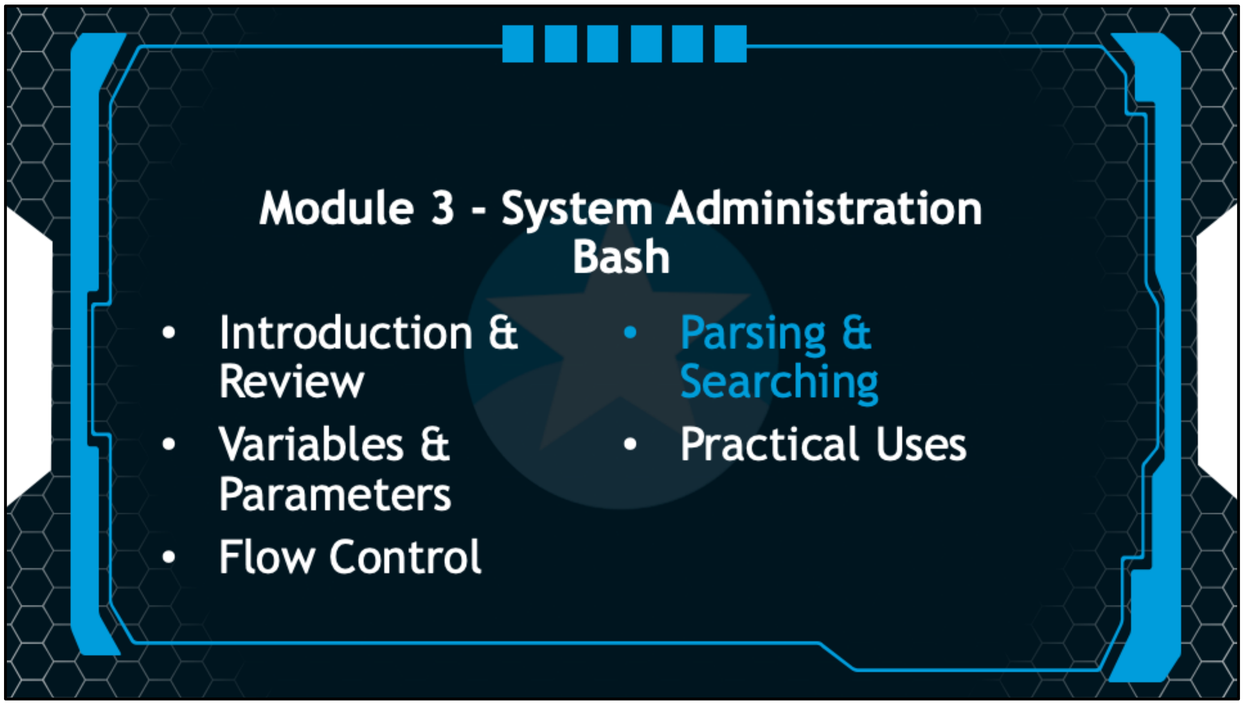
YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

1. Introduction to Operating Systems
   - 01. Linux
   - 02. Windows
2. Networking
3. System Administration
   - 01. Bash
   - 02. PowerShell
   - 03. Python

This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of system administration and scripting. . This session is part of Module 3, System Administration. This module is split into three sections, Bash, PowerShell, and Python. In this session, we will continue our examination of Bash.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

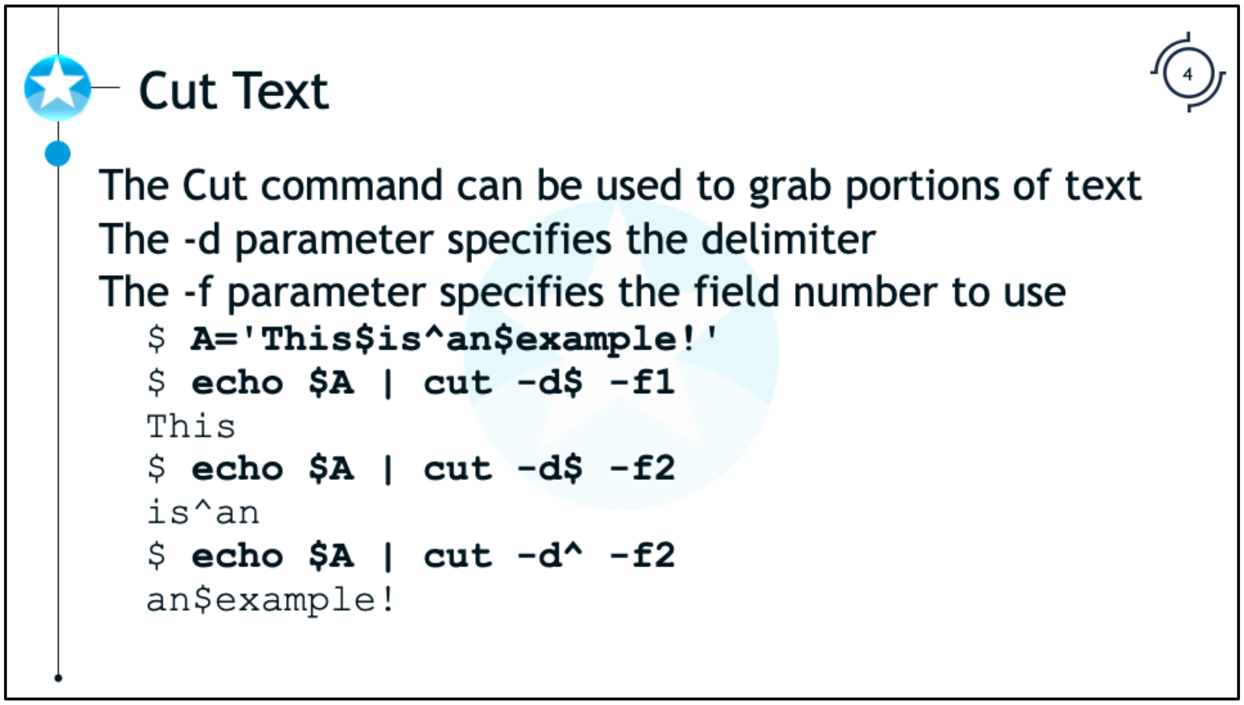
For more information about the Cyber Aces program, please visit the Cyber Aces website at https://CyberAces.org/.



