File & Regular Expressions

Filters

- These a programs that takes some input and transforms it
- Examples of which are wc, grep, more, sort
- The input for some key filter programs take the form of a string pattern along with a data file, grep is one of those
- String patterns can be defined using regular expressions

Bash Filename expansion

bash meta character definitions

* match any characters

? match any single character

[begin a character group

] end a character group

- denotes a character range

! negate a character group

| logical "or" between patterns

?(*pattern*) match zero or one instance of *pattern*

*(*pattern*) match zero or more instances of *pattern*

+(*pattern*) match one or more instances of *pattern*

@(*pattern*) match exactly one instance of *pattern*

!(*pattern*) match any strings that don't contain *pattern*

Bash Filename expansion. The ? Special Character

The question mark ? matches any single character.

Consider a local directory containing five files named

tut1.txt, tut2.txt tut3.txt, abc.txt, tutorial

If one types
% ls tut?.txt

The following output will be generated

tut1.txt tut2.txt tut3.txt

The ? special character will not match the leading period . which hidden files contain.

Bash Filename expansion. The * Special Character

The asterisk special character matches zero or more characters (excluding the leading period found in hidden files).

Example:

```
% ls
a file1 file2 fileaBB fileaaa tt
% ls file*
file1 file2 fileaBB fileaaa
%ls f*le1
file1
```

Bash Filename expansion. The [] Special Characters

The square brackets [] can be used to suggest to the shell to match all filenames containing the characters within the brackets.

Consider a local directory containing five files named

tut1.txt, tut2.txt tut3.txt, abc.txt, tutorial edd.txt

% ls [aei]* will yield the following

abc.txt edd.txt

Bash Filename expansion. The [] Special Characters

The square brackets can specify a range of characters.

Example:
Display the contents of the files part1.txt, part2.txt, part3.txt.
% cat part[1-3].txt

If in addition to the above files, the file part5.txt needs to be displayed:
% cat part[1-35] .txt
This selects the characters in the range 1–3 and also 5.

Example:
List all of the files whose extensions end with
a single character (a through to k).

% ls *·*[a-k]

Bash Filename expansion. The [] Special Characters

Example:
Consider a local directory containing the following files
test01, test02, test03, test11, test12, test13, test21, test22, test23, test 24, test_1

To list all of the files whose names start with the string test and they are followed by a number in the range 0–2 and then followed by a number in the range 1–3 one types

% ls test[0-2][1-3]

The above will select the files test01, test02, test03, test11, test12, test13, test21, test22, test23

It has not selected test 24 or test_1.

Bash Filename expansion. +(*pattern*)

- Match one or more of pattern defined
- Consider files in local folder

1, 1b, 3, 123, 1bb, 4, 1a34 , 2, 9b3, F1, file2, new, a1a6a9aa

- > ls +([0-9a])

  1 123 1a34 2 3 4 a1a66a9aa

Bash Filename expansion. ?(*pattern*)

- Match zero or one of pattern defined
- Consider files in local folder

1, 1b, 3, 123, 1bb, 4, 1a34 , 2, 9b3, F1, file2, new, a1a6a9aa, 1a, 34

- > ls ?([0-9a])

  1 2 3 4

- > ls ?([0-9a])[0-9]

  1 2 3 34 4

Bash Filename expansion. ?(*pattern*)

- Match zero or one of pattern defined
- Consider files in local folder

1, 1b, 3, 123, 1bb, 4, 1a34 , 2, 9b3, F1, file2, new, a1a6a9aa, 1a, 34

- > ls ?([0-9a])[a-z]

  1a 1b --- the last character is a lower case letter

- >ls ?([0-9a])?[a-z] -- match 0 or 1 either 0,1,2,3,4,5,6,7,8,9,a
  then match any single character
  then match one lower case letter character

  1a 1b 1bb
- > ?([0-9a])?([a-z]) -- match 0 or 1 char from either 0,1,2,3,4,5,6,7,8,9,a
  match 0 or 1 char from either a to z

  1 1a 1b 2 3 4

Bash Filename expansion. ![character_group]

- Match files that do not contain a character group
- Consider files in local folder

1, 1b, 3, 123, 1bb, 4, 1a34 , 2, 9b3, F1, file2, new, a1a6a9aa, 1a, 34

- Example
- > ls [!13]*
- Pick up all files that don’t begin with a 1 or 3
- 4, 2, 9b3, F1, file2, new, a1a6a9aa

