DIS08 – Data Modeling

04 – Regular expressions and searching in files

Philipp Schaer, Technische Hochschule Köln, Cologne, Germany

Version: WS 2021

Technology
Arts Sciences
TH Köln

Disclaimer

This lesson is based on the Library Carpentry

- https://librarycarpentry.org/lc-shell/

And RegexOne

- https://regexone.com

And some slides by Mandy Neumann

Agenda

Last week's agenda

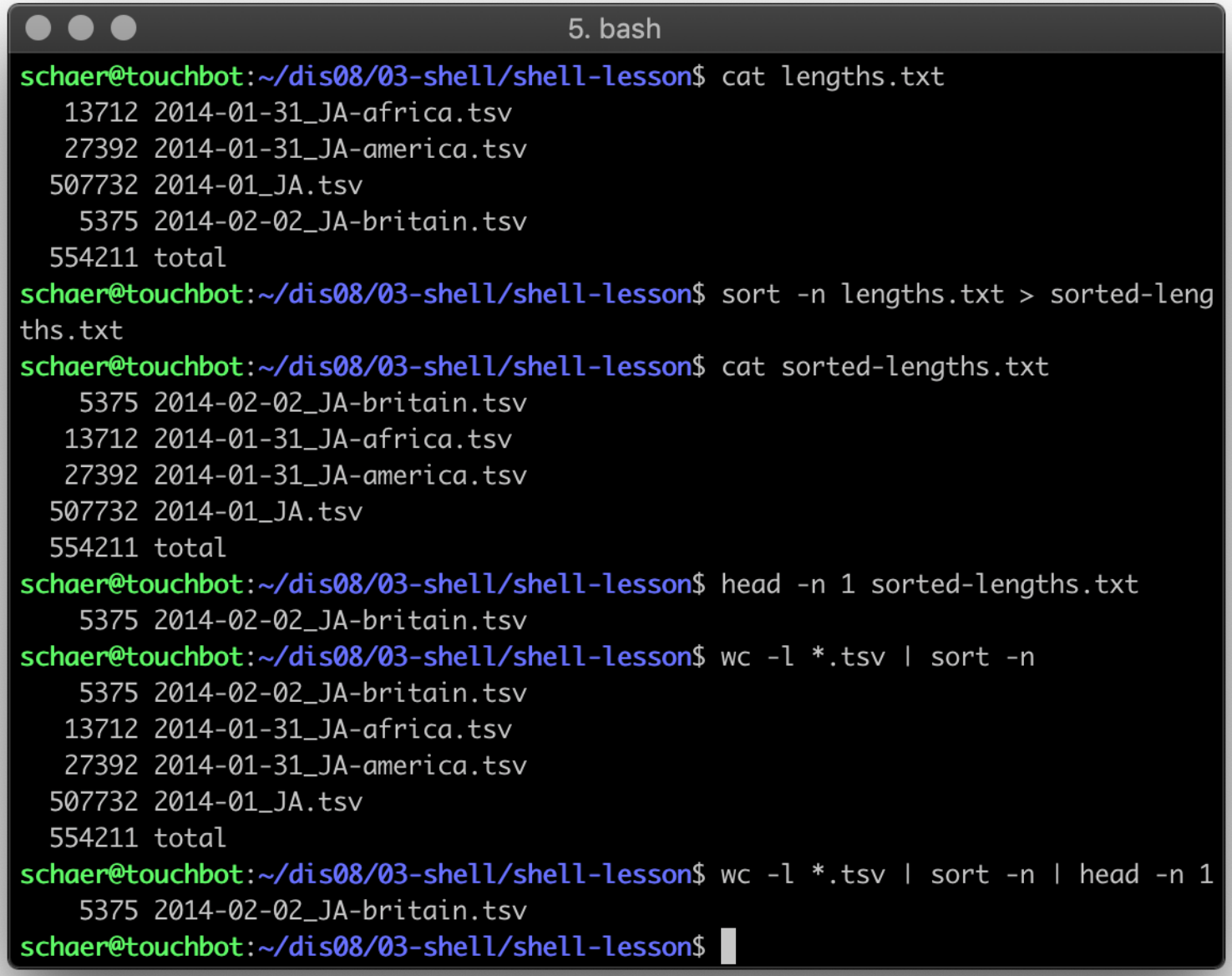
- What is the shell?
- Files and directories
- History and tab completion
- Counting and sorting contents in files
- Pipes and redirection

This week

- Regular expressions
- Mining or searching in files

Next week

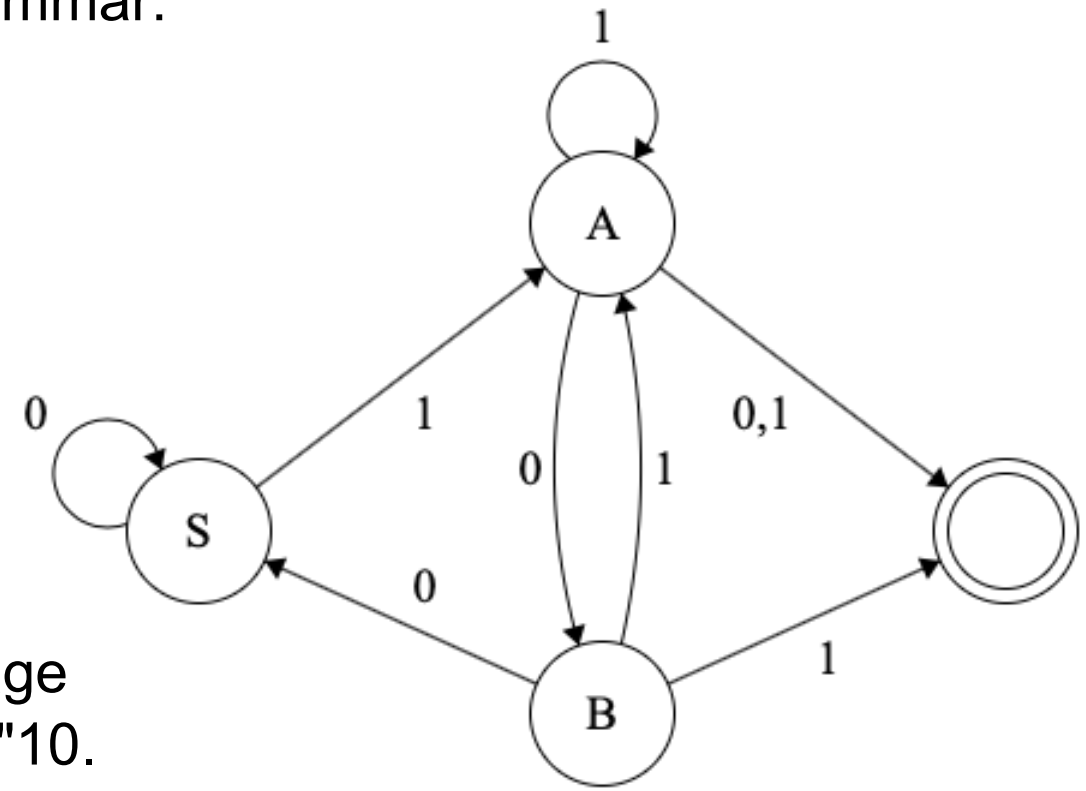
- Open Data and where to get it
- Working with CSV data
- Excel in a nutshell and why the shell is a better alternative!


5. bash
schaer@touchbot:~/dis08/03-shell/shell-lesson$ cat lengths.txt
13712 2014-01-31_JA-africa.tsv
27392 2014-01-31_JA-america.tsv
507732 2014-01_JA.tsv
5375 2014-02-02_JA-britain.tsv
554211 total
schaer@touchbot:~/dis08/03-shell/shell-lesson$ sort -n lengths.txt > sorted-leng
ths.txt
schaer@touchbot:~/dis08/03-shell/shell-lesson$ cat sorted-lengths.txt
5375 2014-02-02_JA-britain.tsv
13712 2014-01-31_JA-africa.tsv
27392 2014-01-31_JA-america.tsv
507732 2014-01_JA.tsv
554211 total
schaer@touchbot:~/dis08/03-shell/shell-lesson$ head -n 1 sorted-lengths.txt
5375 2014-02-02_JA-britain.tsv
schaer@touchbot:~/dis08/03-shell/shell-lesson$ wc -l *.tsv | sort -n
5375 2014-02-02_JA-britain.tsv
13712 2014-01-31_JA-africa.tsv
27392 2014-01-31_JA-america.tsv
507732 2014-01_JA.tsv
554211 total
schaer@touchbot:~/dis08/03-shell/shell-lesson$ wc -l *.tsv | sort -n | head -n 1
5375 2014-02-02_JA-britain.tsv
schaer@touchbot:~/dis08/03-shell/shell-lesson$

Regular Languages

Regular languages can be generated by regular grammars and recognized by (Non-)**Deterministic Finite Automata**.

