# Master Regex From ^ to $

## with Bash Tools & Python

**Dolan**

Jan 18, 2025

VanLUG

LinkedIn

Ubuntu

Vancouver Linux Users Group (VanLUG)

# Master Regex From ^ to $

## with Bash Tools & Python

**Dolan**

Jan 18, 2025

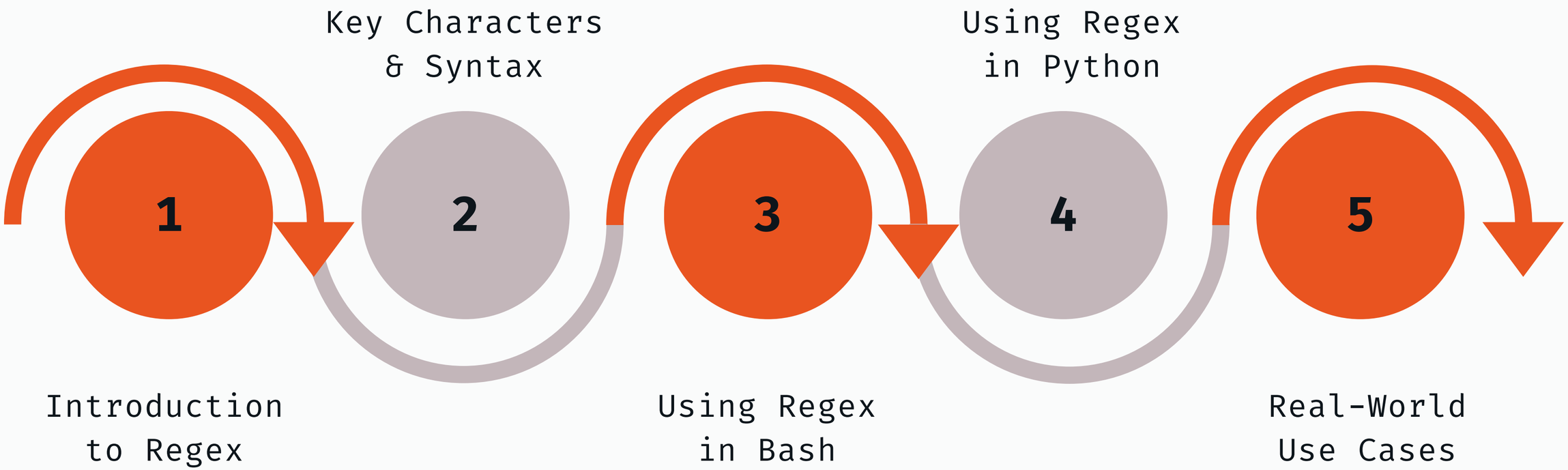
VanLUG

LinkedIn

Ubuntu

Vancouver Linux Users Group (VanLUG)

