

# Introduction to pattern matching

Using Regexes



**CIRCL**

Computer Incident  
Response Center  
Luxembourg

Alexandre Dulaunoy

[alexandre.dulaunoy@circl.lu](mailto:alexandre.dulaunoy@circl.lu)

Christian Studer

[christian.studer@circl.lu](mailto:christian.studer@circl.lu)

[info@circl.lu](mailto:info@circl.lu)

M2 SSI - 2024/01/26

## Typical Linux problem

---

```
1 $ cat files.txt
2 readme.md
3 document.pdf
4 image.png
5 music.mp3
6 video.mp4
7 manual.pdf
```

**Objectives:** List only PDF files

# \$ man grep

---

```
1  GREP(1)                                User Commands                                GREP(1)
2
3  NAME
4      grep, egrep, fgrep, rgrep - print lines that match
      patterns
5
6  SYNOPSIS
7      grep [OPTION...] PATTERNS [FILE...]
8      grep [OPTION...] -e PATTERNS ... [FILE...]
9      grep [OPTION...] -f PATTERN_FILE ... [FILE...]
10
11 DESCRIPTION
12      grep searches for PATTERNS in each FILE. PATTERNS
      is one or more patterns separated by newline
      characters, and grep prints each line that matches
      a pattern. Typically PATTERNS should be quoted
      when grep is used in a shell command.
```

# Using grep

---

```
1 $ cat files.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
```

Easy! However...

## Using grep

---

```
1 $ cat files.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
```

Easy! However...

```
1 $ cat files-2.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
4 homework.pdf.jpg
```

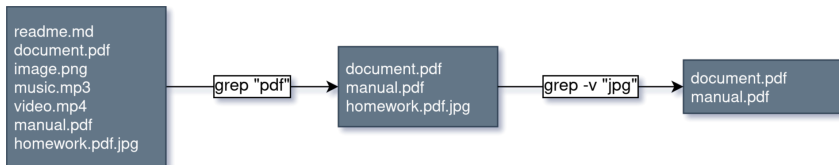
How can we filter out homework.pdf.jpg?

# Using grep

---

```
1 $ cat files-2.txt | grep 'pdf' | grep -v 'jpg'
2 document.pdf
3 manual.pdf
```

`-v` allows us to perform an in**V**ert match



## Using grep

---

```
1 $ cat files-3.txt | grep 'pdf' | grep -v 'jpg'
2 document.pdf
3 manual.pdf
4 adobe_pdf_reader.exe
5 i_hate_pdf.mp3
6 this.is.a.weird.pdf.filename.zip
7 filename with spaces are evil.pdf
```

Using invert match is not going to scale...

## Other commonly encountered problems

---

- Matching valid email addresses
- Matching valid IBAN number
- Matching valid IPv4 addresses

→ Regular Expressions (**Regex**) to the rescue



# Regular Expression

---

A **regular expression** (shortened as **regex** or **regexp**) is a sequence of characters that specifies a search pattern in text.

**Regexes** are extremely useful in extracting information from text.

# Regular Expression Basics

---

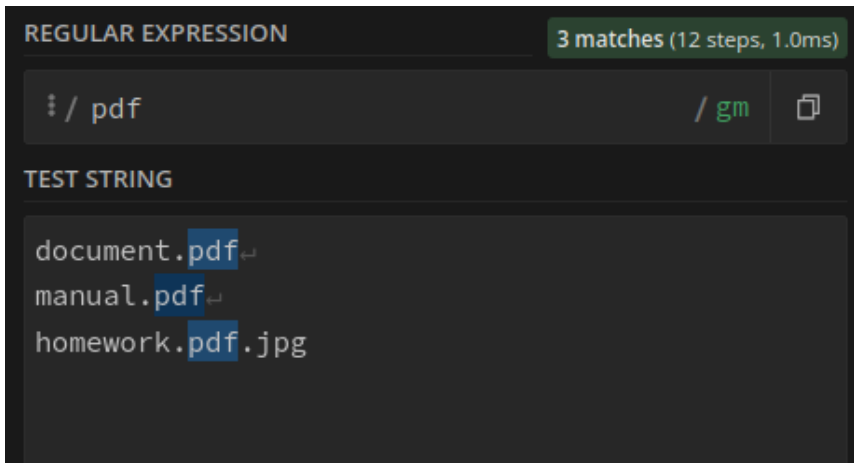
- What ?
  - Literal characters: abc
  - Quantifiers: ab+c
  - Operator OR: (abc|cba)
  - Bracket expressions: [a-z]
  - Meta sequences: \S
  - Capture group: (abc)
  - Anchors: ^abc\$
- New skill ?

```
/ <([a-z]+)(>(.*<\/\1>|\s+\/>) /
```

