Edith Cowan University
School of Science

CSI6203 Scripting Languages

Module 8

Regular Expressions

Contents

- Pattern matching using Regex
- BRE and ERE patterns
- Using regex with grep
- Using regex with sed

Learning Objectives

After finishing this module, you should be able to:

- Understand simple regular expressions and use them to match specific text patterns
- Develop your own regular expressions to improve text parsing scripts
- Use regex with grep and sed

What is a regular expression?

Regular Expressions

- Regular Expressions (or regex for short) are used to match patterns in text
- A regex text pattern is used allow a regex engine to find a match
- Regex works like an advanced way of searching

Regex engines

- There are two types of regular expression engine supported by bash commands

- BRE
  - The Basic Regular Expression Engine
- ERE
  - The Extended Regular Expression Engine

- These are supported by many other commands such as `grep`, `sed` and `awk`

Regex engines

- For these examples we'll use `grep` which is specifically designed for using regex

- grep stands for **G**lobal **R**egular **E**xpression **P**rint

Simple BRE matching

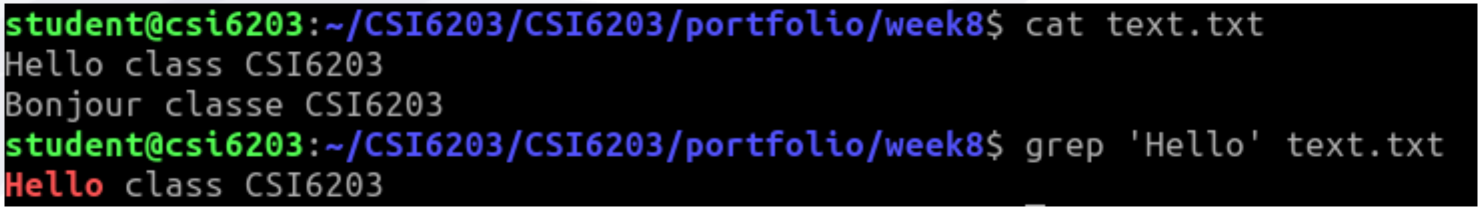
- Simple matching can be done by simply stating the text you wish to match on

```
grep 'Hello' text.txt
```

(Search for the pattern “Hello” in the file text.txt)

Regex Pattern

- Simple patterns can contain any text but some characters that have special meanings in regex may need to be escaped
  - E.g. ‘*[].^${}\+?|()’



```
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ cat text.txt
Hello class CSI6203
Bonjour classe CSI6203
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ grep 'Hello' text.txt
Hello class CSI6203
```

Anchors and Wildcards

Anchor characters

- Regex patterns can use special characters called Anchor Points to represent specific locations within the text

- The two most common anchor points are
  - The start of the line ‘^’
  - The end of the line ‘$’

```
grep '^Hello' text.txt
```

(Search for lines that start with “Hello” in the file text.txt)