Module 3 - System Administration Bash

- Introduction & Review
- Variables & Parameters
- Flow Control
- Parsing & Searching
- Practical Uses

Is this section, you will be introduced to ways to parse input files or output from other commands. We will also cover ways to search for files based on their attributes, searching within files, and a very powerful way of searching using regular expressions.



Cut Text

The Cut command can be used to grab portions of text
The -d parameter specifies the delimiter
The -f parameter specifies the field number to use

```
$ A='This$is^an$example!'
$ echo $A | cut -d$ -f1
This
$ echo $A | cut -d$ -f2
is^an
$ echo $A | cut -d^ -f2
an$example!
```

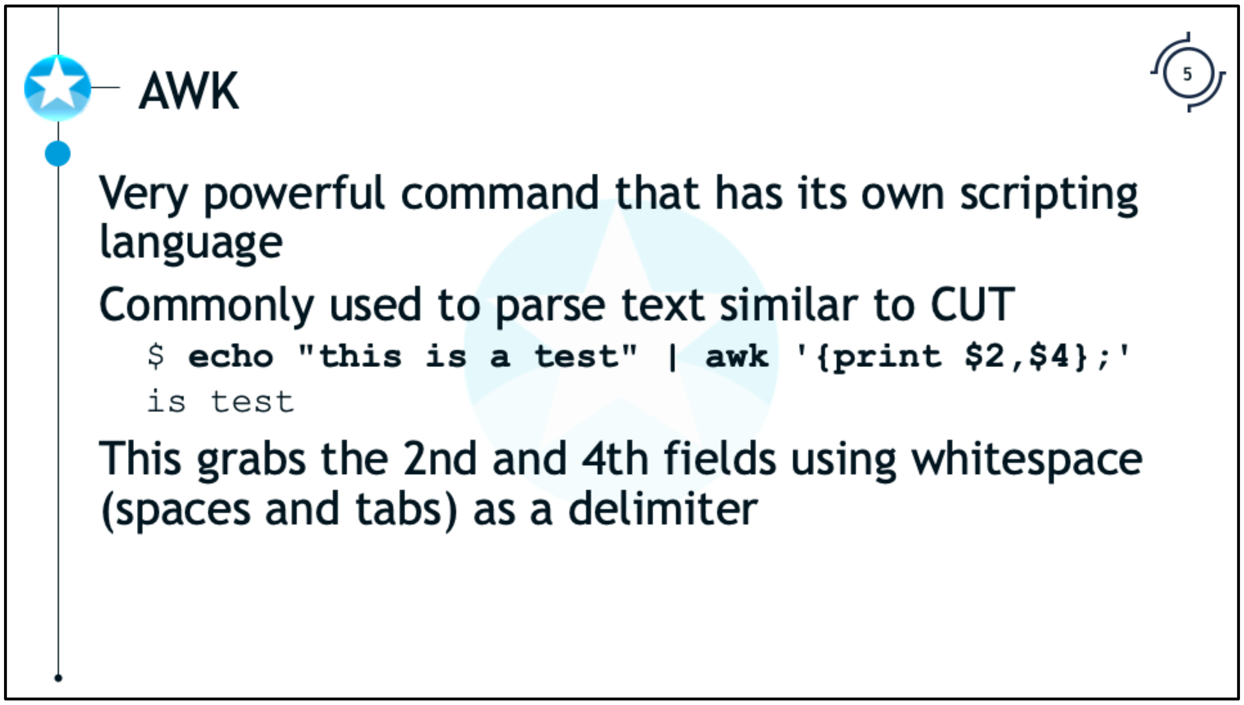
You can use cut to grab portions of text out of a string using the "cut" command. The "-d" parameter specifies the "delimiter" to use to separate the string, and the "-f" parameter specifies the "FIELD" number to use from the delimited string. Consider the following portion of code:

```
$ A='THIS$IS^A$TEST$STRING$EXAMPLE'
$ Fld1=`echo $A | cut -d$ -f1`
$ Fld2=`echo $A | cut -d$ -f2`
$ Fld3=`echo $A | cut -d$ -f3`
$ echo "The first field separated by a dollar sign is $Fld1"
$ echo "The second field separated by a dollar sign is $Fld2"
$ echo "The third field separated by a dollar sign is $Fld3"
$ Fld4=`echo $A | cut -d^ -f1`
$ echo "The first field separated by a carat (^) is $Fld4"
```

This will result in the following output:

```
The first field separated by a dollar sign is THIS
The second field separated by a dollar sign is IS^A
The third field separated by a dollar sign is TEST
The first field separated by a carat (^) is THIS$IS
```

This is very useful when you want to grab a piece of text from the output of a command or a file.