## Regular Expression Basics: Literal characters

---

Letters and digits from the ASCII character set match their respective value

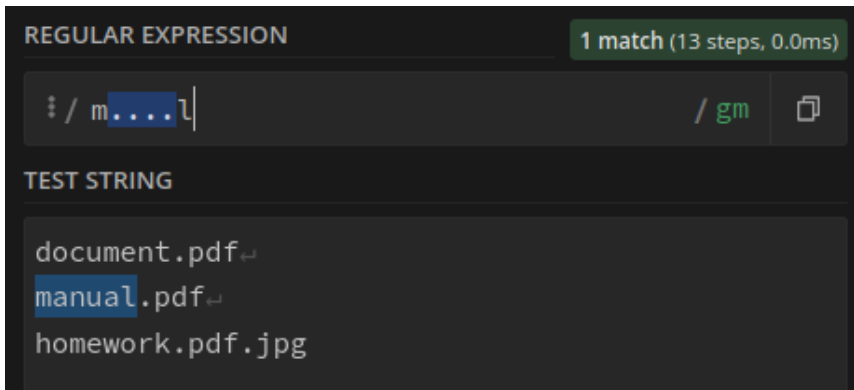


The screenshot shows a dark-themed interface for testing a regular expression. At the top left, the text "REGULAR EXPRESSION" is displayed. To its right, a green box contains the text "3 matches (12 steps, 1.0ms)". Below this, the regular expression "/ pdf" is entered in a text field, with a search icon to its right. To the right of the text field, the flags "/ gm" are visible, along with a copy icon. Below the text field, the text "TEST STRING" is displayed. Underneath, three lines of text are shown: "document.pdf", "manual.pdf", and "homework.pdf.jpg". The ".pdf" portion of each line is highlighted with a blue background, indicating a successful match.

## Regular Expression Basics: The Dot

---

. is a *joker* or *wildcard* that can match any single character

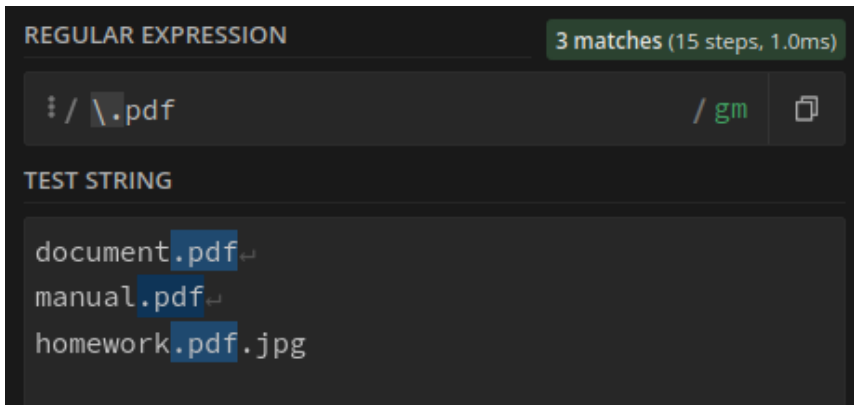


The screenshot shows a dark-themed interface for testing a regular expression. At the top left, the text "REGULAR EXPRESSION" is displayed. To its right, a green badge indicates "1 match (13 steps, 0.0ms)". Below this, the regular expression pattern `/ m.....|` is entered in a text field, with the dots highlighted in blue. To the right of the text field are flags `/ gm` and a copy icon. Below the text field, the section "TEST STRING" is shown with three lines of text: `document.pdf`, `manual.pdf`, and `homework.pdf.jpg`. The `manu` part of `manual.pdf` is highlighted in blue, indicating it is the match.