- Example regular grammar:
  - S -> 0S | 1A
  - A -> 0B | 1A | 0 | 1
  - B -> 0S | 1A | 1

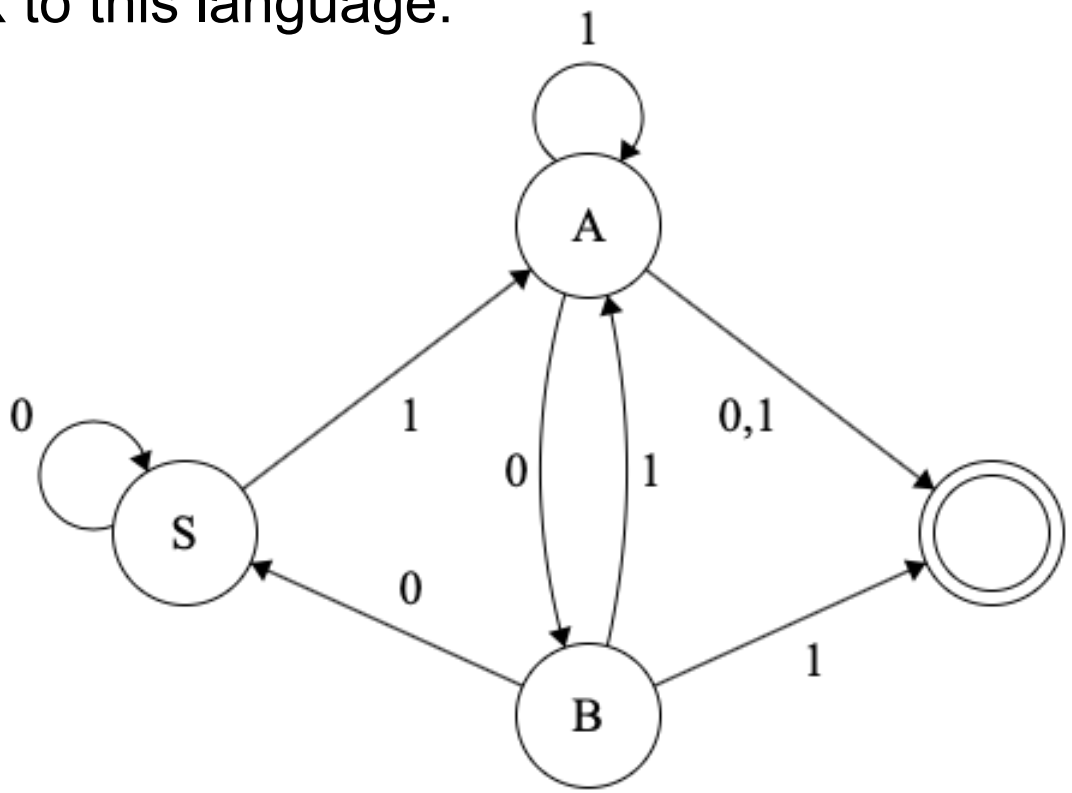


- Words in this language end on "01", "11" or "10.

Regular Expressions

Regular expressions serve the same purpose, i.e. generating and recognizing words of regular languages.

- Corresponding regex to this language:
  - **[01]*(01|11|10)**
  - weird… but powerful!



An Introduction to regex, and the ABCs

Regular expressions are extremely useful in extracting information from text such as

- code,
- log files,
- spreadsheets,
- or even documents.

- The first thing to recognize when using regular expressions is that **everything is essentially a character** (char), and we are **writing patterns** to match a specific sequence of chars (also known as a string).
- Most patterns use normal ASCII, which includes letters, digits, punctuation and other symbols on your keyboard like %#$@!.

#1 Matching letters

Try writing a pattern that matches all three rows, **it may be as simple as the common letters on each line**

Task	Text
Match	abcdefg
Match	abcde
Match	abc
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#1 Matching letters

Try writing a pattern that matches all three rows, **it may be as simple as the common letters on each line**

Task	Text
Match	**abc**defg
Match	**abc**de
Match	**abc**
abc	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#2 Matching digits

- **Characters** include normal **letters**, but **digits** as well. In fact, numbers 0-9 are also just characters and if you look at an ASCII table, they are listed sequentially.
- Over the various steps, you will be introduced to a number of special **metacharacters** used in regular expressions that can be used to match a specific type of character.
- In this case, the character **\d** can be used in place of **any digit from 0 to 9**. The preceding slash distinguishes it from the simple **d** character and indicates that it is a metacharacter.

#2 Matching digits

Try writing a pattern that matches all the digits in the strings below.

Task	Text
Match	abc123xyz
Match	define "123"
Match	var g = 123;
?	

abc…	Letters
123…	**Digits**
\d	**Any digit**
\D	**Any non-digit char**
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#2 Matching digits

Try writing a pattern that matches all the digits in the strings below.

Task	Text
Match	abc**123**xyz
Match	define "**123**"
Match	var g = **123**;
`123`	

abc…	Letters
123…	**Digits**
\d	**Any digit**
\D	**Any non-digit char**
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#2 Matching digits

Try writing a pattern that matches all the digits in the strings below.

Task	Text
Match	abc**123**xyz
Match	define "**123**"
Match	var g = **123**;
`\d\d\d`	

abc…	Letters
123…	**Digits**
\d	**Any digit**
\D	**Any non-digit char**
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#3 The dot

- In some card games, the Joker is a **wildcard** and can represent any card in the deck. With regular expressions, you are often matching pieces of text that you don't know the exact contents of, other than the fact that they share a common pattern or structure (eg. phone numbers or zip codes).
- Similarly, there is the concept of a **wildcard**, which is represented by the **.** (dot) metacharacter, and can **match any single character** (letter, digit, whitespace, everything).
- You may notice that this actually overrides the matching of the period character, so in order to specifically match a period, you need to **escape** the dot by using a slash **\.** accordingly.

#3 The dot

Try to write a single pattern that can match the first three strings, but not the last (to be skipped). You may find that you will have to escape the dot metacharacter to match the period in some of the lines.

Task	Text
Match	cat.
Match	896.
Match	?=+.
Skip	abc1
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	**Any char**
\.	**Period**
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#3 The dot

Try to write a single pattern that can match the first three strings, but not the last (to be skipped). You may find that you will have to escape the dot metacharacter to match the period in some of the lines.

Task	Text
Match	**cat.**
Match	**896.**
Match	**?=+.**
Skip	abc1
`...\.`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	**Any char**
\.	**Period**
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#4 Matching specific characters

- The dot metacharacter from the last example is pretty powerful, but sometimes **too** powerful. If we are matching phone numbers for example, we don't want to validate the letters "(abc) def-ghij" as being a valid number!
- There is a method for **matching specific characters** using regular expressions, by defining them inside **square brackets**. For example, the pattern **[abc]** will only match a **single** a, b, or c letter and nothing else.

#4 Matching specific characters

Below are a couple lines, where we only want to match the first three strings, but not the last three strings.

Task	Text
Match	can
Match	man
Match	fan
Skip	dan
Skip	ran
Skip	pan
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	**Only a, b, or c**
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#4 Matching specific characters

Below are a couple lines, where we only want to match the first three strings, but not the last three strings.

Task	Text
Match	**can**
Match	**man**
Match	**fan**
Skip	dan
Skip	ran
Skip	pan
`[cmf]an`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	**Only a, b, or c**
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#5 Excluding specific characters

- In some cases, we might know that there are specific characters that we don't want to match too, for example, we might only want to match phone numbers that are **not** from the area code 650.
- To represent this, we use a similar expression that **excludes specific characters** using the **square brackets** and the **^** (**hat**). For example, the pattern **[^abc]** will match any **single** character **except for** the letters a, b, or c.

#5 Excluding specific characters

Try writing a pattern that matches only the live animals (hog, dog). Notice how most patterns of this type can also be written using the trick from the last example as they are really two sides of the same coin.

Task	Text
Match	hog
Match	dog
Skip	bog
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	**Not a, b, nor c**
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#5 Excluding specific characters

Try writing a pattern that matches only the live animals (hog, dog). Notice how most patterns of this type can also be written using the trick from the last example as they are really two sides of the same coin.

Task	Text
Match	**hog**
Match	**dog**
Skip	bog
`[^b]og`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	**Not a, b, nor c**
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#5 Excluding specific characters

Try writing a pattern that matches only the live animals (hog, dog). Notice how most patterns of this type can also be written using the trick from the last example as they are really two sides of the same coin.

Task	Text
Match	**hog**
Match	**dog**
Skip	bog
`[hd]og`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	**Not a, b, nor c**
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#6 Character ranges

- What if we want to match a character that can be in a **sequential range characters**? Do we have no choice but to list them all out?
- Luckily, when using the square bracket notation, there is a shorthand for matching a character in a list of **sequential characters** by using the **dash** to indicate a character range.
- For example, the pattern **[0-6]** will only match any single digit character from zero to six, and nothing else. And **[^n-p]** will only match any single character **except** for letters n to p.
- Multiple character ranges can also be used in the same set of brackets, along with individual characters. An example of this is the alphanumeric **\w** metacharacter which is equivalent to the character range **[A-Za-z0-9_]** and often used to match characters in English text.