AWK

Very powerful command that has its own scripting language

Commonly used to parse text similar to CUT

```
$ echo "this is a test" | awk '{print $2,$4};'
is test
```

This grabs the 2nd and 4th fields using whitespace (spaces and tabs) as a delimiter

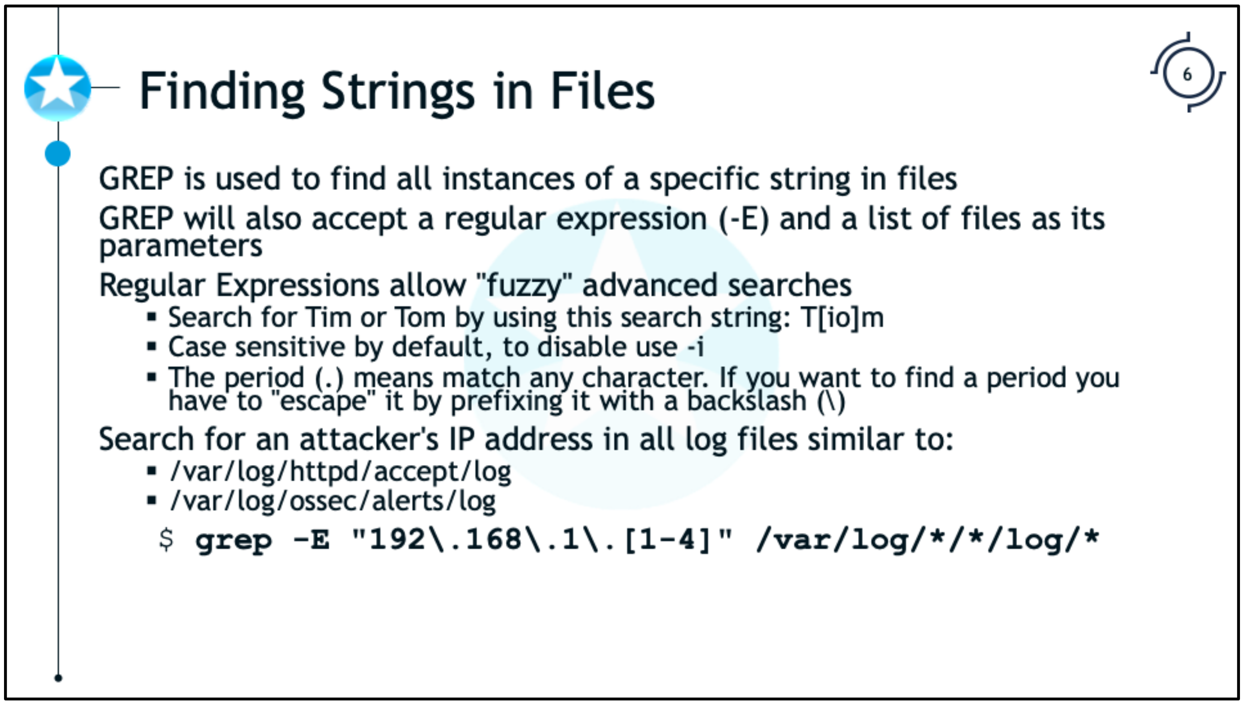
"AWK" is a very powerful command that has a scripting language all its own. The full use of AWK is beyond the scope of this introduction, but we will give examples of a few common uses of AWK in scripting. A frequent use of the AWK command in scripts is to isolate portions of text. Consider this code executed at a Bash prompt:

```
$ echo "this is a test" | awk '{print $2,$4};'
is test
```

The AWK script contained inside the brackets prints the second parameter and the fourth parameter of AWK's input, resulting in "is test" being printed to the screen.

Check out these other creative ways to use AWK to manipulate text.

https://www.redsiege.com/ca/awk



Finding Strings in Files

- GREP is used to find all instances of a specific string in files
- GREP will also accept a regular expression (-E) and a list of files as its parameters
- Regular Expressions allow "fuzzy" advanced searches
  - Search for Tim or Tom by using this search string: T[io]m
  - Case sensitive by default, to disable use -i
  - The period (.) means match any character. If you want to find a period you have to "escape" it by prefixing it with a backslash (\)
- Search for an attacker's IP address in all log files similar to:
  - /var/log/httpd/accept/log
  - /var/log/ossec/alerts/log

```
$ grep -E "192\.168\.1\.[1-4]" /var/log/*/*/log/*
```

The "grep" command is used to find all instances of a specific string in files. Grep will accept a regular expression and a list of files as its parameters.

