BASH CLI CONTROL

...

TLO KNOWLEDGE AND SKILLS

Conditions:

- Given a classroom, applicable references, and a practical exercise, the Cyber Mission Force student will demonstrate an understanding of command line interface control.

Knowledge:

- Identify why regular expressions are necessary in bash scripting.
- Understand the purpose behind utilizing the CLI to execute scripts.

Skills:

- Working knowledge of how to control and manipulate expressions.
- Knowledge of how to edit scripts and streamline processes in the CLI.

OBJECTIVES

- Describe the purpose of regular expressions
- Describe the utilization of the Command Line Interface (CLI)
- Perform Text Manipulation using regular expressions

COMMAND LINE EDITING

BASH has three useful command-line utilities that allow us to use patterns to accomplish tasks.

Grep - "Global Regular Expression Print," allowing us to search for patterns using regex.

Awk - allows you to use patterns to manipulate the output.

Sed - "Stream editor" and allows us to edit our input stream of text with the ability to filter text based on patterns.

GREP

Grep allows you to search files for strings and print the matching lines. Grep will search each line of a file, grep itself does not change lines of files.

-E allows use for Extended Regex to search for patterns

-o prints out only the match

-A Prints n number of lines after the match (cannot be used in conjunction with -o)

-B Prints n number of lines before the match (cannot be used in conjunction with -o)

-C Prints n number of lines around the match (cannot be used in conjunction with -o)

-v searches for all lines NOT containing the string/pattern

GREP EXAMPLES

```
grep "Name" file.txt

# Searches for the string “Name” in file.txt, Name will always be capital without specifiying otherwise because the N is capital in our string

grep -A 2 "Name" file.txt

# Searches for the string “Name” in file.txt, printing out the line with our match plus the 2 lines after

grep -B 2 "Name" file.txt

# Searches for the string “Name” in file.txt, printing out the line with our match plus the 2 lines before

grep -C 2 "Name" file.txt

# Searches for the string “Name” in file.txt, printing out the line with our match plus the 2 lines around

grep -o "Name" file.txt

# Searches for the string “Name” and will only print out the string Name in each line with that string
```

AWK

Allows us to use pattern matching to perform operations on specific lines of files or strings.

Separates strings of text by default using a space ' '. The delimiter can be changed based on the statement you write.

Uses many of the control statements that we've seen in our BASH scripts including if statements, while statements, for statements, etc.

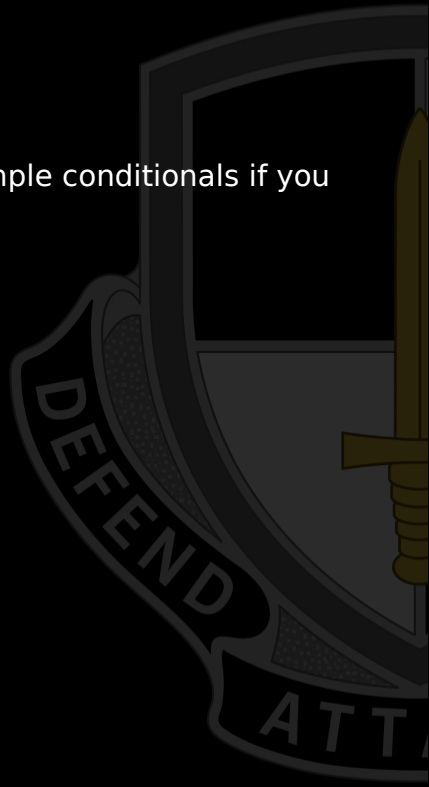
AWK EXAMPLES

```
awk '/Name/ {print}' HR_Employee_list.txt

# This will print out all the lines containing "Name" in the file


awk '/Name/ {print NR " "$2" "$3}' HR_Employee_list.txt

# This will print out all the lines containing "Name". NR allows us to print out the row number, and we have included spaces between items 2 and 3 (first and last names)


awk '/Name/ { if (length($2) > max) max = length($2) } END {print max}' HR_Employee_list.txt

# This will print out the length of the longest first name
```

SED

Sed is a Stream editor allowing you to edit a stream of characters with pattern matching and simple conditionals if you choose.

This works better for files that are not columned unlike our previous command awk.



SED EXAMPLES

```
echo "Hello World" | sed s/World/Class/

# This uses the substitute function of sed to replace the pattern World with the pattern Class


sed -Ee 's/[0-9]{3}-[0-9]{2}-[0-9]{4}/XXX-XX-XXXX/' HR_Employee_list.txt

# -e must be used to read a file. We are replacing all social security numbers here


sed -Ee 's/[0-9]{3}-[0-9]{2}-[0-9]{4}/XXX-XX-XXXX/w; file.txt' HR_Employee_list.txt

# writes changes to new file (-i will allow the changes to be made to the original file)
```

CHECK ON LEARNING

1. True or False. Grep allows you to replace the contents of files based on pattern matching?

False

2. What type of pattern matching do we use in BASH?

Regex

REGULAR EXPRESSIONS

A regular expression (also known as regex) is a form of patter matching.

This means that I can use a specified pattern that when compared to a text file will return both the words cat and bat based on a simple pattern.

We can use pattern matching within conditionals as well as using it to extract specific lines using our grep command.

It's important to realize that there are multiple types of syntax available; we will be using extended regex.