## Regular Expression Basics: The Period

---

The period character `.` can be matched using the escape character `\`.



REGULAR EXPRESSION 3 matches (15 steps, 1.0ms)

`/\.pdf` / gm 

TEST STRING

document.pdf ↵  
manual.pdf ↵  
homework.pdf.jpg

## Regular Expression Basics: OR Operator

---

The ( | ) structure can be used as a logical operator to match one sequence or the other

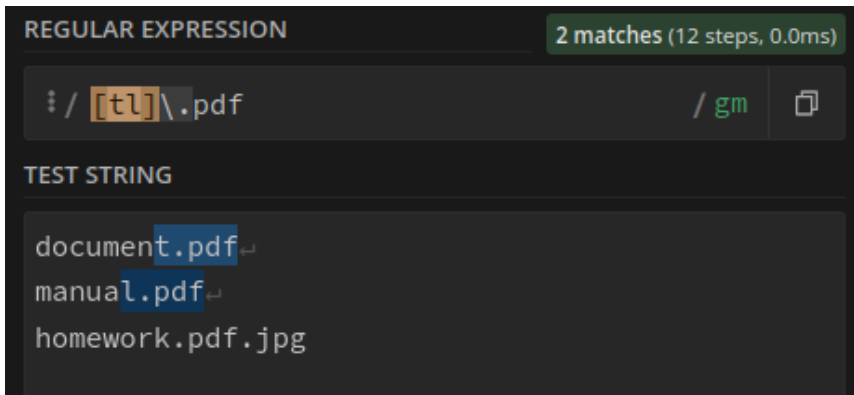
The screenshot shows a dark-themed interface for testing a regular expression. At the top, it says "REGULAR EXPRESSION" and "2 matches (35 steps, 0.0ms)". The regular expression being tested is `/ (document|manual)\.pdf`. Below this, under the heading "TEST STRING", there are three lines of text: `document.pdf`, `manual.pdf`, and `homework.pdf.jpg`. The first two lines are highlighted with a green background, indicating they are matches. The third line is not highlighted, indicating it is not a match.

```
REGULAR EXPRESSION 2 matches (35 steps, 0.0ms)  
/ (document|manual)\.pdf / gm  
TEST STRING  
document.pdf  
manual.pdf  
homework.pdf.jpg
```

## Regular Expression Basics: Bracket Expression (1)

---

The `[ ]` structure can be used to specify a set of characters that can match

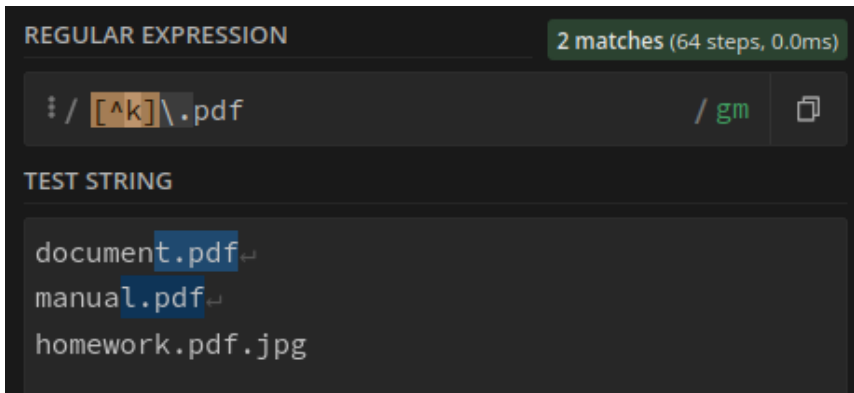


The screenshot shows a dark-themed interface for testing a regular expression. At the top left, the text "REGULAR EXPRESSION" is displayed. To its right, a green box contains the text "2 matches (12 steps, 0.0ms)". Below this, the regular expression `/ [t\] \.pdf` is entered in a text field, with the bracketed characters `[t\]` highlighted in orange. To the right of the text field are the flags `/ gm` and a copy icon. Below the text field, the text "TEST STRING" is displayed. Underneath, a list of three test strings is shown: `document.pdf`, `manual.pdf`, and `homework.pdf.jpg`. The `ent` part of `document.pdf` and the `al` part of `manual.pdf` are highlighted in blue, indicating they are matches for the regular expression.