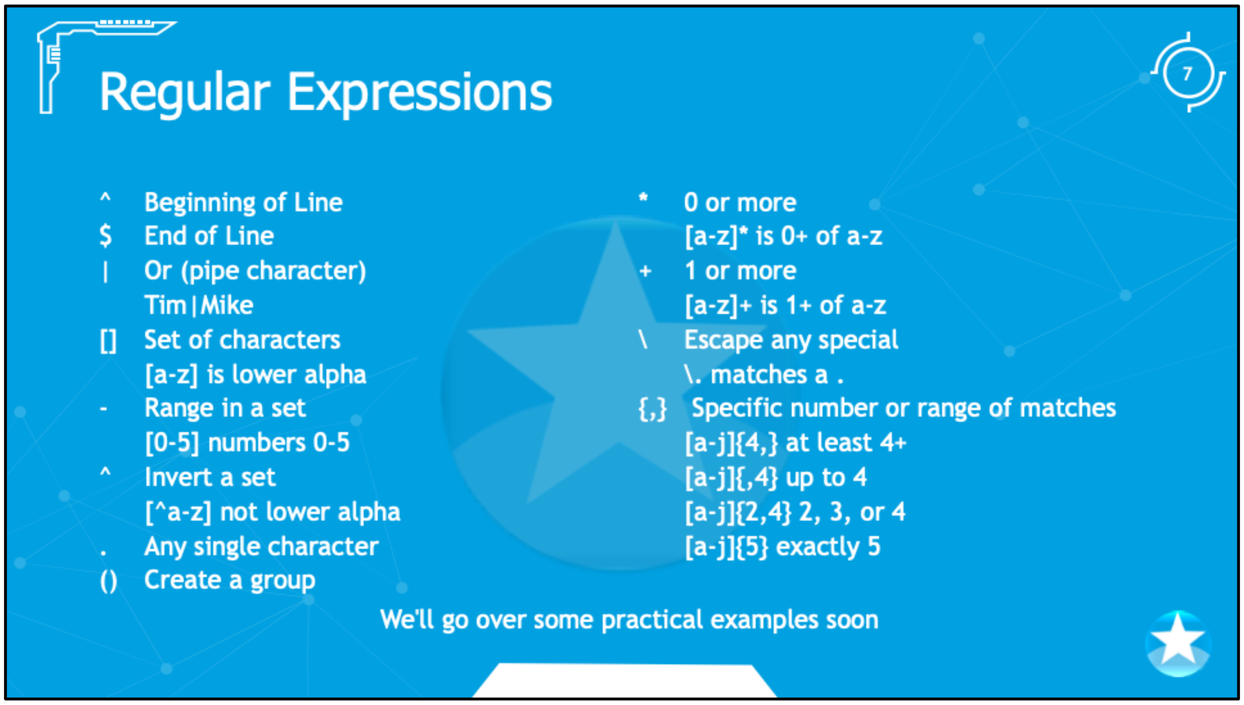
Imagine the following directories stores logs that might have recorded the attackers IP address:

- /var/log/httpd/accept/log
- /var/log/ossec/alerts/log

You want to find all the files that contain the IP address 192.168.1.1. You could use the following grep statement:

```
$ grep -iE "192\.168\.1\.1" /var/log/*/*/log/*
```

It is worth mentioning that if you're going to search for IP addresses in logs, you should be aware of the format of the IP addresses in your logs. For example, an application may record your IP address as 192.168.1.1, or it might record it as 192.168.001.001. Take a few minutes to review your data source and make sure you build your regular expressions properly.



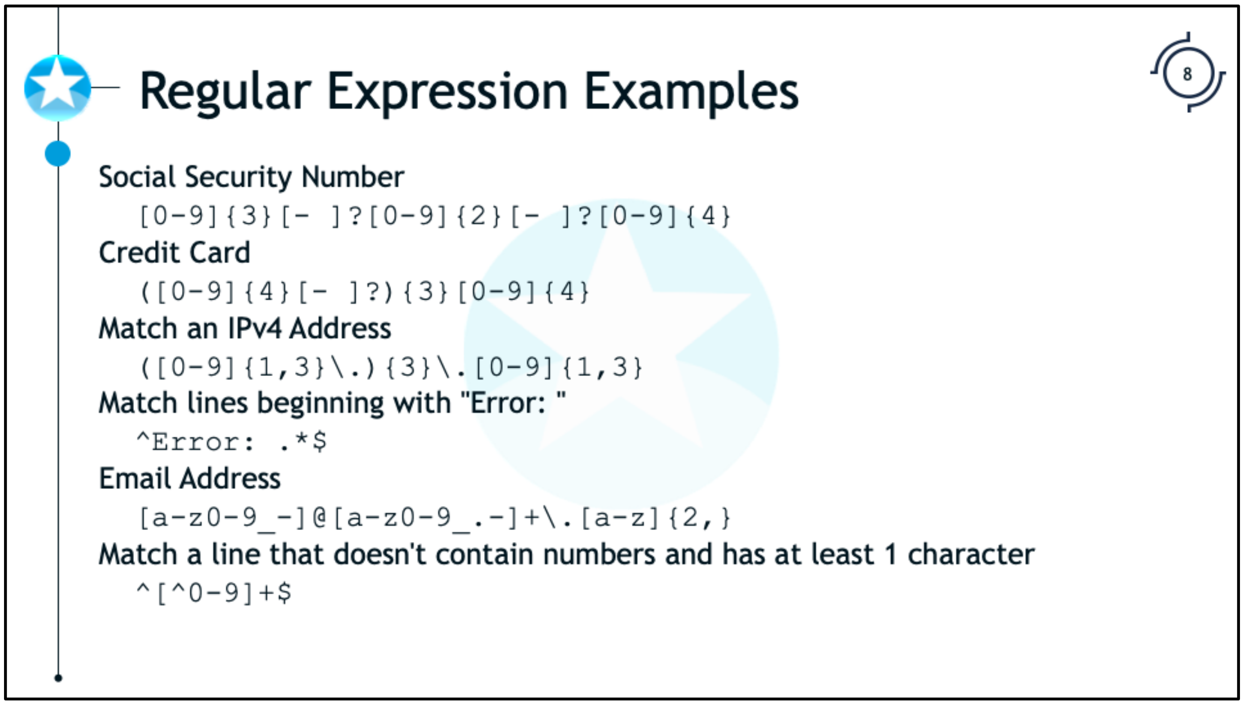
Regular Expressions

^	Beginning of Line	*	0 or more
$	End of Line		[a-z]* is 0+ of a-z
|	Or (pipe character)	+	1 or more
	Tim|Mike		[a-z]+ is 1+ of a-z
[]	Set of characters	\	Escape any special
	[a-z] is lower alpha		\. matches a .
-	Range in a set	{,}	Specific number or range of matches
	[0-5] numbers 0-5		[a-j]{4,} at least 4+
^	Invert a set		[a-j]{,4} up to 4
	[^a-z] not lower alpha		[a-j]{2,4} 2, 3, or 4
.	Any single character		[a-j]{5} exactly 5
()	Create a group		

We'll go over some practical examples soon

There are a number of special special characters used in Regular Expressions that allow for very granular searches.