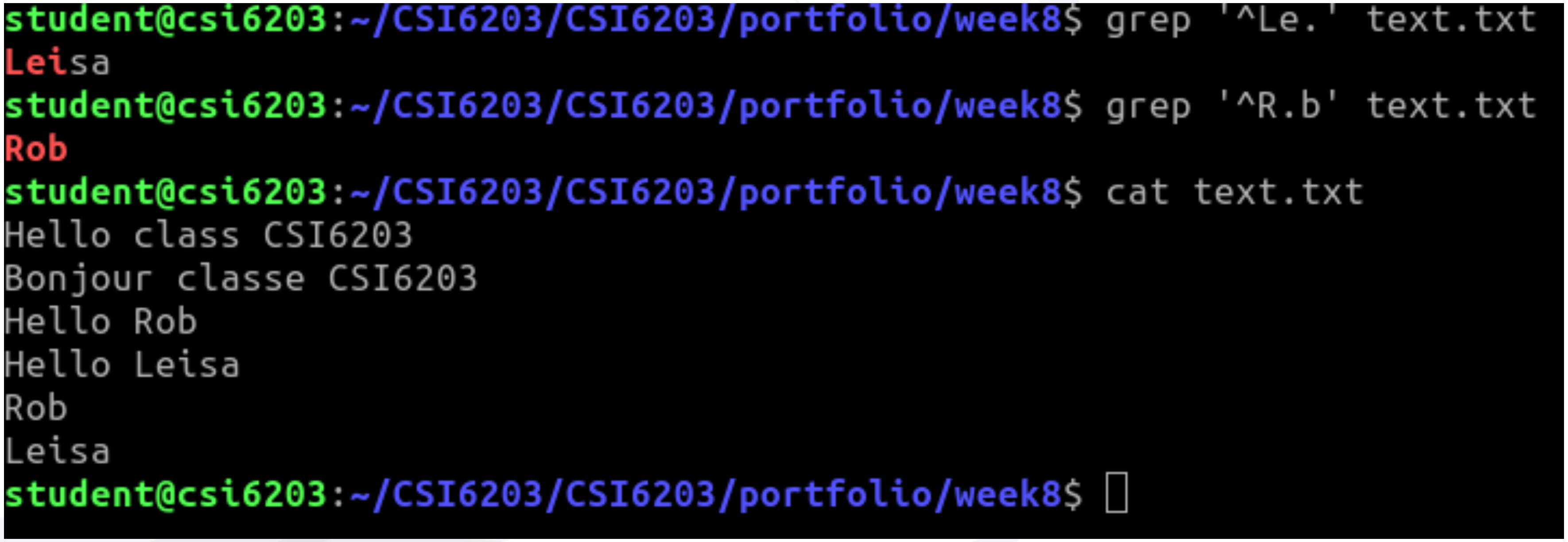
Wildcard characters

- Wildcards are characters that could match a range of characters.
- In regex, the most common wildcard is dot '.'
- The dot character can be used to represent any character
  - E.g. find lines that start with Rib, Rob, Rub, etc

```
grep '^R.b' text.txt
```

Wildcard characters

Example output from previous slide



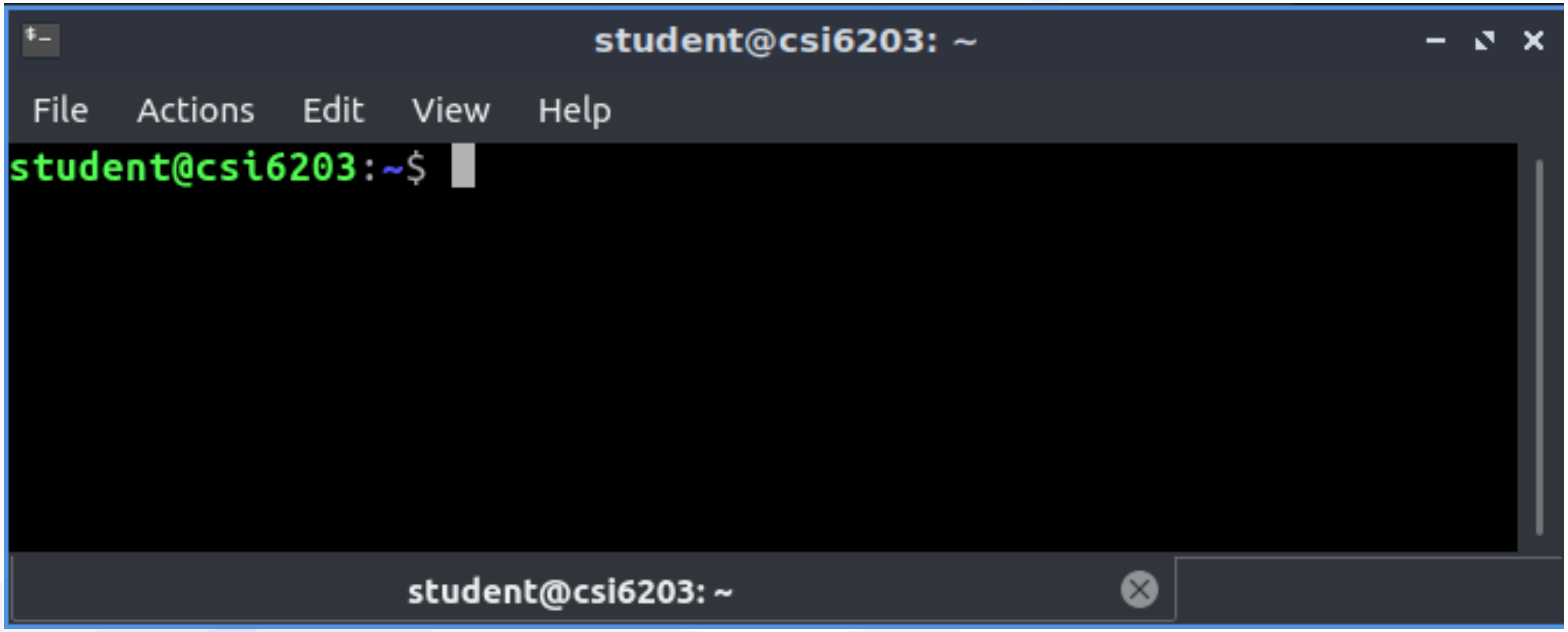
```
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ grep '^Le.' text.txt
Leisa
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ grep '^R.b' text.txt
Rob
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ cat text.txt
Hello class CSI6203
Bonjour classe CSI6203
Hello Rob
Hello Leisa
Rob
Leisa
student@csi6203:~/CSI6203/CSI6203/portfolio/week8$ 
```