## Regular Expression Basics: Bracket Expression (2)

---

The `[^ ]` structure can be used to exclude a specific set of characters



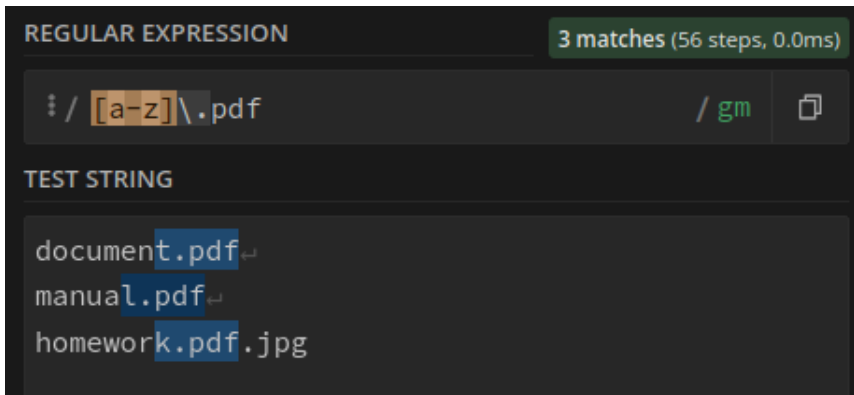
The screenshot shows a dark-themed interface for testing a regular expression. At the top left, it says "REGULAR EXPRESSION". On the top right, a green badge indicates "2 matches (64 steps, 0.0ms)". Below this, the regular expression `/ [^k]\\.pdf` is entered in a text field, with the `[^k]` part highlighted in orange. To the right of the text field are flags `/ gm` and a copy icon. Below the text field, the section "TEST STRING" contains three lines of text: `document.pdf`, `manual.pdf`, and `homework.pdf.jpg`. The `document.pdf` and `manual.pdf` strings are highlighted in blue, indicating they are matches for the regular expression.



## Regular Expression Basics: Bracket Expression (3)

---

The `[ - ]` structure can be used to specify a range of sequential characters



The screenshot shows a dark-themed interface for testing a regular expression. At the top left, the text "REGULAR EXPRESSION" is displayed. To its right, a green badge indicates "3 matches (56 steps, 0.0ms)". Below this, the regular expression `/ [a-z] \.pdf` is entered in a text field, with `[a-z]` highlighted in orange. To the right of the text field are flags `/ gm` and a copy icon. Below the text field, the section "TEST STRING" is visible, containing three lines of text: `document.pdf`, `manual.pdf`, and `homework.pdf.jpg`. In each line, the file extension part is highlighted in blue: `.pdf` in the first line, `.pdf` in the second line, and `.pdf` in the third line.

## Regular Expression Basics: Meta Sequences (1)

---

- / . /
  - Any single character
- / \w /
  - Any word character
  - / [a-zA-Z0-9\_] /
  - **Match:** any non-whitespace character \$!-:;
- / \W /
  - Any non-word character
  - / [^a-zA-Z0-9\_] /
  - **Match:** any whitespace character \$!-:;

## Regular Expression Basics: Meta Sequences (2)

---

- / \d /

- Any digit

- **Match:** one: 1 , two: 2 ;

- / \s /

- Any whitespace character

- **Match:** any whitespace character \$!-:;

- / \S /

- Any non-whitespace character

- **Match:** any non-whitespace character \$!-:;