^	The beginning of the line
$	The end of line
|	An Or
[ ]	A set of characters; it can have multiple. i.e to match hex output [0-9a-f]
–	A range in a set (see above)
^	Inverts a set. For example, to match anything that isn't a number use [^0-9]
.	Any single character
*****	The preceding item will be matched 0 or more times
+	The preceding item will be matched 1 or more times
\	Used to "escape" another character so it will not be interpreted as a special character. i.e. An IP address 10\.10\.10\.10
{ , }	Used to match the preceding a specific number (or range) or times



Regular Expression Examples

Social Security Number

```
[0-9]{3}[- ]?[0-9]{2}[- ]?[0-9]{4}
```

Credit Card

```
([0-9]{4}[- ]?){3}[0-9]{4}
```

Match an IPv4 Address

```
([0-9]{1,3}\.){3}\.[0-9]{1,3}
```

Match lines beginning with "Error: "

```
^Error: .*$
```

Email Address

```
[a-z0-9_-]@[a-z0-9_.-]+\.[a-z]{2,}
```

Match a line that doesn't contain numbers and has at least 1 character

```
^[^0-9]+$
```

Regular Expression are extremely useful and powerful. To get a perfect match takes some time, so many times we look for a "good enough" search when looking for results for the sake of speed and efficiency. Below each example is explained.

Social Security Number is in the format of ###-##-#### but the dashes can be removed or be replaced with spaces. So we need 3 numbers, an optional dash or space, 2 more numbers, an optional dash or space, and another 4 numbers.

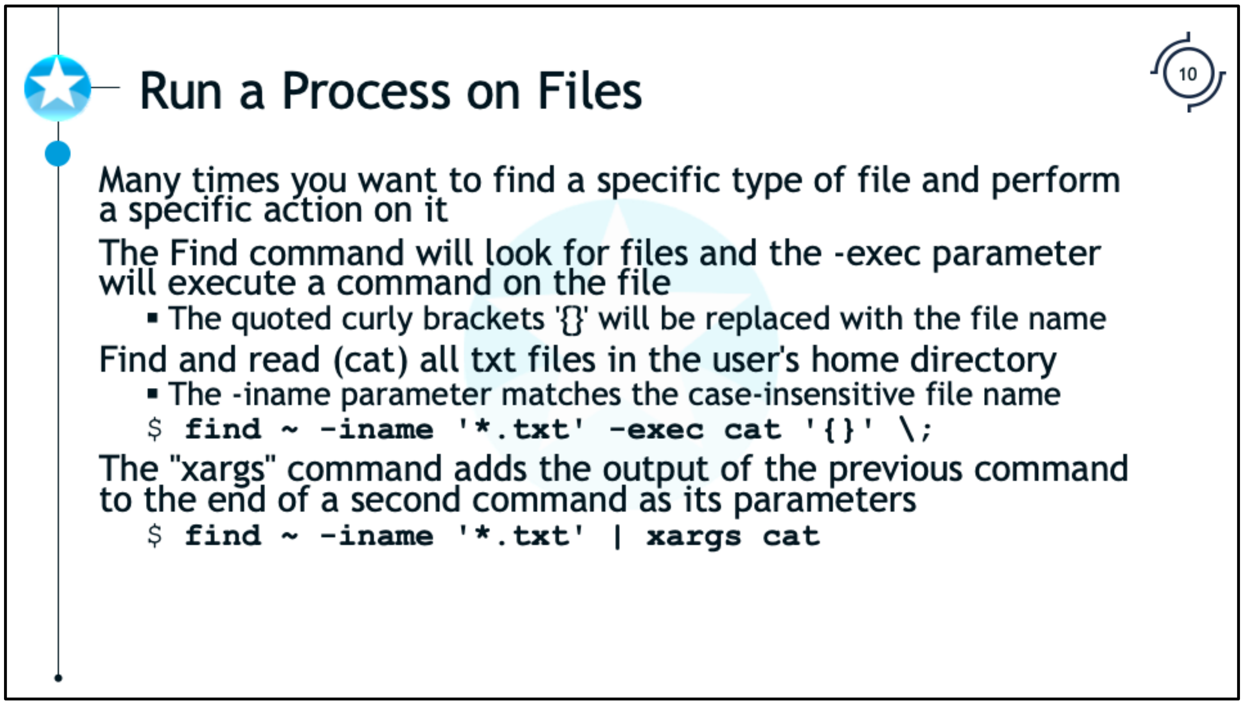
[0-9]{X} will match exactly X number of numbers
[-]? will accept a dash or a space if it exists
This will accept match input in the following:

123-45-6789
123 45 6789
123456789
123 45-6789
12345 6789

The credit card example is very similar. We need a number in the form of ####-####-####-#### but the dash may be omitted or replaced with spaces.