Bash Filename expansion. @(*pattern*) and | logical or

@(*pattern*) match exactly one instance of *pattern*

| logical or operator usage pattern1|pattern2

Example

ls prog*([0-9])@(.f|.c)

lists all files which start with prog, followed by zero or more digits, followed by .f or .c

Will match: prog.f   prog123.c

Will **not** match prog12345

Brace expansion {}

- Similar to filename expansion
- Patterns to be expanded take the form

[optionalstring] {string, string,..} [optionalstring]

A{d,c,b}e yields

Ade Ace Abe

Brace expansion {}

- A sequence expression takes the form {x..y[..incr]}, where x and y are either integers or single characters, and incr, an optional increment, is an integer
- x, y can represent range of numbers eg. {2..10..2} yields
- 2 4 6 8 10
- integers may be prefixed with '0' to force each term to have the same width. The shell attempts to force all generated terms to contain the same number of digits using '0' padding
- Example {95 .. 1000..100} yields

  95 195 295 395 495 595 695 795 895 995
- Example {095 .. 1000..100} yields

  0095 0195 0295 0395 0495 0595 0695 0795 0895 0995
- Example {95 .. 01000..100} yields

  00095 00195 00295 00395 00495 00595 00695 00795 00895 00995

Brace expansion {}

- When characters are supplied, the expression expands to each character lexicographically between x and y
- Note that both x and y must be of the same type
- When the increment is supplied, it is used as the difference between each term. The default increment is 1 or -1 as appropriate.
- Example {a..g..2} yields a c e g

Regular Expressions

- Regular expressions are different to file extensions

- There are also 2 versions of Regular Expression
  1)standard and
  2) extended.
- **Metacharacters**
- Metacharacters can be used in combination with part of a string,
  to make the shell select for example a list of existing files
- When one of these characters appears in an argument on the command line, the shell expands that argument into a list of filenames and passes the list to the program that the command line is calling.

Regular Expressions

- Regular expressions are made of
- 1. Anchor –species the position of the string pattern with respect to the string that is compared e.g. ^ - beginning of the string, $ end of the string
- 2. character sets – definition of one or more characters e.g. [a-z] defines range of lower case alphabet characters
- 3. modifiers – special characters that define number of times the character set is repeated, + defining 1 or more times

Meta characters - operators

operator	Type	Definition
.	Character set	Match a single character
?	Modifier	Item just before operator – optional matched maximum of 1 times
*	Modifier	Item just before operator matched 0 or more times
+	Modifier	Item before operator matched 1 or more times
{N}	Modifier	Item before operator matched N times
{N,}	Modifier	Item before operator matched N or more times
{N,M}	Modifier	Item before operator matched a min of N times and a max of M times
-	Character set	Range (must not be the 1st or last in list)
|	OR function	OR compares two strings
^	Anchor Character set	Matches empty string at the beginning of a line Also represents NOT in the range
$	Anchor	Matches the empty string at the end of a line
\b	Anchor	Matches empty string at the edge of word
\B	Anchor	Matches empty string NOT at the edge of word
\<	Anchor	Match empty string at beginning of word
\>	Anchor	Match empty string at the end of word

Regular expressions in bash

- bash has a built-in regular expression comparison operator, represented by =~
- Usage
- String =~ regular expression

  The results if a match is made will be true, otherwise false

Character classes

- Can be defined syntax [:class:]
- Where class defined

Class	Definition
alnum	alpha-numeric
alpha	A-z, A-Z
ascii	Ascii
blank	Blank space
digit	number
lower	Lower case char
space	Blank space
upper	Upper case characters

These can be used to define a regular expressions, thus avoid you generating/defining large lists

Regular expressions

- If we wish to test for characters that are used as meta characters e.g. ?
- To identify it's a character and not a meta-character we include the back slash \ followed by the character

character	Regular expression match
.	\.
?	\?
*	*
+	\+
-	\-
^	\^
$	\$
{	\{

Regular expressions – examples using Bash

- Reg expression: ^\^[0-9a-dA-C]*$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
^	\^	[	0-9	a-d	A-C	]	*	$							

Item position 0: ^ start string comparison from the beginning so
Item pos 1: The 1st char is ^ that the string must have a ^ at the bebinning