## Regular Expression Basics: Reading Exercises (1)

---

1. / facebo.k /

## Regular Expression Basics: Reading Exercises (1)

---

1. / facebo.k /
  - **Match:** facebook, faceboak, facebo&k
2. / 4\.2 /

## Regular Expression Basics: Reading Exercises (1)

---

1. / facebo.k /
  - **Match:** facebook, faceboak, facebo&k
2. / 4\.2 /
  - **Match:** 4.2
  - **Match:** Nice number: 4.2
3. / drink (beer|wine) ! /

## Regular Expression Basics: Reading Exercises (1)

---

1. / facebo.k /
  - **Match:** facebook, faceboak, facebo&k
2. / 4\.2 /
  - **Match:** 4.2
  - **Match:** Nice number: 4.2
3. / drink (beer|wine) ! /
  - **Match:** I drink beer !
  - **Match:** I drink wine !
4. / [e-h] /

## Regular Expression Basics: Reading Exercises (1)

---

1. / facebo.k /
  - **Match:** facebook, faceboak, facebo&k
2. / 4\.2 /
  - **Match:** 4.2
  - **Match:** Nice number: 4.2
3. / drink (beer|wine) ! /
  - **Match:** I drink beer !
  - **Match:** I drink wine !
4. / [e-h] /
  - **Match:** fefe, hehe
  - **No match:** haha



## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*

## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*  
/ *\_light* /
2. **Match:** *red\_light* and *green\_light* **but not** *white\_light*

## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*  
/ *\_light* /
2. **Match:** *red\_light* and *green\_light* **but not** *white\_light*  
/ (red|green)\_light /
3. **Match:** *\*\_light* where *\** is any digit

## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*  
/ *\_light* /
2. **Match:** *red\_light* and *green\_light* **but not** *white\_light*  
/ (red|green)\_light /
3. **Match:** *\*\_light* where *\** is any digit  
/ [0-9]\_light /
4. **Match:** *?\_light* where *?* is 4-letters color name

## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*  
/ *\_light* /
2. **Match:** *red\_light* and *green\_light* **but not** *white\_light*  
/ (red|green)\_light /
3. **Match:** *\*\_light* where *\** is any digit  
/ [0-9]\_light /
4. **Match:** *?\_light* where *?* is 4-letters color name  
/ [a-z][a-z][a-z][a-z]\_light /

## Regular Expression Basics: Writing Exercises (1)

---

1. **Match:** *red\_light*, *green\_light* and *!=\_light*  
/ *\_light* /
2. **Match:** *red\_light* and *green\_light* **but not** *white\_light*  
/ (red|green)\_light /
3. **Match:** *\*\_light* where \* is any digit  
/ [0-9]\_light /
4. **Match:** *?\_light* where ? is 4-letters color name  
/ [a-z][a-z][a-z][a-z]\_light /

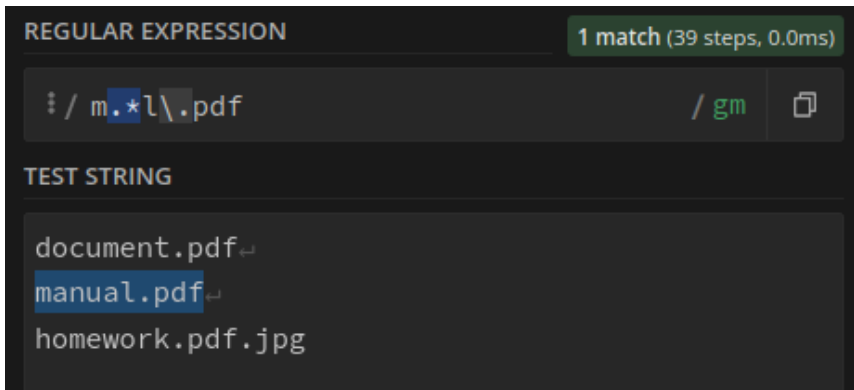
**Question:** *?\_light* where ? is any color between 3 and 6 letters