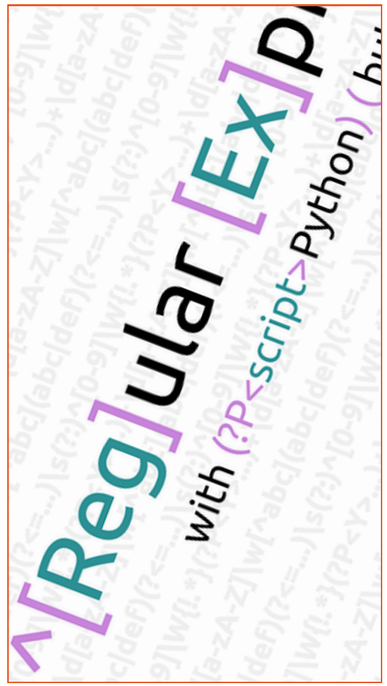
# AGENDA

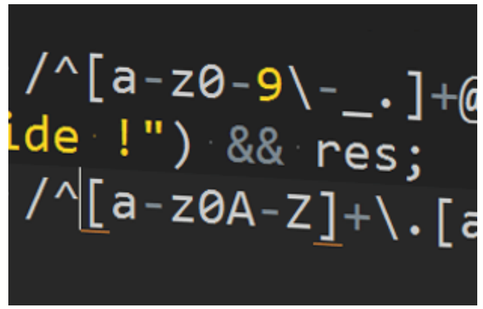

1
Introduction to Regex
2
Key Characters & Syntax
3
Using Regex in Bash
4
Using Regex in Python
5
Real-World Use Cases

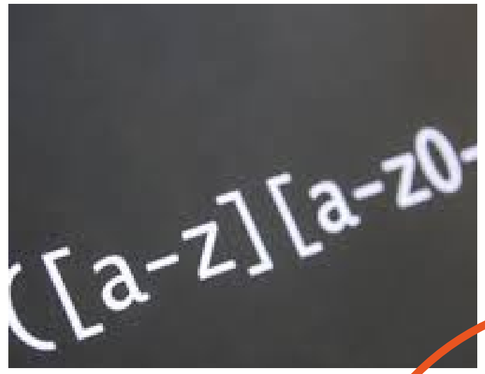
# Introduction to Regex

**Regular Expressions** (Regex) are sequences of characters that define search patterns. Regex is an incredibly powerful tool used across various domains for text processing, pattern matching, and data manipulation. **Regex** simplifies complex text-searching tasks, such as finding or replacing specific patterns, and is widely used in tasks like form validation, log parsing, and data extraction. The most important places where Regex is commonly used:

- Programming and Software and Web Development
- Databases, Data Processing and Analytics
- System Administration and Networking
- Search Engines and Text Retrieval
- Cybersecurity
- Text Editing and IDEs
- Natural Language Processing (NLP)
- Automation and Scripting
- Email and Communication Systems
- AI and Machine Learning

# Syntax overview

Key symbols:

**^ - $ - . - [] - * - + - ? - () - {} - |**

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

^ – $ – . – [] – * – + – ? – () – {} – |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

^ – $ – . – [] – * – + – ? – () – {} – |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

^ - $ - . - [] - * - + - ? - () - {} - |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

^ – $ – . – [] – * – + – ? – () – {} – |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

^ – $ – . – [] – * – + – ? – () – {} – |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

**+ (Plus):** Matches 1 or more repetitions.

^ – $ – . – [] – * – + – ? – () – {} – |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

**+ (Plus):** Matches 1 or more repetitions.

**? (Question Mark):** Matches 0 or 1 repetition.

^ — $ — . — [] — * — + — ? — () — {} — |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

**+ (Plus):** Matches 1 or more repetitions.

**? (Question Mark):** Matches 0 or 1 repetition.

**() (Parentheses):** Groups expressions and captures the match.

^ — $ — . — [] — * — + — ? — () — {} — |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

**+ (Plus):** Matches 1 or more repetitions.

**? (Question Mark):** Matches 0 or 1 repetition.

**() (Parentheses):** Groups expressions and captures the match.

**{} (Curly Braces):** Specifies exact or range repetitions.

^ — $ — . — [] — * — + — ? — () — {} — |

**^ (Caret):** Matches the beginning of a string, or the start of each line in multiline mode.

**$ (Dollar Sign):** Matches the end of a string, or just before the newline at the end, in multiline mode.

**. (Dot):** Matches any character except a newline.

**[] (Square Brackets):** Defines a character set.

*** (Asterisk):** Matches 0 or more repetitions.

**+ (Plus):** Matches 1 or more repetitions.

**? (Question Mark):** Matches 0 or 1 repetition.

**() (Parentheses):** Groups expressions and captures the match.

**{} (Curly Braces):** Specifies exact or range repetitions.

**| (Pipe):** Matches either of the alternatives (logical OR).

^ — $ — . — [] — * — + — ? — () — {} — |

**\n:** Matches a newline.

**\t:** Matches a tab.

**\b:** Matches a word boundary.

**\d:** Matches any digit (0-9).

**\w:** Matches any word character (alphanumeric + _).

**\s:** Matches any whitespace (space, tab, newline).

**\D:** Matches any non-digit.

**\W:** Matches any non-word character.

**\S:** Matches any non-whitespace.

\n - \t - \b - \d - \w - \s - \D - \W - \S

# REGEX IN BASH

```
echo "apple banana cherry" | grep '^a'  # matches "apple"
echo "apple banana cherry" | grep 'a$'  # matches "banana"
```