Item positions 2 to 6: definition in square brackets – look for a single character
Item Positions 3 to 5: has list of characters defined
Item pos 3: 0-9 : char digits 0 or 1 or 2 .. Or 9
Item pos 4: a-d: chars a or b or c or d
Item pos 5: A-C: chars A or B or C
From this list only one needs to match

Regular expressions - examples using Bash

- Reg expression: ^\^[0-9a-dA-C]*$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
^	\^	[	0-9	a-d	A-C	]	*	$							

item pos 7: *: this will act on previous item pos 2 to 9 and match this 0 or more times

Item pos 8: $ matches till the end of the line so last char has to be either a digit, or one of a,b,c,d,A,B,C

Examples

^12abd5689ABC	match
1234	No match
^	match
^^^	No match

Regular expressions - examples using Bash

- Reg expression: ^^[0-9a-dA-C]*$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
^	^	[	0-9	a-d	A-C	]	*	$							

Here item pos 1 has been changed from \^ to just the character ^

Example inputs
^1AD No Match
^1Ad No Match
1Ad Match
^ No Match

Regular expressions – examples using Bash

- Reg expression: ^[0-9][a-dA-C]+$

0	1	2	3	4	5	6	7
^	[0-9]	[a-dA-C]	+	$			

Item pos 0: ^ start string comparison from the beginning so
Item pos 1: The 1st char is ^ position 1 that the string must have is any numeric digit

Item pos 2: definition in square brackets – look for a single character
a-d: chars a or b or c or d OR
A-C: chars A or B or C
From this list only one character to match

Item pos 3: + this will act on previous item definition pos 2 and match this 1 or more times

Item pos 4: $ matches till the end of the line item pos 2, so last char either a,b,c,d,A,B,C

Regular expressions – examples using Bash

- Reg expression: ^[0-9][a-dA-C]+$

0	1	2	3	4	5	6	7
^	[0-9]	[a-dA-C]	+	$			

^45aA No match
45aA No match
4aAaA match
aaaa No match
2a match
2a4a No match
2 No match

Regular expressions – examples using Bash

- Reg expression: ^*[0-9].([0-9]\/)+$

0	1	2	3	4	5	6
^	*	[0-9]	.	([0-9]\/)	+	$

1st char position 0 ^ start string comparison from the beginning so
The 1st char is * (item pos 1) that the string must start with

Item pos 2: [0-9] match a single digit defined in square brackets as a range 0 to 9

Item pos 3: . Match any single character

Item pos 4: ([0-9]\/) grouped in circular brackets as an item defining a single digit [0-9] (legal values 0 to 9)
Then \/ backslash followed by forward slash – define / forward slash

Regular expressions – examples using Bash

- Reg expression: ^*[0-9].([0-9]\/)+$

0	1	2	3	4	5	6
^	*	[0-9]	.	([0-9]\/)	+	$

Item pos 4: ([0-9]\/) grouped in circular brackets as an item defining a single digit [0-9] (legal values 0 to 9)
Then \/ backslash followed by forward slash – define / forward slash

Item pos 5: defines one or more of the item to the left i.e. digit followed by the forward slash

Regular expressions – examples using Bash

- Reg expression: ^*[0-9].([0-9]\/)+$

0	1	2	3	4	5	6
^	*	[0-9]	.	([0-9]\/)	+	$

*4u7/8/	Match
op9/	No match
*3D77/7/	No match
*4a0/1/2/3/4/6/	Match

Regular expressions – examples using Bash

- Reg expression: ^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$

0	1	2	3	4	5	6	7	8	9	10	11	12
^	[0-9]	{2}	\ /	[0-9]	{2}	\ /	[0-9]	{4}	$			

Item pos 0: ^ start string comparison from the beginning so
Item pos 1: [0-9] The 1st char is ^ position 1 that the string must have is a digit

item pos 2: {2} defines we require 2 of the previous item – so 2 digits

Item pos 3: \ / - escape followed by / so the 3rd character in the string must be a /

Item pos 4: [0-9] single character 0 to 9

Item pos 5: {2} – links to previous item – a single character but 2 of them

Regular expressions – examples using Bash

- Reg expression: ^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$

0	1	2	3	4	5	6	7	8	9	10	11	12
^	[0-9]	{2}	\ /	[0-9]	{2}	\ /	[0-9]	{4}	$			

Item pos 6: \ / - escape followed by / so the 3rd character in the string must be a /