We need a way to express occurrences... Introducing **quantifiers**

## Regular Expression Basics: Quantifiers (1)

---

The \* control character can be used to describe **zero or more** occurrences

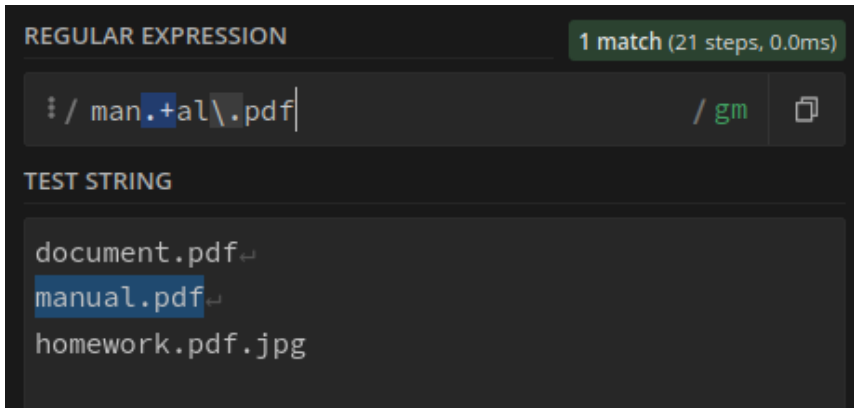


The screenshot shows a dark-themed interface for testing a regular expression. At the top left, the text "REGULAR EXPRESSION" is displayed. To its right, a green badge indicates "1 match (39 steps, 0.0ms)". Below this, the regular expression `/ m.*l\.pdf` is entered in a text field, with the asterisk highlighted in blue. To the right of the text field are flags `/ gm` and a copy icon. Below the text field, the section "TEST STRING" is visible, containing a list of strings: `document.pdf`, `manual.pdf` (highlighted in blue), and `homework.pdf.jpg`.

## Regular Expression Basics: Quantifiers (2)

---

The + control character can be used to describe **one or more** occurrences



REGULAR EXPRESSION 1 match (21 steps, 0.0ms)

`/ man.+al\\.pdf | / gm`

TEST STRING

```
document.pdf ↵
manual.pdf ↵
homework.pdf.jpg
```



## Regular Expression Basics: Quantifiers (3)

---

- / a? /
  - Match 0 or one a character
- / a{3} /
  - Match exactly 3 a character
- / a{3,} /
  - Match 3 or more a character
- / a{3,6} /
  - Match between 3 and 6 a character

## Regular Expression Basics: Reading Exercises (2)

---

1. / colou?r /

## Regular Expression Basics: Reading Exercises (2)

---

1. / colou?r /
  - **Match:** colour and color
2. / go\*gle /

## Regular Expression Basics: Reading Exercises (2)

---

1. / colou?r /
  - **Match:** colour and color
2. / go\*gle /
  - **Match:** gogle, goooooogle, ggle, ...
3. / waz+up /

## Regular Expression Basics: Reading Exercises (2)

---

1. / colou?r /
  - **Match:** colour and color
2. / go\*gle /
  - **Match:** gogle, goooooogle, ggle, ...
3. / waz+up /
  - **Match:** wazup, wazzzzzup, ...
4. / +352[0-9]{6,8} /

## Regular Expression Basics: Reading Exercises (2)

---

1. / colou?r /
  - **Match:** colour and color
2. / go\*gle /
  - **Match:** gogle, goooooogle, ggle, ...
3. / waz+up /
  - **Match:** wazup, wazzzzzup, ...
4. / +352[0-9]{6,8} /
  - **Match:** +352791648, +35226791349

## Regular Expression Basics: Writing Exercises (2)

---

1. **Match:** The time (16:42, 03:59)

## Regular Expression Basics: Writing Exercises (2)

---

1. **Match:** The time (16:42, 03:59)  
/ [0-9] [0-9] : [0-9] [0-9] /  
(not perfect but good enough for the exercise)
2. **Match:** Luxembourg postal code (L-4253, L-1110)



## Regular Expression Basics: Writing Exercises (2)

---

1. **Match:** The time (16:42, 03:59)  
/ [0-9] [0-9] : [0-9] [0-9] /  
(not perfect but good enough for the exercise)
2. **Match:** Luxembourg postal code (L-4253, L-1110)  
/ L-[0-9]{4} /
3. **Match:** \*\_light where \* is any color?