([0-9]{4}[-]?) looks for a set of 4 numbers followed by an optional dash or space
([0-9]{4}[-]?){3} matches the set above three times. It would match the first 12 (4*3) digits in our credit card number. We need a final four digits ([0-9]{4}) to complete our credit card number

The IPv4 address is quite similar to the Credit Card in that we look for the "###." three times before looking for the final ###. Of course, each octet can contain between 1 and 3 numbers. Also, the dot is a special character and we need to match the dot exactly (not any character) so we have to escape it with the backslash (\).



Run a Process on Files

Many times you want to find a specific type of file and perform a specific action on it

The Find command will look for files and the -exec parameter will execute a command on the file

- The quoted curly brackets '{}' will be replaced with the file name

Find and read (cat) all txt files in the user's home directory

- The -iname parameter matches the case-insensitive file name

```
$ find ~ -iname '*.txt' -exec cat '{}' \;
```

The "xargs" command adds the output of the previous command to the end of a second command as its parameters

```
$ find ~ -iname '*.txt' | xargs cat
```

Another routine task is to perform some action on a set of files that match some criteria. For example, it may be necessary to move all files that are bigger than a specific size or older than a specific date to an archived location.

An easy way to do this is with the "-exec" parameter of the find command. For example, the following command finds all files in your home directory that end in ".txt" (case insensitive because it is "-iname" instead of "-name") and print the contents of those files to the screen with cat. The brackets '{}' are replaced with results of the 'find' command.

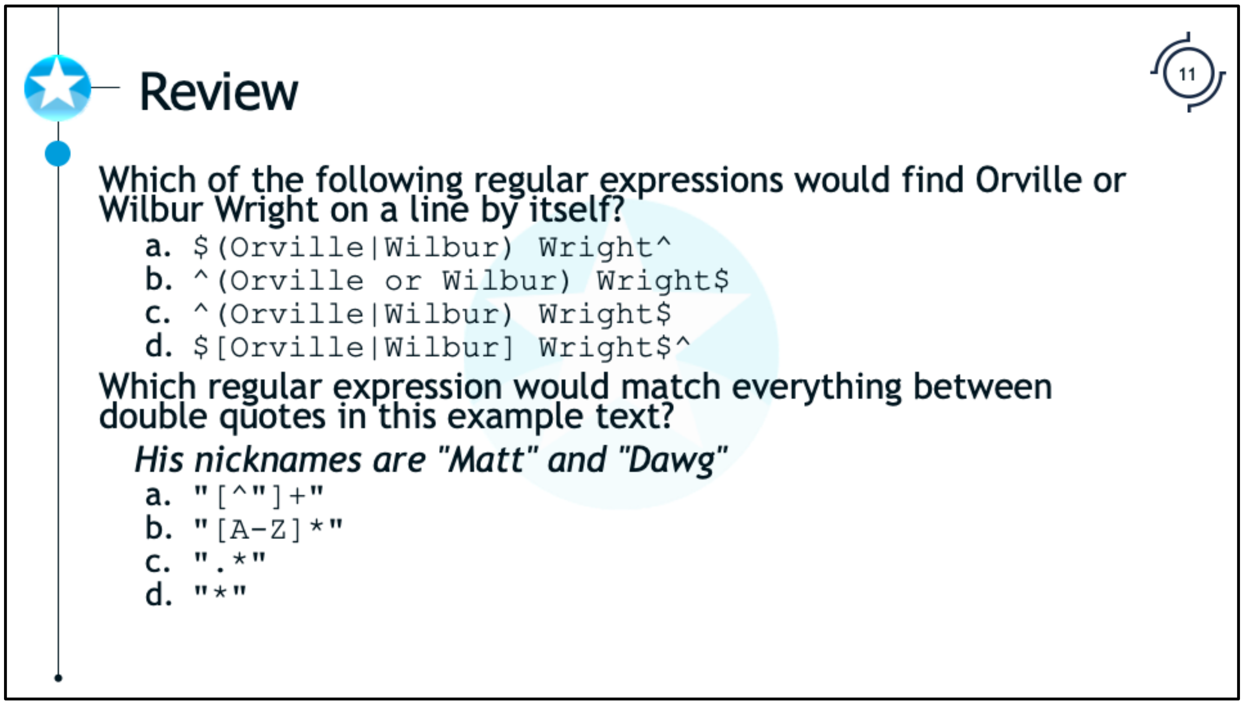
```
$ find ~ -iname '*.txt' -exec cat '{}' \;
```

Read this article on the use of the find command:

http://dsl.org/cookbook/cookbook_10.html

This works fine in many circumstances. However, a very long list of file names from the find command will cause find to produce an error rather than execute the commands. Find is also limited to executing the commands one at a time. If you run into either of these problems, you can use "xargs" to execute your code. "xargs" adds the output of the previous command to the end of a second command as its parameters. So we could rewrite our find command above like this:

```
$ find ~ -iname '*.txt' | xargs cat
```



Review

Which of the following regular expressions would find Orville or Wilbur Wright on a line by itself?

a. `$(Orville|Wilbur) Wright^`
b. `^(Orville or Wilbur) Wright$`
c. `^(Orville|Wilbur) Wright$`
d. `$[Orville|Wilbur] Wright$^`

Which regular expression would match everything between double quotes in this example text?