When creating regex patterns it must be written character by character, not word by word.

SINGLE CHARACTER MATCHING

If you want to match a specific character you can simply include that character in your pattern, no need to include any special characters (remember, BASH is case-sensitive).

For example, if I want to match every letter a in a string or file my regex pattern would simply be ‘a’.

If I wanted to match every instance of the word cat my regex pattern would be ‘cat’.

SINGLE CHARACTER MATCHING

When writing regex patterns that aren't a single specific string we must write a character set, this is a set of characters that we are looking for.

These can be numbers, letter, symbols, or a combination of the 3.

We are able to use ranges of numbers and letters within these character sets. Character sets lie within brackets.

- [a-z] matches any lowercase letter
- [A-Z] matches any uppercase letter
- [0-9] matches any number character
- [A-Za-z] matches any uppercase or lowercase character
- [A-Za-z0-9] matches any uppercase or lowercase character or any number character

SINGLE CHARACTER MATCHING

To match a specific set of numbers or letters we would simply include those within our character set, you could include a range as well as single letters/numbers. The order of your character set does not matter.

- [abcdefg01234] will match a single character that is either an a, b, c, d, e, f, g, 0, 1, 2, 3, 4
- [a-g0-4] will match a single character that is either an a, b, c, d, e, f, g, 0, 1, 2, 3, 4
- [Ag.El$40*] will match a single character that is either an A, E, g, l, 0, 4, '.', '$', '*'

SINGLE CHARACTER MATCHING

To match all values except for what you include in your character set you would include the caret within the bracket and at the beginning of your character set.

- `[^a-z]` will match any character that is not a lowercase letter
- `[^aK4$]` will match any character that is not an a, K, 4, $

SINGLE CHARACTER MATCHING EXAMPLES


```
----

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'


echo $string | grep -Eo '[a-z]'

echo $string | grep -Eo '[0-9]'

echo $string | grep -Eo '[A-Z]'

echo $string | grep -Eo '[$ @!\(?.]'

echo $string | grep -Eo '[^An@!]'

----
```

SINGLE CHARACTER MATCHING

When using regex in BASH scripts or as comparisons we can use the automatic variable ${BASH_REMATCH} to print out the match that we found.

```
#!/bin/bash

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'

pattern='[a-z]'

if [[ $string =~ $pattern ]]

then

  echo "pattern found \"${BASH_REMATCH}\""

fi
```

MULTI CHARACTER MATCHING

If you want to match more than just a single character you can either type out multiple character sets or you can use quantifiers.

Using a quantifier will allow you to search for a specific character set a specific number of times.

- {n} will match the previous character set n number of times
- {n,m} will match the previous character set between n and m number of times
- {,m} will match the previous character set up to m number of times
- {n,} will match the previous character set at least n number of times
- * will match the previous character set zero or more times
- + will match the previous character set one or more times
- ? will match the previous character set zero or one times

MULTI CHARACTER MATCHING EXAMPLES


```
----

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'


echo $string | grep -Eo '[a-z]{2}'

echo $string | grep -Eo '[0-9]{,3)'

echo $string | grep -Eo '[A-Z]{1,}'

echo $string | grep -Eo '[$ @!\(?.]*'

echo $string | grep -Eo '[^An@!]?'

----
```

MULTI CHARACTER MATCHING EXAMPLES


```
----
#!/bin/bash

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'
pattern='[a-z]+'

if [[ $string =~ $pattern ]]
then
  echo "pattern found \"${BASH_REMATCH}\""
fi
----
```

SPECIAL CHARACTER MATCHING

There are a few other important characters in regex that will allow for better precision when writing our regex patterns.

- `.` will match any single character
- `\` is an escape character allowing you to search for special characters without including them in a character set
- `^` when outside a character set will search only at the beginning of a string
- `$` when outside of a character set will search for the pattern at the end of a string
- `( | )` or, matches one of the things in ()

SPECIAL CHARACTER MATCHING EXAMPLES


```
----

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'


echo $string | grep -Eo '[a-z].$'

echo $string | grep -Eo '^[a-z].'

echo $string | grep -Eo '[0-9]\ [a-z]?[A-Z]'

echo $string | grep -Eo '[0-9] *[a-z]?\?'

echo $string | grep -Eo '[$ @!\(?.]*.{1,4}'

echo $string | grep -Eo '(This|That)'

----
```

SPECIAL CHARACTER MATCHING EXAMPLES


```
----
#!/bin/bash

string='ThisIsmy$tring C@Nw3 F!nD AnY\maT(h3s?.'
pattern='[a-z]+.*'

if [[ $string =~ $pattern ]]
then
  echo "pattern found \"${BASH_REMATCH}\""
fi
----
```

COUNTING MATCHES

To determine the number of matches found you can use the grep -c command to count lines with matches or you can use the wc -l command to count every match.

To use wc you must first use the grep -o option to print out ONLY your matches. This will give every match on its own line allowing us to count the number of lines with wc.

Grep -c will give you the number of lines containing matches but if you have multiple matches per line it will not include those number of matches. This is why wc is the recommended way to count lines.

CHECK ON LEARNING

1. Write a string that will match this regex: [0-9].\\t[a-z]+

   8f\tsss

2. Write a string that will match this regex: (True | False)[0-9a-z]?\?

   True8?