## Regular Expression Basics: Writing Exercises (2)

---

- Match:** The time (16:42, 03:59)  
/ [0-9][0-9]:[0-9][0-9] /  
(not perfect but good enough for the exercise)
- Match:** Luxembourg postal code (L-4253, L-1110)  
/ L-[0-9]{4} /
- Match:** \*\_light where \* is any color?  
/ [a-z]+\_light /
- Match:** any hexadecimal color (#ff0000, #f7f8f9)

## Regular Expression Basics: Writing Exercises (2)

---

1. **Match:** The time (16:42, 03:59)  
/ [0-9][0-9]:[0-9][0-9] /  
(not perfect but good enough for the exercise)
2. **Match:** Luxembourg postal code (L-4253, L-1110)  
/ L-[0-9]{4} /
3. **Match:** \*\_light where \* is any color?  
/ [a-z]+\_light /
4. **Match:** any hexadecimal color (#ff0000, #f7f8f9)  
/ #[a-f0-9]{6} /

## Regular Expressions: Final question

---

What does these regexes do?

1. / ([12]\d{3}-(0[1-9]|1[0-2]))-(0[1-9]| [12]\d|3[01])) /
2. / <([a-z]+)>(.\*<\/\1>|\s+\/>) /
  - \1 is used to reference the first capturing group
  - First capturing group is ([a-z]+)

## Regexes: Going further

---

- `^` and `$` anchors
- Capture **groups**
- **Greedy** and **Lazy** quantifiers
- **Possessive** quantifier

# Linux commands and Regexes

# Linux commands and Regexes

---

You have a small subset of a files crawled on Pastebin

```
1 $ du -hd1
2 158M      .
3 $ ls | wc
4 11875     11875    106875
```

# Linux commands and Regexes

---

Let's list the number of files that have the disney keyword

```
1 $ grep -irl disney | wc
2 88      88      792
```

Let's filter on common email providers

```
1 $ egrep -ir 'disney'
2     | egrep -i '(hotmail|live|outlook|gmail|yahoo) '
3     | wc
4 52  122772 1566533
```



## Linux commands and Regexes

---

Let's filter on the email:password pattern

```
1 $ egrep -ir 'disney'  
2   | egrep -i '(hotmail|live|outlook|gmail|yahoo)'  
3   | egrep -i '[a-zA-Z0-9_\-\\+\\.]+@\\w+(\\.\\w+)+:\\S+ '  
4   | wc  
5 13      902      21758
```

Let's extract all credentials and only the credentials

```
1 $ egrep -ir 'disney'  
2   | egrep -i '(hotmail|live|outlook|gmail|yahoo)'  
3   | egrep -io '[a-zA-Z0-9_\-\\+\\.]+@\\w+(\\.\\w+)+:\\S+ '  
4   | wc  
5 13      13      438
```

# Linux commands and Regexes

---

Let's search for other type of accounts

```
1 $ egrep -ir 'psn'
2   | egrep -i '(hotmail|live|outlook|gmail|yahoo)'
3   | egrep -i '[a-zA-Z0-9_-\+\\.]+\@\w+(\.\w+)+:\S+'
4   | wc
5 2      124      583
```

# Fun with Regular Expressions

---

<https://regexcrossword.com/>

	[^SPEAK]+	EP IP EF
HE LL O+		
[PLEASE]+		

# Fun with Regular Expressions

---

<https://regexcrossword.com/>

	[^SPEAK]+	EP IP EF
HE LL O+	H	E
[PLEASE]+	L	P

# Fun with Regular Expressions

---

<https://regexcrossword.com/>

	(FI A)+	(YE OT)K	(.) [IF]+	[NODE]+	(FY F RG)+
(Y F)(.)\2[DAF]\1					
(U O I)*T[FRO]+					
[KANE]*[GIN]*					

# Fun with Regular Expressions

---

<https://regexcrossword.com/>

	(FI A)+	(YE OT)K	(.) [IF]+	[NODE]+	(FY F RG)+
(Y F)(.)\2[DAF]\1	F	O	O	D	F
(U O I)*T[FR0]+	I	T	F	O	R
[KANE]*[GIN]*	A	K	I	N	G