#6 Character ranges

Use the bracket notation to match or skip each character from each line. Be aware that patterns are **case sensitive** and **a-z differs** from **A-Z** in terms of the characters it matches (lower vs upper case).

Task	Text
Match	Ana
Match	Bob
Match	Cpc
Skip	aax
Skip	bby
Skip	ccz
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	**chars a to z**
[0-9]	**Numbers 0 to 9**
\w	**Any alphanumeric char**
\W	**Any non-alphanum. char**
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#6 Character ranges

Use the bracket notation to match or skip each character from each line. Be aware that patterns are **case sensitive** and **a-z differs** from **A-Z** in terms of the characters it matches (lower vs upper case).

Task	Text
Match	**Ana**
Match	**Bob**
Match	**Cpc**
Skip	aax
Skip	bby
Skip	ccz
`[A-C][n-p][a-c]`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	**chars a to z**
[0-9]	**Numbers 0 to 9**
\w	**Any alphanumeric char**
\W	**Any non-alphanum. char**
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#7 Catching some zzz's

- How about the number of **repetitions** of characters that we want to match? One way that we can do this is to explicitly spell out exactly how many characters we want,
  eg. **\d\d\d** which would match exactly three digits.
- A more convenient way is to specify how many repetitions of each character we want using the **curly braces** notation. For example, **a{3}** will match the a character exactly three times. Certain regular expression engines will even allow you to specify a range for this repetition such that **a{1,3}** will match the a character no more than 3 times, but no less than once.
- This quantifier can be used with any character, or special metacharacters, for example **w{3}** (three w's), **[wxy]{5}** (five characters, each of which can be a w, x, or y)
  and **.{2,6}** (between two and six of **any** character).

#7 Catching some zzz's

Try writing a pattern that matches only the first two spellings by using the curly brace notation.

Task	Text
Match	wazzzzzup
Match	wazzzup
Skip	wazup
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	**m repetitions**
{m,n}	**m to n repetitions**
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#7 Catching some zzz's

Try writing a pattern that matches only the first two spellings by using the curly brace notation.

Task	Text
Match	**wazzz**zzup
Match	**wazzz**up
Skip	wazup
`waz{3}`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	**m repetitions**
{m,n}	**m to n repetitions**
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#7 Catching some zzz's

Try writing a pattern that matches only the first two spellings by using the curly brace notation.

Task	Text
Match	**wazzzzzup**
Match	**wazzzup**
Skip	wazup
`waz{3,5}up`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	**m repetitions**
{m,n}	**m to n repetitions**
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#8 Mr. Kleene, Mr. Kleene

- Imagine that you wrote a form that has a donation field that takes a numerical value in dollars. A wealthy user may want to donate $25,000, while a normal user may want to donate $25.
- One way to express such a pattern would be to use what is known as the **Kleene Star** / **Kleene Plus**, which essentially represents either **0 or more** or **1 or more** of the character that it follows (it always follows a character or group).
- To match the donations, we can use the pattern **\d*** to match any number of digits, but a tighter regular expression would be **\d+** which ensures that the input has at least one digit.
- These quantifiers can be used with any character or special metacharacters, for example **a+** (one or more a's), **[abc]+** (one or more of any a, b, or c character) and **.*** (zero or more of **any** character).

#8 Mr. Kleene, Mr. Kleene

Match the strings using both the star and plus metacharacters.

Task	Text
Match	aaaabcc
Match	aabbbbc
Match	aacc
Skip	a
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	**Zero or more repetitions**
+	**One or more repetitions**
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#8 Mr. Kleene, Mr. Kleene

Match the strings using both the star and plus metacharacters.

Task	Text
Match	**aaaabcc**
Match	**aabbbbc**
Match	**aacc**
Skip	a
`aa+b*c+`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	**Zero or more repetitions**
+	**One or more repetitions**
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#8 Mr. Kleene, Mr. Kleene

Match the strings …

Task	Text
Match	**aaaabcc**
Match	**aabbbbc**
Match	**aacc**
Skip	a

```
a{2,4}b{0,4}c{1,2}
```

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	**Zero or more repetitions**
+	**One or more repetitions**
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#9 Characters optional

- Another quantifier that is really common when matching and extracting text is the **?** (question mark) metacharacter which denotes **optionality**. This metacharacter allows you to match either zero or one of the preceding character or group. For example, the pattern **ab?c** will match either the strings "abc" or "ac" because the b is considered optional.
- Similar to the dot metacharacter, the question mark is a special character and you will have to escape it using a slash **\?** to match a plain question mark character in a string.

#9 Characters optional

Try writing a pattern that uses the optionality metacharacter to match only the lines where one or more files were found.

Task	Text
Match	1 file found?
Match	2 files found?
Match	24 files found?
Skip	No files found.
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	**Optional char**
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#9 Characters optional

Try writing a pattern that uses the optionality metacharacter to match only the lines where one or more files were found.

Task	Text
Match	**1 file found?**
Match	**2 files found?**
Match	**24 files found?**
Skip	No files found.

```
\d+ files? found\?
```

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	**Optional char**
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#10 All this whitespace

- When dealing with real-world input, such as log files and even user input, it's difficult not to encounter whitespace. We use it to format pieces of information to make it easier to read and scan visually, and a single space can put a wrench into the simplest regular expression.
- The most common forms of whitespace you will use with regular expressions are the **space** (␣), the **tab** (**\t**), the **new line** (**\n**) and the carriage return (**\r**) (useful in Windows environments), and these special characters match each of their respective whitespaces.
- In addition, a **whitespace** special character **\s** will match **any** of the specific whitespaces above and is extremely useful when dealing with raw input text.

#10 All this whitespace

Try writing a pattern that can match each line regardless of how much whitespace is between the number and the content.

Task	Text
Match	1. abc
Match	2. abc
Match	3. abc
Skip	4.abc
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	**Any whitespace**
\S	**Any non-whitespace char**
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#10 All this whitespace

Try writing a pattern that can match each line regardless of how much whitespace is between the number and the content.

Task	Text
Match	**1. abc**
Match	**2. abc**
Match	**3. abc**
Skip	4.abc
`\d\.\s+abc`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	**Any whitespace**
\S	**Any non-whitespace char**
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#11 Starting and ending

- We wanted to match the word "success" in a log file. We certainly don't want that pattern to match a line that says "Error: unsuccessful operation"! That is why it is often **best practice to write as specific regular expressions as possible** to ensure that we don't get false positives.
- One way to tighten our patterns is to define a pattern that describes both the **start and the end of the line** using the special **^** (**hat**) and **$** (**dollar sign**) metacharacters. In the next example, we can use the pattern **^success** to match **only** a line that begins with the word "success".
- Note that this is different from the hat used inside a set of bracket **[^...]** for excluding characters, which can be confusing when reading regular expressions.

#11 Starting and ending

Try to match each of the strings below using these new special characters.

Task	Text
Match	Mission: successful
Skip	Last Mission: unsuccessful
Skip	Next Mission: successful upon capture of target
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	**Starts and ends**
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#11 Starting and ending

Try to match each of the strings below using these new special characters.

Task	Text
Match	**Mission: successful**
Skip	Last Mission: unsuccessful
Skip	Next Mission: successful upon capture of target
`^Mission: successful$`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	**Starts and ends**
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#12 Match groups

- Regular expressions allow us to not just match text but also to **extract information for further processing**. This is done by defining **groups of characters** and capturing them using the special parentheses **(** and **)** metacharacters.
- Any subpattern inside a pair of parentheses will be **captured** as a group. In practice, this can be used to extract information like phone numbers or emails from all sorts of data.
- Imagine for example that you had a command line tool to list all the image files you have in the cloud. You could then use a pattern such as **^(IMG\d+\.png)$** to capture and extract the full filename, but if you only wanted to capture the filename without the extension, you could use the pattern **^(IMG\d+)\.png$** which only captures the part before the period.

#12 Match groups

Try to use this to write a regular expression that matches only the filenames (not including extension) of the PDF files below.

Task	Text
Capture	file_record_transcript.pdf
Capture	file_07241999.pdf
Skip	testfile_fake.pdf.tmp
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	**Capture group**
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#12 Match groups

Try to use this to write a regular expression that matches only the filenames (not including extension) of the PDF files below.

Task	Text
Capture	**file_record_transcript.pdf**
Capture	**file_07241999.pdf**
Skip	testfile_fake.pdf.tmp
`^(file.+)\.pdf$`	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	**Capture group**
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

#13 It's all conditional

- It's always good to **be precise**. For example, you wouldn't write a grocery list for someone to **Buy more .*** because you would have no idea what you could get back. Instead you would write **Buy more milk** or **Buy more bread**, …
- Specifically when using groups, you can use the **|** (**logical OR, aka. the pipe**) to denote **different possible sets of characters**. In the above example, I can write the pattern "Buy more (milk|bread|juice)" to match only the strings Buy more milk, Buy more bread, or Buy more juice.
- Like normal groups, you can use any sequence of characters or metacharacters in a condition, for example, ([cb]ats*|[dh]ogs?) would match either cats or bats, or, dogs or hogs. Writing patterns with many conditions can be hard to read, so you should consider making them separate patterns if they get too complex.