Item pos 7: [0-9] single character 0 to 9

Item pos 8: {2} – links to previous item – a single character but 2 of them

Item pos 9: identifies the previous item has to be at the end of string

Regular expressions – examples using Bash

- Reg expression: ^[0-9][abcA-C]{2,3}$

0	1	2	3	4	5	6	7
^	[0-9]	[abcA-C]	{2,3}	$			

Item 0: ^ match the next item (item 1) to be at the beginning of string

Item 1: [0-9] single digit legal values 0 to 9

Item 2: [abcA-C] single character defined legal values a,b,c,A,B,C

Item 3:{2,3} referring to previous item (item 2) match minimum of 2 and a maximum of 3

Regular expressions – examples using Bash

- Reg expression: ^[0-9][abcA-C]{2,3}$

0	1	2	3	4	5	6	7
^	[0-9]	[abcA-C]	{2,3}	$			

8abc Match
8ab Match
8a No match
Abc No match
8abcA No match

Regular expressions – examples using Bash

- Reg expression: ^[0-9][abcA-C]{2,}$

0	1	2	3	4	5	6	7
^	[0-9]	[abcA-C]	{2,}	$			

Item 0: ^ match the next item (item 1) to be at the beginning of string

Item 1: [0-9] single digit legal values 0 to 9

Item 2: [abcA-C] single character defined legal values a,b,c,A,B,C

Item 3:{2,} referring to previous item (item 2) match minimum of 2 or more

Regular expressions – examples using Bash

- Reg expression: ^[0-9][abcA-C]{2,}$

0	1	2	3	4	5	6	7
^	[0-9]	[abcA-C]	{2,}	$			

8abc Match
8ab Match
8a No match
Abc No match
8abcA Match

Regular expressions – examples using Bash

- Reg expression: ^[^0-4]([abcA-C]{4}|E123)$

0	1	2	3
^	[^0-4]	([abcA-C]{4}|E123)	$

Item pos 0 : ^ match next item to the right has to be at the beginning of string

Item pos 1 : [^0-4] match a single character except for digits 0,1,2,3,4

Item pos 2: ([abcA-C]{4}|E123)

Item in brackets – so grouped

| -- logical OR legal match [abcA-C]{4} OR E123 match just one

[abcA-C]{4} – represents set of single characters a,b,c,A,B,C from this set to match 4 characters in the string, where elements can be used more than once e.g. aaAbC

Item pos 3: identifies last item to match the end of the string

Regular expressions – examples using Bash

- Reg expression: [0-9]{3}-?[0-9]{3}-?[0-9]{4}

0	1	2	3	4	5	6	7	8	9
[0-9]	{3}	-	?	[0-9]	{3}	-	?	[0-9]	{4}

Item pos 0 : [0-9] single digit

Item pos 1 : {3} exactly 3 of the item on the left

Item pos 2 : match – character - alternative can have \-

Item pos 3: ? Refers to previous item on left (item 2) 0 or 1 of them

Regular expressions – examples using Bash

- Reg expression: [0-9]{3}-?[0-9]{3}-?[0-9]{4}

0	1	2	3	4	5	6	7	8	9
[0-9]	{3}	-	?	[0-9]	{3}	-	?	[0-9]	{4}

Item pos 4 : [0-9] single digit

Item pos 5 : {3} exactly 3 of the item on the left

Item pos 6 : - character, alternative can have \-

Item pos 7: ? Refers to previous item on left (item 6). 0 or 1 of them

Regular expressions – examples using Bash

- Reg expression: [0-9]{3}-?[0-9]{3}-?[0-9]{4} [0-9]{3}\-?[0-9]{3}\-?[0-9]{4}

0	1	2	3	4	5	6	7	8	9
[0-9]	{3}	-	?	[0-9]	{3}	-	?	[0-9]	{4}

Item pos 8 : [0-9] single digit

Item pos 9 : {4} exactly 4 of the item on the left

Note as we don't have ^ at the beginning or $ the substring can be found in the middle of the string

1231231234	Match
123-123-123	No match
123-123-1234	Match
abc123-123-1234ab67hs	Match

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[[:digit:]]

0	1	2	3
^	[[:alpha:]]	{4}	[[:digit:]]

Item 0 : ^ match item to the right (item 1) from the beginning of string

Item 1 : [[:alpha:]] Character classes –matches single character

Item 2 : {4} – exactly 4 items (of left item (item 1)), so matches 4 alphabetic characters