His nicknames are "Matt" and "Dawg"

a. `"[^"]+"`
b. `"[A-Z]*"`
c. `".*"`
d. `"*"`

Which of the following regular expressions would find Orville or Wilbur Wright on a line by itself?

a. `$(Orville|Wilbur) Wright^`

b. `^(Orville or Wilbur) Wright$`

c. `^(Orville|Wilbur) Wright$`

d. `$[Orville|Wilbur] Wright$^`

Which regular expression would match everything between double quotes in this example text?

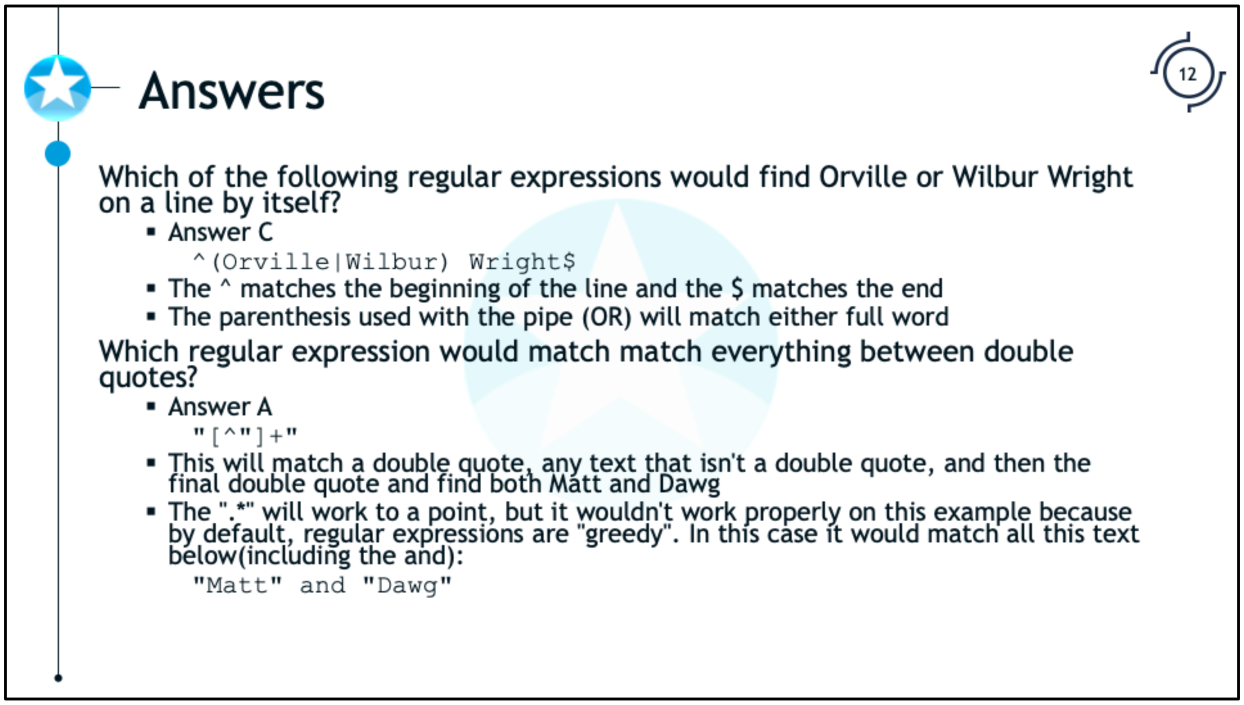
```
His nicknames are "Matt" and "Dawg"
```

a. `"[^"]+"`

b. `"[A-Z]*"`

c. `".*"`

d. `"*"`



Answers

12

Which of the following regular expressions would find Orville or Wilbur Wright on a line by itself?

- Answer C

```
^(Orville|Wilbur) Wright$
```

- The ^ matches the beginning of the line and the $ matches the end
- The parenthesis used with the pipe (OR) will match either full word

Which regular expression would match match everything between double quotes?

- Answer A

```
"[^"]+"
```

- This will match a double quote, any text that isn't a double quote, and then the final double quote and find both Matt and Dawg
- The ".*" will work to a point, but it wouldn't work properly on this example because by default, regular expressions are "greedy". In this case it would match all this text below(including the and):

```
"Matt" and "Dawg"
```

Which of the following regular expressions would find Orville or Wilbur Wright on a line by itself?

Answer C

```
^(Orville|Wilbur) Wright$
```

The ^ matches the beginning of the line and the $ matches the end

The parenthesis used with the pipe (OR) will match either full word

Which regular expression would match match everything between double quotes?