#13 It's all conditional

Try writing a conditional pattern that matches only the lines with small fuzzy creatures below.

Task	Text
Match	I love cats
Match	I love dogs
Skip	I love logs
Skip	I love cogs
?	

abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	**Matches abc or def**

#13 It's all conditional

Try writing a conditional pattern that matches only the lines with small fuzzy creatures below.

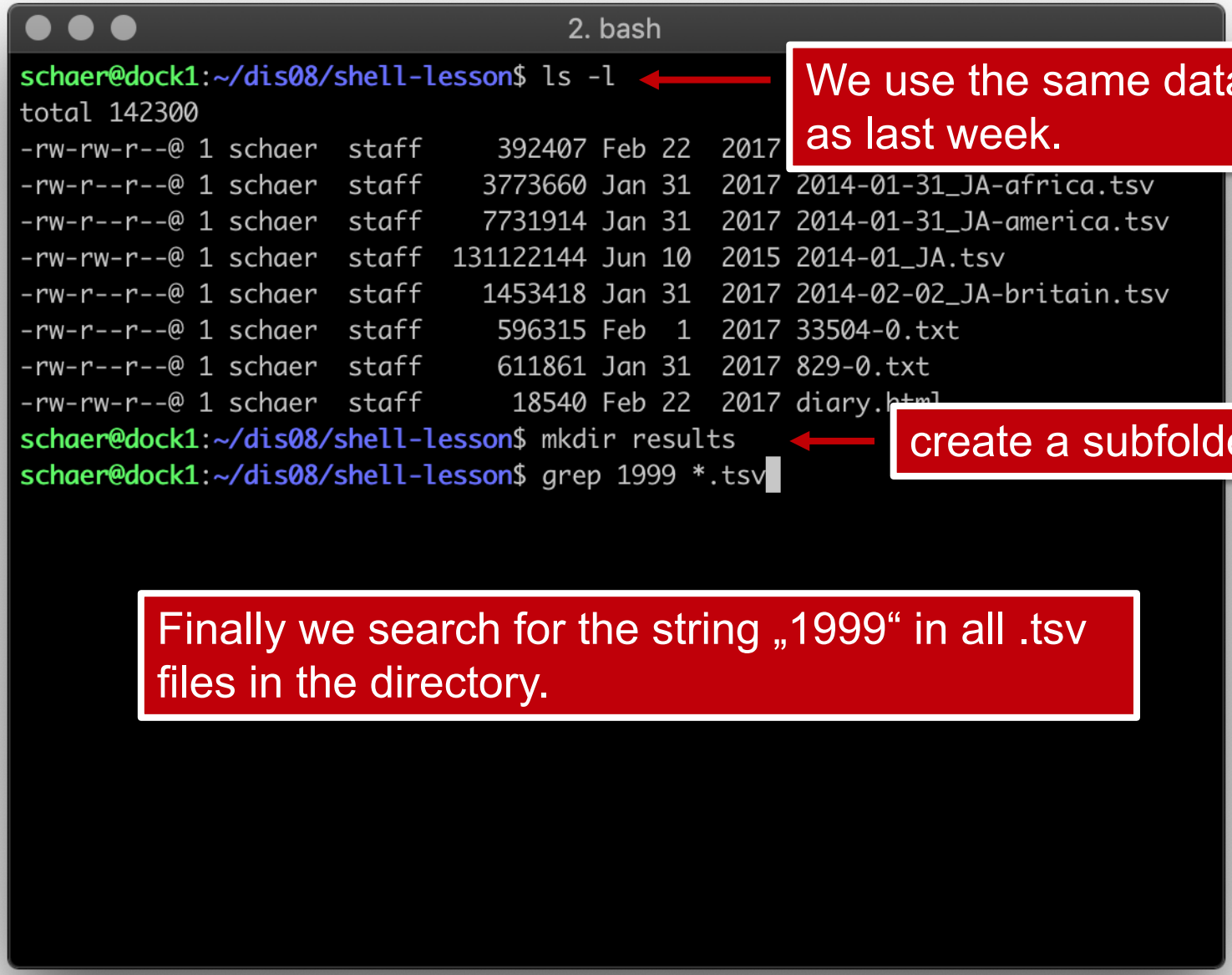
Task	Text
Match	**I love cats**
Match	**I love dogs**
Skip	I love logs
Skip	I love cogs