Classed wildcards

- What if you want to match only a specific set of characters?
- Using square brackets [] you can restrict a wildcard to be only one of a set of values
  - Find lines that end with an R followed by a vowel followed by a b (Rib, Rob but not Rxb)

```
grep 'R[aeiou]b$' text.txt
```

Example



```
student@csi6203:~$ 
```

Ranged classes

- Square brackets can also specify a range of potential characters

```
grep '[A-Z]' text.txt
```

- It works with numbers too!

```
grep '[1-5]' text.txt
```

Extended Regex

- ERE can also match with several other collections of classes

```
grep -e '[[:digit:]]' text.txt
```

Pattern	Effect
[[:alpha:]]	Alphabetical character A-z, a-z
[[:alnum:]]	Alphanumeric character A-z, a-z, 0-9
[[:digit:]]	Digit 0-9
[[:upper:]]	Uppercase A-Z
[[:lower:]]	Lowercase a-z
[[:space:]]	Any whitespace character (space tab newline)
[[:blank:]]	Space or tab
[[:punct:]]	Punctuation character e.g. “!,.;”

Repetition and optionality in regex

Asterisk

- In regex, the asterisk can be used to repeat the previous part of the pattern 0 or more times.

- E.g. the following pattern search:

```
grep 'ye*s' text.txt
```

- Would find the words
  - yes, yees, yeeeeeeeees, ys, etc.

Asterisk

- The asterisk can be used with other regex characters too

- E.g. the following pattern search:

```
grep 'y[ea]*s' text.txt
```

- Would find the words
  - yes, yeeees, yas, yaaaaas, etc.

ERE

- In the ERE syntax, there are more powerful patterns for matching including
- +
- ?
- {}
- |
- ()

Plus +

- The Plus character “+” acts the same way as asterisk “*” except instead of 0 or more repetitions, there must be at least one

- E.g. the following pattern search:

```
grep 'y[ea]+s' text.txt
```

- Would find the words
  - “yes”, “yeeeees”, “yas”, “yaaaaas”, but not “ys”

Question Mark ?

- The Question Mark character “?” acts as an “optional” meaning that the preceding character may or may not be there

- E.g. the following pattern search:

```
grep 'b?ash' text.txt
```

- Would find the words
  – “bash” and “ash”

Curly Braces {}

- Braces can be used to define a specific number of repetitions of a character or sequence.

- E.g. the following pattern search:

```
grep 'R.{4}t' text.txt
```

- Would find the words
  - “Robert” and “Rabbit” and “Rupert”

OR and expression grouping

Expression Grouping

- Using parentheses “()” regex patterns can be grouped together to allow for complex combinations

```
grep -E '^regex can be (very)+ confusing' text.txt
```

- In this case the + is applied to the entire group

Expression Grouping

- Using parentheses “()” regex patterns can be grouped together to allow for complex combinations

```
grep -E '^regex can be (very)+ confusing' text.txt
```

- In this case the + is applied to the entire group
- Could match:

“regex can be very confusing”

Expression Grouping

- Using parentheses “()” regex patterns can be grouped together to allow for complex combinations

```
grep -E '^regex can be (very)+ confusing' text.txt
```

- In this case the + is applied to the entire group
- Could match:

“regex can be very very confusing”

Expression Grouping

- Using parentheses “()” regex patterns can be grouped together to allow for complex combinations

```
grep -E '^regex can be (very)+ confusing' text.txt
```

- In this case the + is applied to the entire group
- Could match:

“regex can be very very very very very very very very very very confusing”

OR |

- In bash, the pipe operator “|” is usually used for redirecting the output of one script or command to the input of another.
- However, in regex, it has a different meaning.

OR |

- In regex, the | character means OR and can be used to match one of several options

- E.g. the following pattern search:

```
grep -E '(bash)|(fish)$' text.txt
```

- Would find any line that ends with either the word “bash” or the word “fish”

Regex and sed

Regex and sed

- Regex doesn’t just work in grep
- Many other commands can support it too, including sed

```
sed '/^favorite/p' foods.txt
```

Regex and sed

- The combination of regex and sed together allows for some very complex text processing

- Find any line that starts with the word “favourite” and then replace any alphabetical text after a space character to contain the word “gnocchi”

```
sed -i '/^favorite/ s/[[:blank:]][[:alpha:]]*/ Gnocchi/g'
foods.txt
```

Regex and sed

- Before

foods.txt

```
Pie
Toast
Rice
Favorite: Spaghetti
Noodles
```

- After

foods.txt

```
Pie
Toast
Rice
Favorite: Gnocchi
Noodles
```

Summary

- Terms to review and know include:
  - BRE
  - ERE
  - Regex
  - Pattern matching

References and Further Reading

- Ebrahim, M. and Mallet, A. (2018) Mastering Linux Based Scripting (2nd Ed) Chapter 11, pp 194-215
- http://regular-expressions.info/engine.html
- http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html
- https://regexr.com/
- https://regex101.com/