Answer A

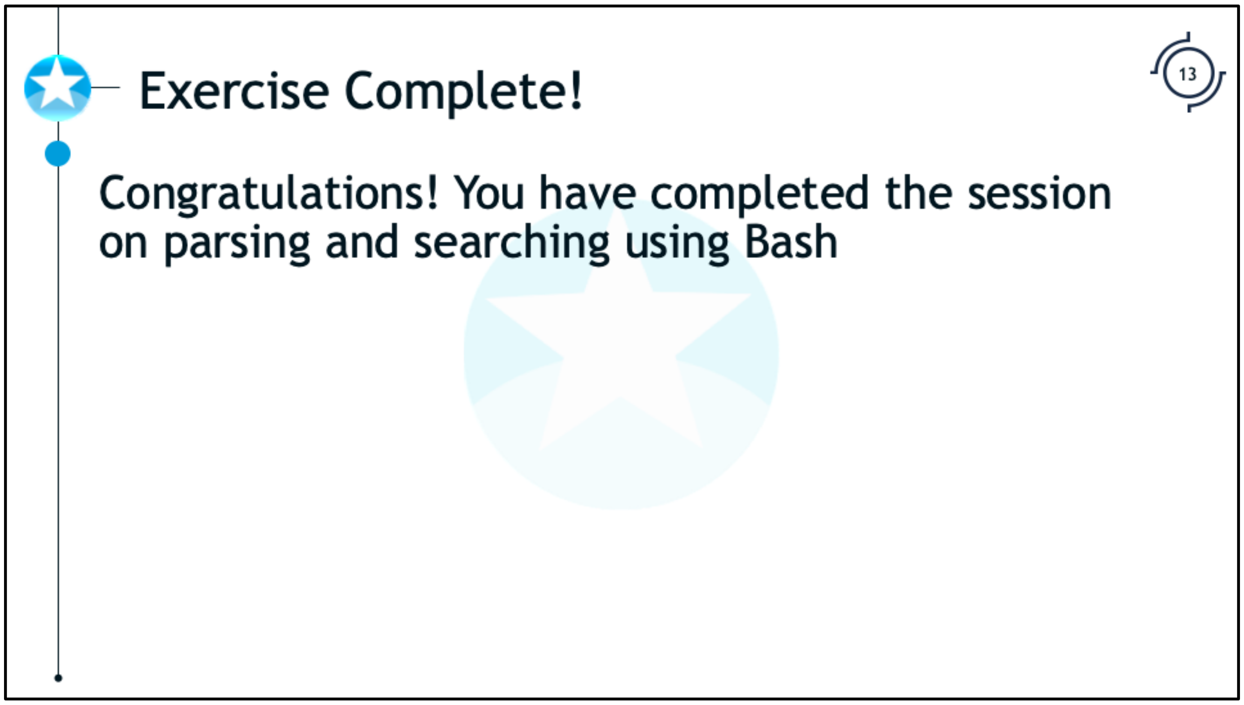
```
"[^"]+"
```

This will match a double quote, any text that isn't a double quote, and then the final double quote and find both Matt and Dawg

The `".*"` will work to a point, but it wouldn't work properly on this example as it would grab all this text (including the and):.

```
"Matt" and "Dawg"
```

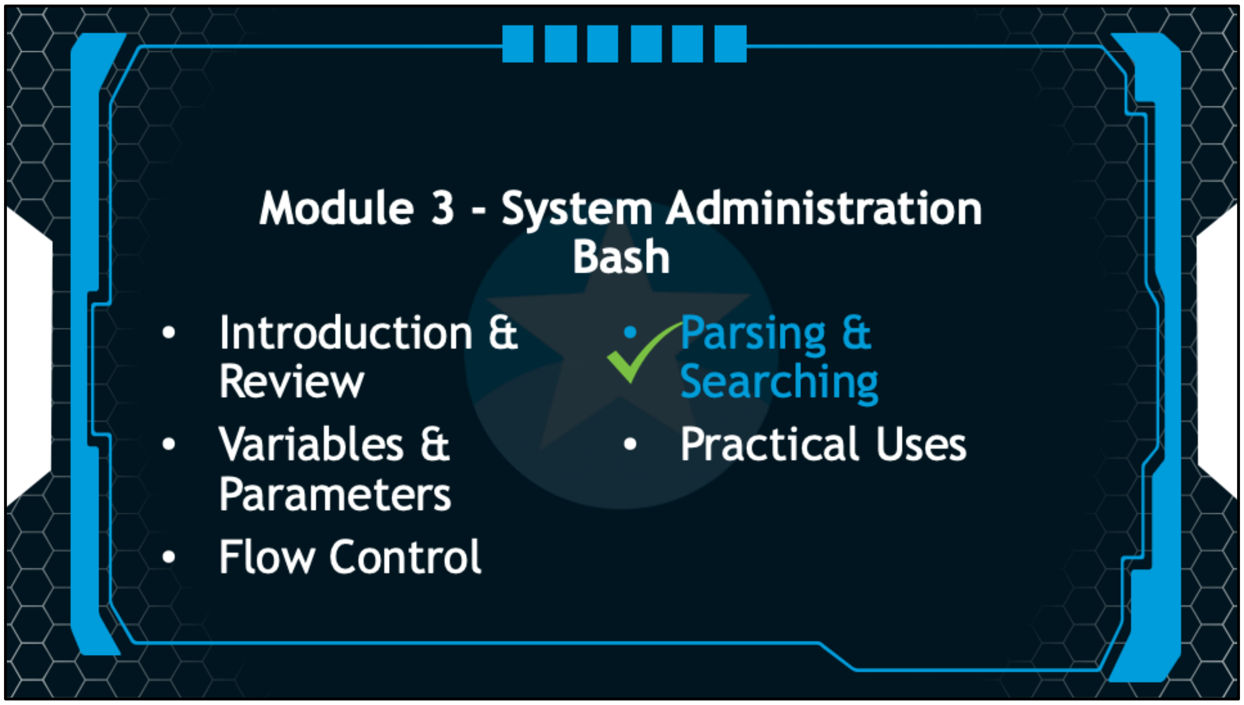
Regular Expressions are, by default, greedy, meaning it will grab as many characters as possible. This is why it will also grab the *and*. It will start with the double quote before the *M* and continue until the double quote after the *g*.



Exercise Complete!

Congratulations! You have completed the session on parsing and searching using Bash

Congratulations! You have completed the session on parsing and searching using Bash.



Module 3 - System Administration Bash

- Introduction & Review
- Variables & Parameters
- Flow Control
- ✓ Parsing & Searching
- Practical Uses

In the next session we will discuss practical uses for Bash and Bash scripts.