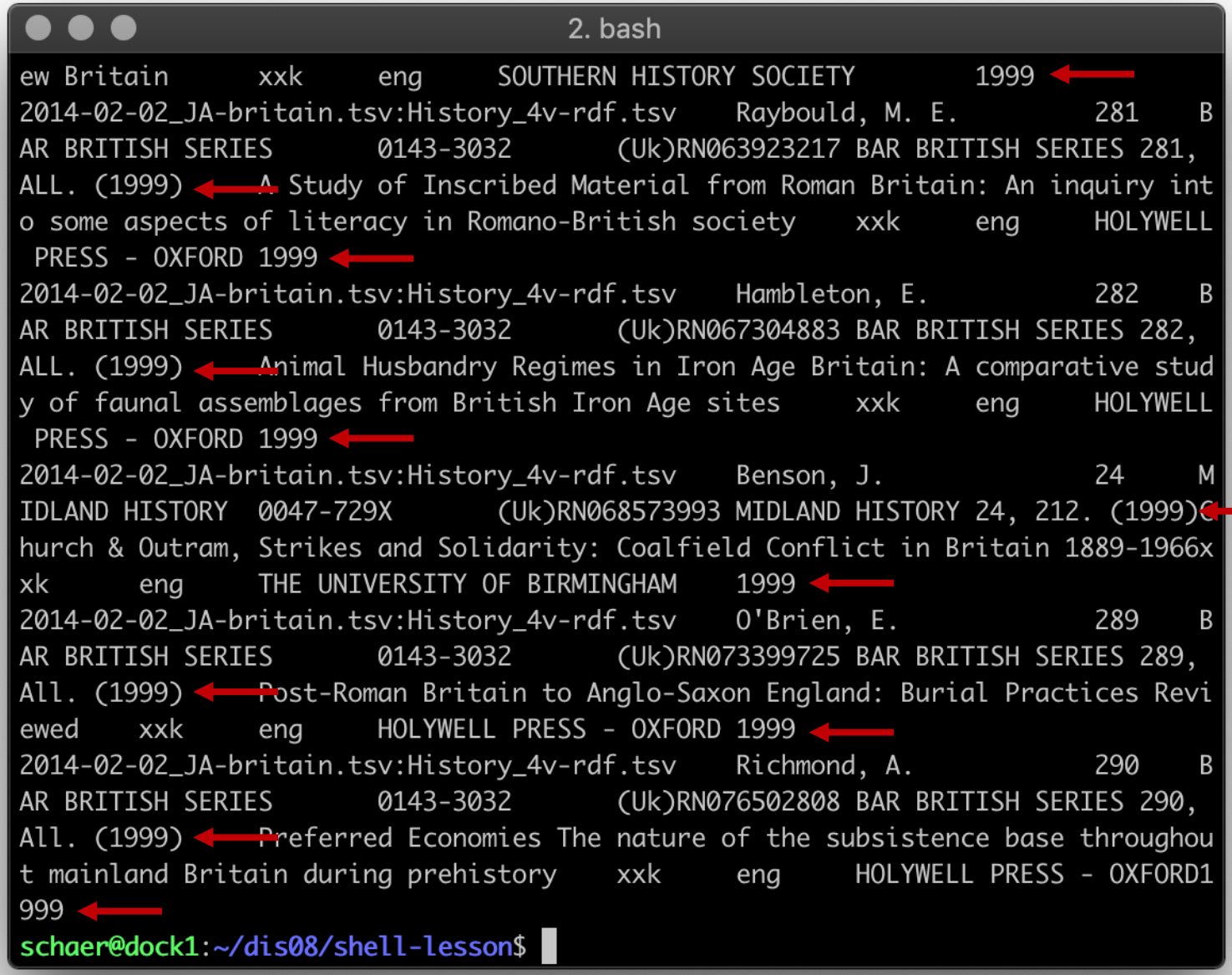
```
I love (cats|dogs)
```

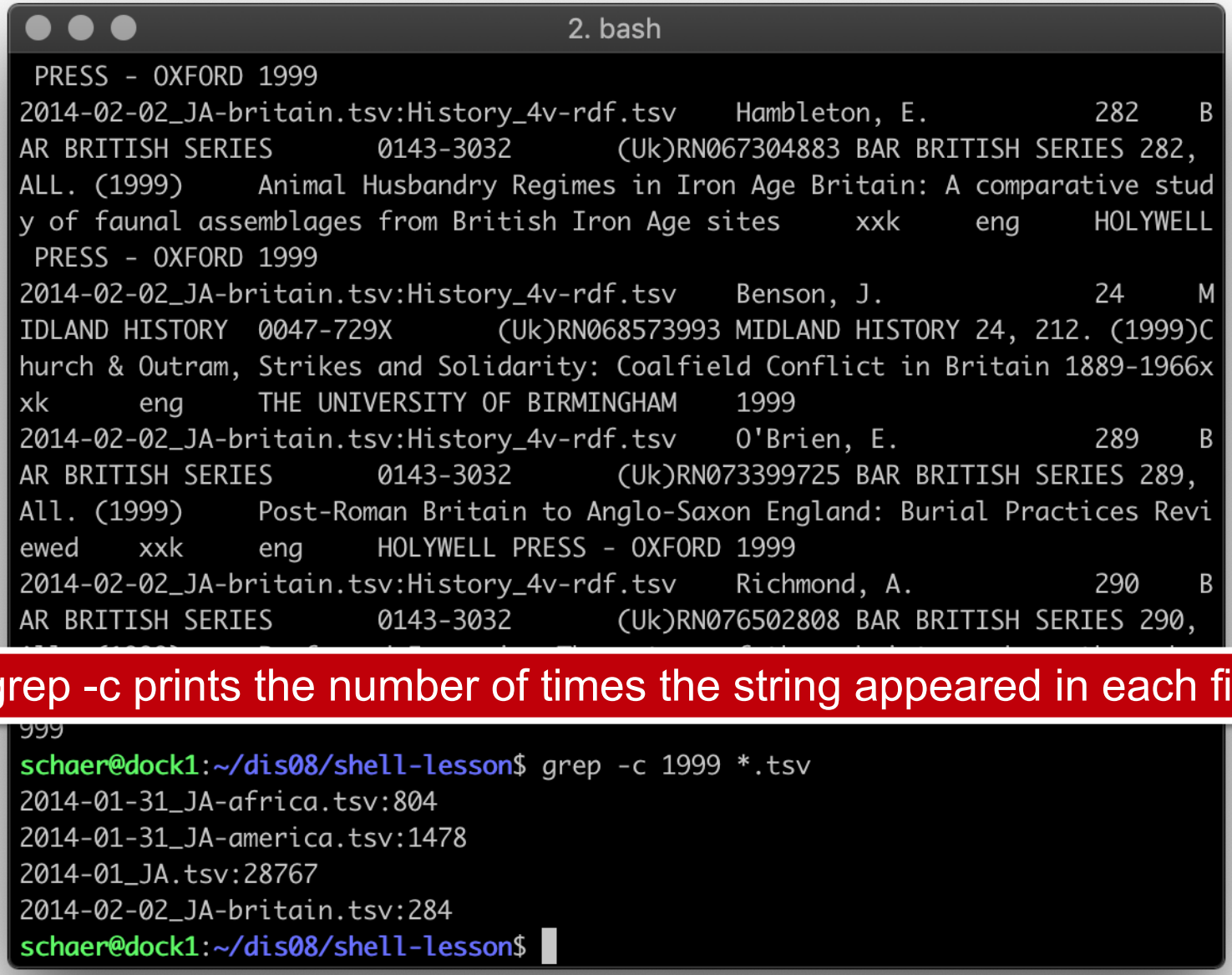
abc…	Letters
123…	Digits
\d	Any digit
\D	Any non-digit char
.	Any char
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	chars a to z
[0-9]	Numbers 0 to 9
\w	Any alphanumeric char
\W	Any non-alphanum. char
{m}	m repetitions
{m,n}	m to n repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional char
\s	Any whitespace
\S	Any non-whitespace char
^…$	Starts and ends
(…)	Capture group
(a(bc))	Capture sub-group
(.*)	Capture all
(abc|def)	**Matches abc or def**