## Simple CTF challenge with regular expressions

---

- Hidden 4 flags in the dump
- They have the following pattern
  - flag letters in any order
  - \_ The underscore character
  - 2 upper case character
  - at least one special character from the list @ , & , :
  - 2 lower case character
- Example: `fa1g_AA&:bb`

# Simple CTF challenge with regular expressions

---

## SHA1 Sum of the flags

```
1 $ echo 'falg_AA&:bb' | md5sum
2 ee9326ee21572fe4bba8e686a7ba6e5e
```

```
falg_AA&:bb ee9326ee21572fe4bba8e686a7ba6e5e
?           53923b8f8490072107b3e8bb614749ce
?           429698c6d1742f02212ae89d3696577d
?           40178e8ef4264385fb7194176faf2318
```



# Annex

## Regular Expressions: Tag matching (1)

---

Create a Regex that validates the following tags:

```
1 namespace: predicate
2 namespace: predicate="value"
3 name_space: pred_icate="value"
4 namespace: predicate="qwert _+$- yuiop"
5 namespace: predicate="qwert=:yuiop"
```

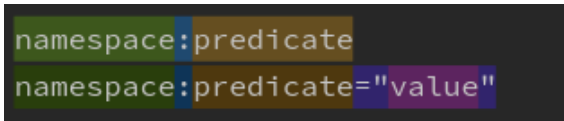
But not these:

```
1 tag
2 name space: pred icate="value"
3 name-space: predicate="value"
4 namespace: predicate="qwert"yuiop"
```

## Regular Expressions: Tag matching (2)

---

A valid tag is composed of 2 or 3 parts



```
namespace: predicate
namespace: predicate="value"
```

1. The namespace
2. The predicate
3. The optional value

## Regular Expressions: Tag matching (3)

---

Tag validator

```
/ ^([\w]+):([\w]+)(="([\^\n"]+)")?$ /
```

## Regular Expressions: Tag matching (4)

---

### EXPLANATION

▼ / `^([\w]+):([\w]+)(="([\^\n"]+)")?$/ gm`

`^` asserts position at start of a line [?](#)

▶ **1st Capturing Group** `([\w]+)`

: matches the character `:` with index `5810` (`3A16` or `728`) literally (case sensitive)

▶ **2nd Capturing Group** `([\w]+)`

▶ **3rd Capturing Group** `(="([\^\n"]+)")?`

`$` asserts position at the end of a line [?](#)

## Regular Expressions: Tag matching (5)

---

▼ **1st Capturing Group** (`([\w]+)`)

▼ **Match a single character present in the list below** `([\w])`

`+` matches the previous token between **one** and **unlimited** times, as many times as possible, giving back as needed (*greedy*)

`\w` matches any word character (equivalent to `[a-zA-Z0-9_]`)

## Regular Expressions: Tag matching (5)

---

### ▼ 3rd Capturing Group `(="([\n"]+)")?`

`?` matches the previous token between **zero** and **one** times, as many times as possible, giving back as needed (greedy)

▶ `=` matches the characters `=` literally (case sensitive)

### ▶ 4th Capturing Group `([\n"]+)`

`"` matches the character `"` with index  $34_{10}$  ( $22_{16}$  or  $42_8$ ) literally (case sensitive)

### ▼ 4th Capturing Group `([\n"]+)`

#### ▼ Match a single character not present in the list below `([\n"]+)`

`+` matches the previous token between **one** and **unlimited** times, as many times as possible, giving back as needed (greedy)

`\n` matches a line-feed (newline) character (ASCII 10)

`"` matches the character `"` with index  $34_{10}$  ( $22_{16}$  or  $42_8$ ) literally (case sensitive)