Item 3 : [[:digit:]] uses digit character class – outer square brackets define 1 character match

This regular expression will match from the left so this can describe a substring

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[[:digit:]]

0	1	2	3
^	[[:alpha:]]	{4}	[[:digit:]]

12de1 No Match
abcd1 Match
123a4 No Match
AcaB99 Match

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[[:digit:]]$

0	1	2	3	4
^	[[:alpha:]]	{4}	[[:digit:]]	$

Same as before but for
Item 4: $ here we define the end has to end with a single digit

So now we define the entire string

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[[:digit:]]$

0	1	2	3
^	[[:alpha:]]	{4}	[[:digit:]]

12de1 No Match
abcd1 Match
123a4 No Match
AcaB99 No Match

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[e]

0	1	2	3
^	[[:alpha:]]	{4}	[e]

Item 0: ^ match the string from the left (beginning) where we compare with item 1

Item 1: [[:alpha:]] Character classes –matches single character

Item 2: {4} we require 4 of the item to the left matches i.e. 4 letters

Item 3: [e] match single character e

Note as we do not have $ at the end of the regular expression we match substring from the left

Regular expressions – examples using Bash

- Reg expression: ^[[:alpha:]]{4}[e]

0	1	2	3
^	[[:alpha:]]	{4}	[e]

Abcde Match
Abcdef Match
Abcde123 Match

Regular expressions – examples using Bash

- Reg expression: [[:alpha:]]{4}[e]$

0	1	2	3
[[:alpha:]]	{4}	[e]	$

Item 0: [[:alpha:]] Character classes –matches single character

Item 1: {4} we require 4 of the item to the left matches i.e. 4 letters

Item 2: [e] match single character e

Item 3: $ match the regex pattern from the end of the string towards the beginning (LHS)

Regular expressions – examples using Bash

- Reg expression: [[:alpha:]]{4}[e]$

0	1	2	3
^	[[:alpha:]]	{4}	[e]

7897abcde Match
Abcdef No Match
Abcde123 No Match

Regular expressions – examples using Bash

- Reg expression: ^(\(|\{) ?[\ ]*test[\ ]*(\)|\})?$

0	1	2	3	4	5	6	7	8	9	10
^	(\(|\{)	?	[\ ]	*	test	[\ ]	*	(\)|\})	?	$

Item 0: no string before and we start with item 1

Item 1: ( ) – form group; within group \(|\{ escape circular bracket (OR function escape curly bracket { Test if we have either a circular bracket or a curly bracket

Item 2: ? - 0 or one of item on left (item 1)

Item 3: [\ ] match single blank space escape followed by blank

Item 4: * 0 or more of item 3 (blank spaces)

Regular expressions – examples using Bash

- Reg expression: ^(\(|\{) ?[\ ]*test[\ ]*(\)|\})?$

0	1	2	3	4	5	6	7	8	9	10
^	(\(|\{)	?	[\ ]	*	test	[\ ]	*	(\)|\})	?	$

Item 5: string test

Item 6: [\ ] match single blank space escape followed by blank

Item 7: * 0 or more of item 6 (blank space)

Item 8: ( ) – form group; within group \)|\} escape circular bracket) OR function escape curly bracket } either a circular bracket or a curly bracket

Item 9: ? - 0 or one of item on left (item 8)

Item 10: $ end of string

Regular expressions – examples using Bash

- Reg expression: ^(\(|\{) ?[\ ]*test[\ ]*(\)|\})?$

0	1	2	3	4	5	6	7	8	9	10
^	(\(|\{)	?	[\ ]	*	test	[\ ]	*	(\)|\})	?	$

test Match
(test) Match
{test) Match
{test) Match
(test) Match
((test) No match
( ) No match
(test)) No match

Note no outer quotes used in bash script

$str1 =~ ^(\(|\{) ?[\ ]*test[\ ]*(\)|\})?$

Bash script used to test regular expression

```
#!/bin/bash
read -p "Type string  " str1
echo -e "String entered :$str1\n"
if [[ $str1 =~ ^\^[0-9a-dA-C]*$ ]]
then
   echo -e "match\n"
else
   echo -e "no match\n"
fi
exit
```

Note:
Single [] are posix shell compliant condition tests.
Double [[]] are an extension to the standard []