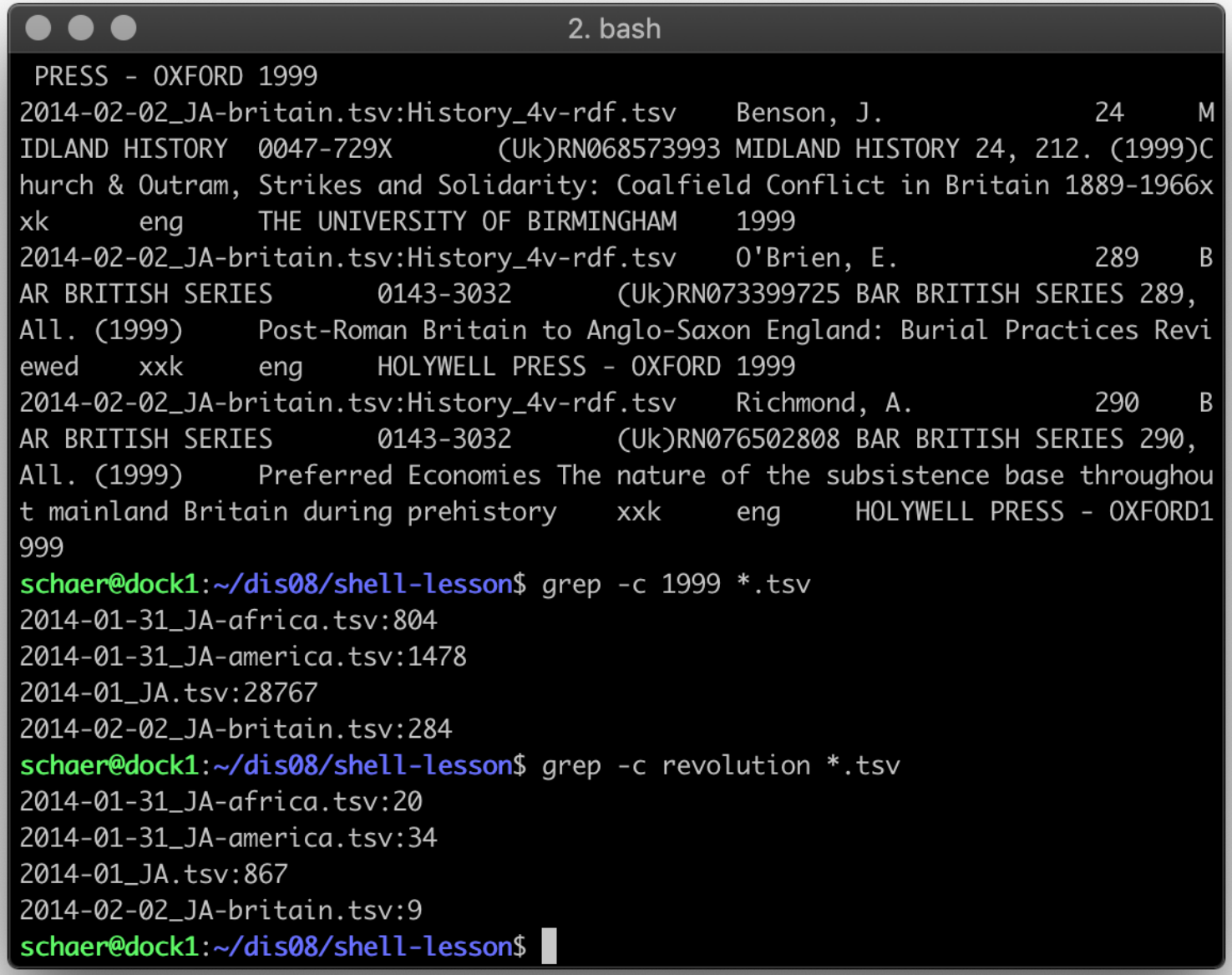
Using regex in the bash

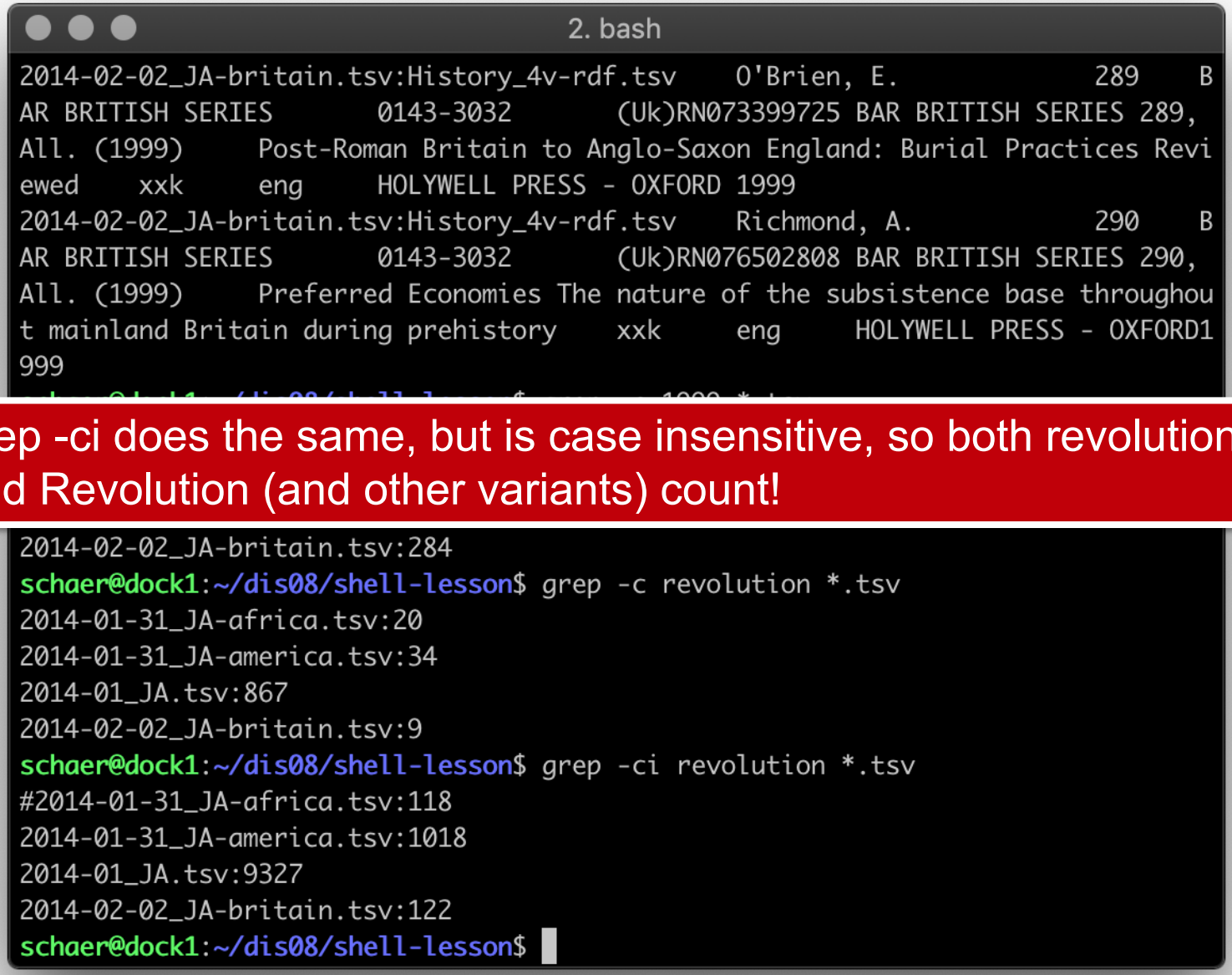
- Searching for something in one or more files is something we'll often need to do, so let's introduce a command for doing that: **`grep`** (short for **global regular expression print**).
- As the name suggests, it supports regular expressions and is therefore only limited by your imagination, the shape of your data, and - when working with thousands or millions of files - the processing power at your disposal.

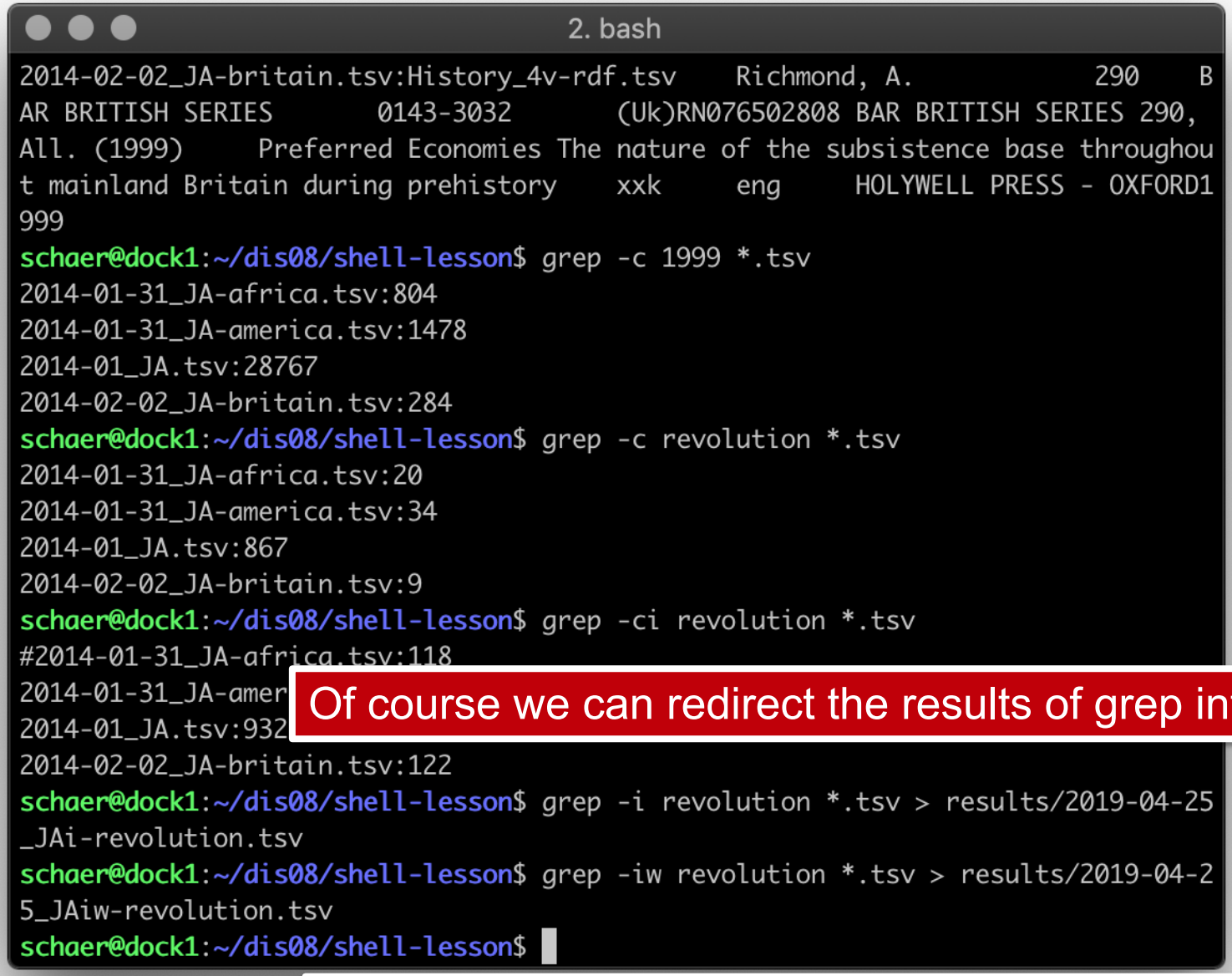

2. bash
schaer@dock1:~/dis08/shell-lesson$ ls -l
total 142300
-rw-rw-r--@ 1 schaer staff 392407 Feb 22 2017
-rw-r--r--@ 1 schaer staff 3773660 Jan 31 2017 2014-01-31_JA-africa.tsv
-rw-r--r--@ 1 schaer staff 7731914 Jan 31 2017 2014-01-31_JA-america.tsv
-rw-rw-r--@ 1 schaer staff 131122144 Jun 10 2015 2014-01_JA.tsv
-rw-r--r--@ 1 schaer staff 1453418 Jan 31 2017 2014-02-02_JA-britain.tsv
-rw-r--r--@ 1 schaer staff 596315 Feb 1 2017 33504-0.txt
-rw-r--r--@ 1 schaer staff 611861 Jan 31 2017 829-0.txt
-rw-rw-r--@ 1 schaer staff 18540 Feb 22 2017 diary.html
schaer@dock1:~/dis08/shell-lesson$ mkdir results
schaer@dock1:~/dis08/shell-lesson$ grep 1999 *.tsv
We use the same data set as last week.
create a subfolder
Finally we search for the string „1999“ in all .tsv files in the directory.


2. bash
ew Britain xxk eng SOUTHERN HISTORY SOCIETY 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Raybould, M. E. 281 B
AR BRITISH SERIES 0143-3032 (Uk)RN063923217 BAR BRITISH SERIES 281,
ALL. (1999) A Study of Inscribed Material from Roman Britain: An inquiry int
o some aspects of literacy in Romano-British society xxk eng HOLYWELL
PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Hambleton, E. 282 B
AR BRITISH SERIES 0143-3032 (Uk)RN067304883 BAR BRITISH SERIES 282,
ALL. (1999) Animal Husbandry Regimes in Iron Age Britain: A comparative stud
y of faunal assemblages from British Iron Age sites xxk eng HOLYWELL
PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Benson, J. 24 M
IDLAND HISTORY 0047-729X (Uk)RN068573993 MIDLAND HISTORY 24, 212. (1999)C
hurch & Outram, Strikes and Solidarity: Coalfield Conflict in Britain 1889-1966x
xk eng THE UNIVERSITY OF BIRMINGHAM 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv O'Brien, E. 289 B
AR BRITISH SERIES 0143-3032 (Uk)RN073399725 BAR BRITISH SERIES 289,
All. (1999) Post-Roman Britain to Anglo-Saxon England: Burial Practices Revi
ewed xxk eng HOLYWELL PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Richmond, A. 290 B
AR BRITISH SERIES 0143-3032 (Uk)RN076502808 BAR BRITISH SERIES 290,
All. (1999) Preferred Economies The nature of the subsistence base throughou
t mainland Britain during prehistory xxk eng HOLYWELL PRESS - OXFORD1
999
schaer@dock1:~/dis08/shell-lesson$


2. bash
PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Hambleton, E. 282 B
AR BRITISH SERIES 0143-3032 (Uk)RN067304883 BAR BRITISH SERIES 282,
ALL. (1999) Animal Husbandry Regimes in Iron Age Britain: A comparative stud
y of faunal assemblages from British Iron Age sites xxk eng HOLYWELL
PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Benson, J. 24 M
IDLAND HISTORY 0047-729X (Uk)RN068573993 MIDLAND HISTORY 24, 212. (1999)C
hurch & Outram, Strikes and Solidarity: Coalfield Conflict in Britain 1889-1966x
xk eng THE UNIVERSITY OF BIRMINGHAM 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv O'Brien, E. 289 B
AR BRITISH SERIES 0143-3032 (Uk)RN073399725 BAR BRITISH SERIES 289,
All. (1999) Post-Roman Britain to Anglo-Saxon England: Burial Practices Revi
ewed xxk eng HOLYWELL PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Richmond, A. 290 B
AR BRITISH SERIES 0143-3032 (Uk)RN076502808 BAR BRITISH SERIES 290,
grep -c prints the number of times the string appeared in each file
schaer@dock1:~/dis08/shell-lesson$ grep -c 1999 *.tsv
2014-01-31_JA-africa.tsv:804
2014-01-31_JA-america.tsv:1478
2014-01_JA.tsv:28767
2014-02-02_JA-britain.tsv:284
schaer@dock1:~/dis08/shell-lesson$

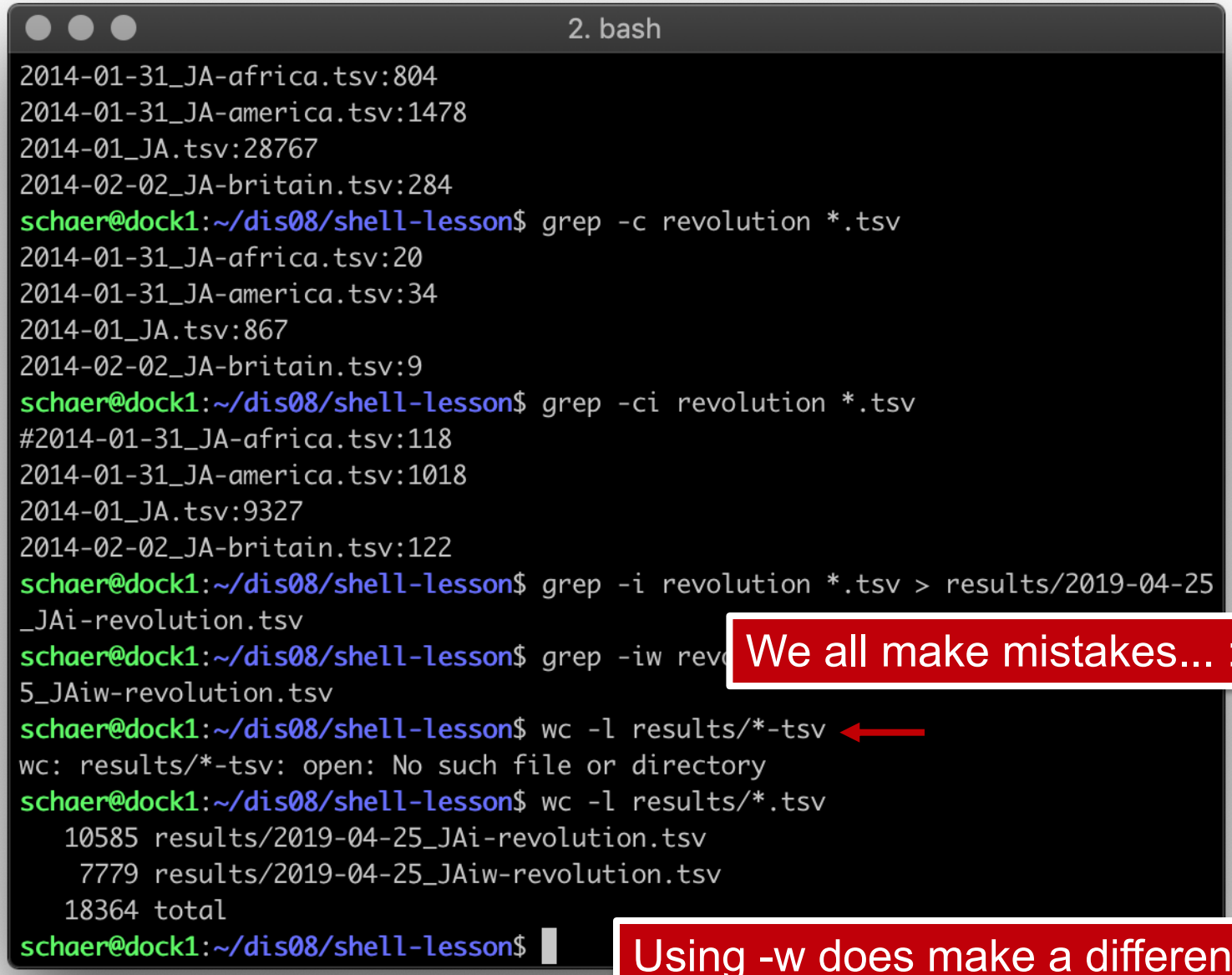

2. bash
PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Benson, J. 24 M
IDLAND HISTORY 0047-729X (Uk)RN068573993 MIDLAND HISTORY 24, 212. (1999)C
hurch & Outram, Strikes and Solidarity: Coalfield Conflict in Britain 1889-1966x
xk eng THE UNIVERSITY OF BIRMINGHAM 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv O'Brien, E. 289 B
AR BRITISH SERIES 0143-3032 (Uk)RN073399725 BAR BRITISH SERIES 289,
All. (1999) Post-Roman Britain to Anglo-Saxon England: Burial Practices Revi
ewed xxk eng HOLYWELL PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Richmond, A. 290 B
AR BRITISH SERIES 0143-3032 (Uk)RN076502808 BAR BRITISH SERIES 290,
All. (1999) Preferred Economies The nature of the subsistence base throughou
t mainland Britain during prehistory xxk eng HOLYWELL PRESS - OXFORD1
999
schaer@dock1:~/dis08/shell-lesson$ grep -c 1999 *.tsv
2014-01-31_JA-africa.tsv:804
2014-01-31_JA-america.tsv:1478
2014-01_JA.tsv:28767
2014-02-02_JA-britain.tsv:284
schaer@dock1:~/dis08/shell-lesson$ grep -c revolution *.tsv
2014-01-31_JA-africa.tsv:20
2014-01-31_JA-america.tsv:34
2014-01_JA.tsv:867
2014-02-02_JA-britain.tsv:9
schaer@dock1:~/dis08/shell-lesson$


2. bash
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv O'Brien, E. 289 B
AR BRITISH SERIES 0143-3032 (Uk)RN073399725 BAR BRITISH SERIES 289,
All. (1999) Post-Roman Britain to Anglo-Saxon England: Burial Practices Revi
ewed xxk eng HOLYWELL PRESS - OXFORD 1999
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Richmond, A. 290 B
AR BRITISH SERIES 0143-3032 (Uk)RN076502808 BAR BRITISH SERIES 290,
All. (1999) Preferred Economies The nature of the subsistence base throughou
t mainland Britain during prehistory xxk eng HOLYWELL PRESS - OXFORD1
999
grep -ci does the same, but is case insensitive, so both revolution and Revolution (and other variants) count!
2014-02-02_JA-britain.tsv:284
schaer@dock1:~/dis08/shell-lesson$ grep -c revolution *.tsv
2014-01-31_JA-africa.tsv:20
2014-01-31_JA-america.tsv:34
2014-01_JA.tsv:867
2014-02-02_JA-britain.tsv:9
schaer@dock1:~/dis08/shell-lesson$ grep -ci revolution *.tsv
#2014-01-31_JA-africa.tsv:118
2014-01-31_JA-america.tsv:1018
2014-01_JA.tsv:9327
2014-02-02_JA-britain.tsv:122
schaer@dock1:~/dis08/shell-lesson$



Of course we can redirect the results of grep into files!

grep -w searches for whole words only. So we don‘t mix up revolution and revolutionary (and others).

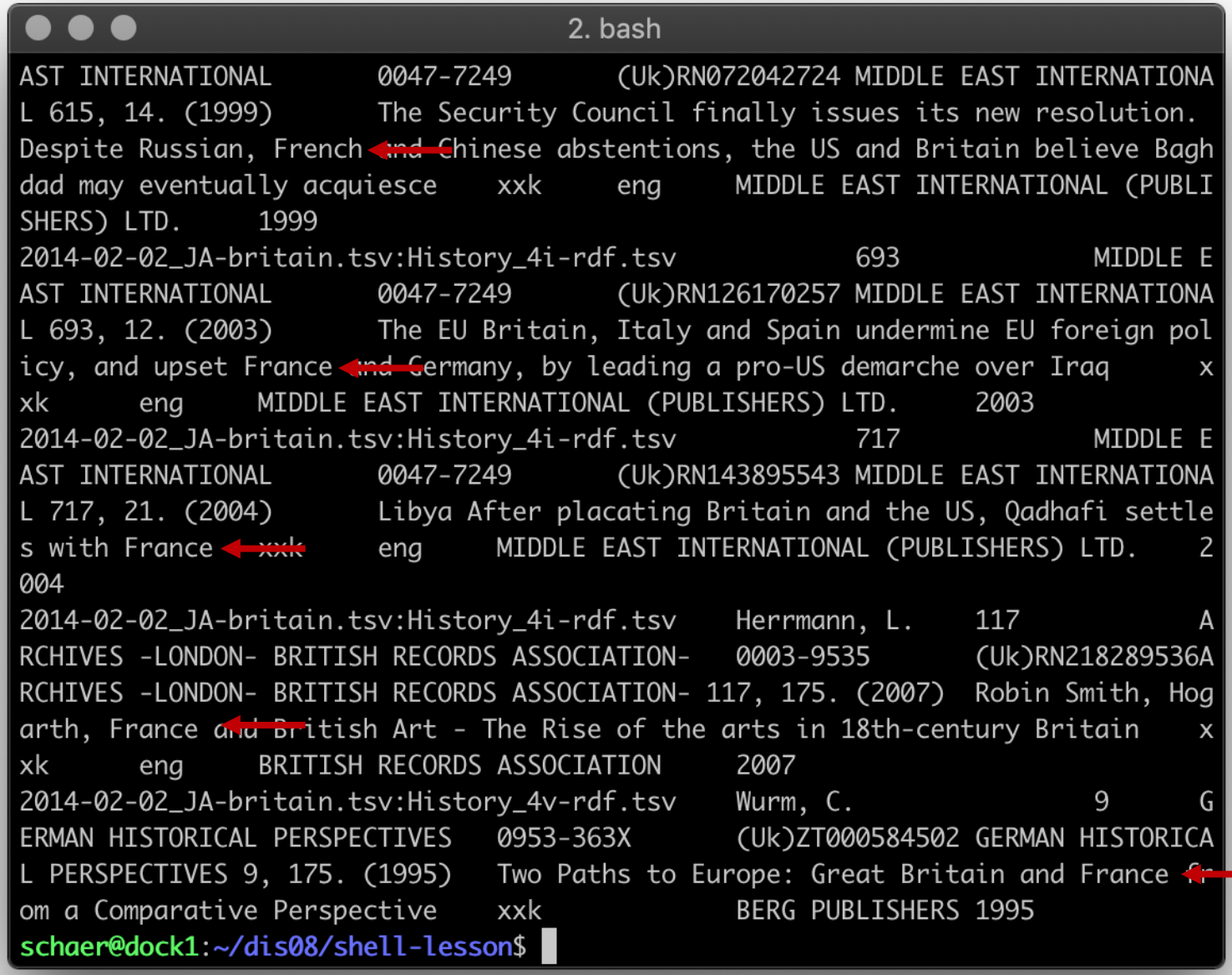

2. bash
2014-01-31_JA-africa.tsv:804
2014-01-31_JA-america.tsv:1478
2014-01_JA.tsv:28767
2014-02-02_JA-britain.tsv:284
schaer@dock1:~/dis08/shell-lesson$ grep -c revolution *.tsv
2014-01-31_JA-africa.tsv:20
2014-01-31_JA-america.tsv:34
2014-01_JA.tsv:867
2014-02-02_JA-britain.tsv:9
schaer@dock1:~/dis08/shell-lesson$ grep -ci revolution *.tsv
#2014-01-31_JA-africa.tsv:118
2014-01-31_JA-america.tsv:1018
2014-01_JA.tsv:9327
2014-02-02_JA-britain.tsv:122
schaer@dock1:~/dis08/shell-lesson$ grep -i revolution *.tsv > results/2019-04-25
_JAi-revolution.tsv
schaer@dock1:~/dis08/shell-lesson$ grep -iw rev
5_JAiw-revolution.tsv
schaer@dock1:~/dis08/shell-lesson$ wc -l results/*-tsv
wc: results/*-tsv: open: No such file or directory
schaer@dock1:~/dis08/shell-lesson$ wc -l results/*.tsv
10585 results/2019-04-25_JAi-revolution.tsv
7779 results/2019-04-25_JAiw-revolution.tsv
18364 total
schaer@dock1:~/dis08/shell-lesson$
We all make mistakes... :-)
Using -w does make a difference!

Basic and extended regex in grep

- There is unfortunately both “**basic**” and “**extended**” regular expressions.
- In **basic regular expressions** the meta-characters ‘?’, ‘+’, ‘{’, ‘|’, ‘(’, and ‘)’ lose their special meaning.
- This is a cause of **confusion**, since most tutorials, teach extended regular expression, but grep uses basic by default.
- Make your life easy by always using extended regular expressions (`-E` flag) when doing something more complex than searching for a plain string! Or simply use **`egrep`**.

https://www.gnu.org/software/grep/manual/html_node/Basic-vs-Extended.html


2. bash
schaer@dock1:~/dis08/shell-lesson$ wc -l results/*.tsv
10585 results/2019-04-25_JAi-revolution.tsv
7779 results/2019-04-25_JAiw-revolution.tsv
18364 total
schaer@dock1:~/dis08/shell-lesson$ grep -iwE 'fr[ae]nc[eh]' *.tsv
‘fr[ae]nc[eh]’ will match “france”, “french”, but also “frence” and “franch”…


2. bash
AST INTERNATIONAL 0047-7249 (Uk)RN072042724 MIDDLE EAST INTERNATIONA
L 615, 14. (1999) The Security Council finally issues its new resolution.
Despite Russian, French and Chinese abstentions, the US and Britain believe Bagh
dad may eventually acquiesce xxk eng MIDDLE EAST INTERNATIONAL (PUBLI
SHERS) LTD. 1999
2014-02-02_JA-britain.tsv:History_4i-rdf.tsv 693 MIDDLE E
AST INTERNATIONAL 0047-7249 (Uk)RN126170257 MIDDLE EAST INTERNATIONA
L 693, 12. (2003) The EU Britain, Italy and Spain undermine EU foreign pol
icy, and upset France and Germany, by leading a pro-US demarche over Iraq x
xk eng MIDDLE EAST INTERNATIONAL (PUBLISHERS) LTD. 2003
2014-02-02_JA-britain.tsv:History_4i-rdf.tsv 717 MIDDLE E
AST INTERNATIONAL 0047-7249 (Uk)RN143895543 MIDDLE EAST INTERNATIONA
L 717, 21. (2004) Libya After placating Britain and the US, Qadhafi settle
s with France xxk eng MIDDLE EAST INTERNATIONAL (PUBLISHERS) LTD. 2
004
2014-02-02_JA-britain.tsv:History_4i-rdf.tsv Herrmann, L. 117 A
RCHIVES -LONDON- BRITISH RECORDS ASSOCIATION- 0003-9535 (Uk)RN218289536A
RCHIVES -LONDON- BRITISH RECORDS ASSOCIATION- 117, 175. (2007) Robin Smith, Hog
arth, France and British Art - The Rise of the arts in 18th-century Britain x
xk eng BRITISH RECORDS ASSOCIATION 2007
2014-02-02_JA-britain.tsv:History_4v-rdf.tsv Wurm, C. 9 G
ERMAN HISTORICAL PERSPECTIVES 0953-363X (Uk)ZT000584502 GERMAN HISTORICA
L PERSPECTIVES 9, 175. (1995) Two Paths to Europe: Great Britain and France fr
om a Comparative Perspective xxk BERG PUBLISHERS 1995
schaer@dock1:~/dis08/shell-lesson$

What can you do next...?

- Use regular expressions to find all ISSN numbers (four digits followed by hyphen followed by four digits) in the file `2014-01_JA.tsv` and print the results to a file `results/issns.tsv`.
- Find all the ISSN numbers and give an overview over the unique values (using the `uniq` command).
- Count the unique ISSN numbers.
- Do it all in one line of shell code...