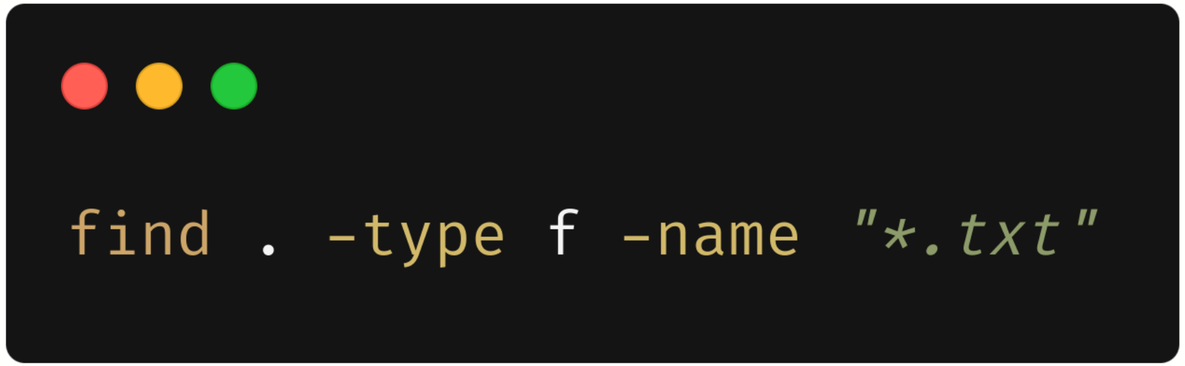
```
# Replace the first occurrence of "foo" with "bar"
echo "foo baz foo" | sed 's/^foo/bar/'
# Output: bar baz foo
```

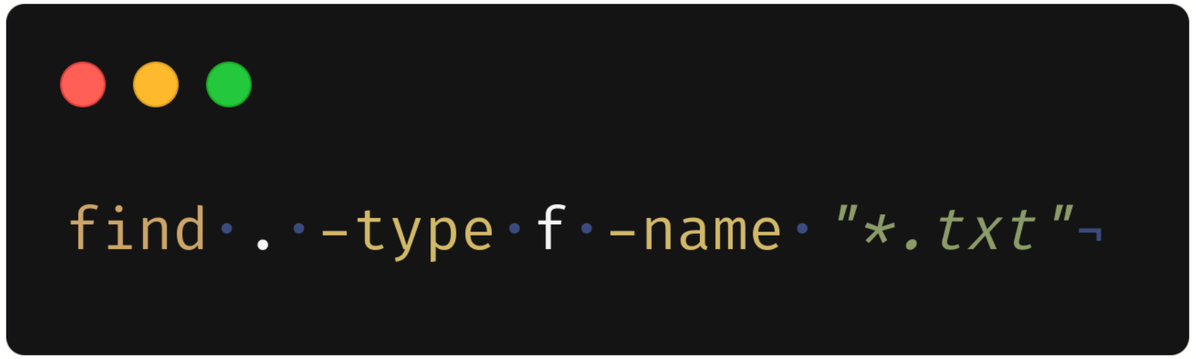
```
# Match lines that start with "Hello"
echo -e "Hello World\nHi World\nHello there" | grep '^Hello'
# Output: Hello World
#         Hello there

# Match lines that end with "World"
echo -e "Hello World\nHi World\nGoodbye World" | grep 'World$'
# Output: Hello World
#         Hi World
#         Goodbye World
```

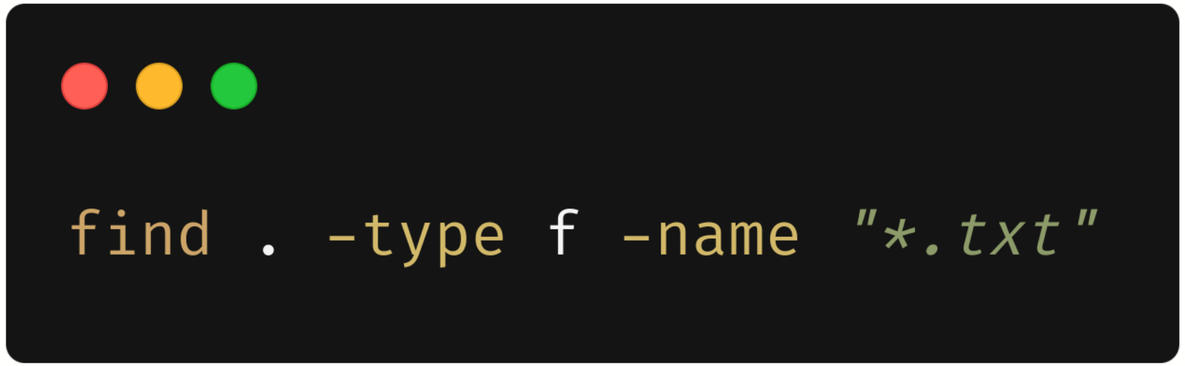
Advanced File Search with `find` and Regex

find all `.txt` files within a directory and its subdirectories:

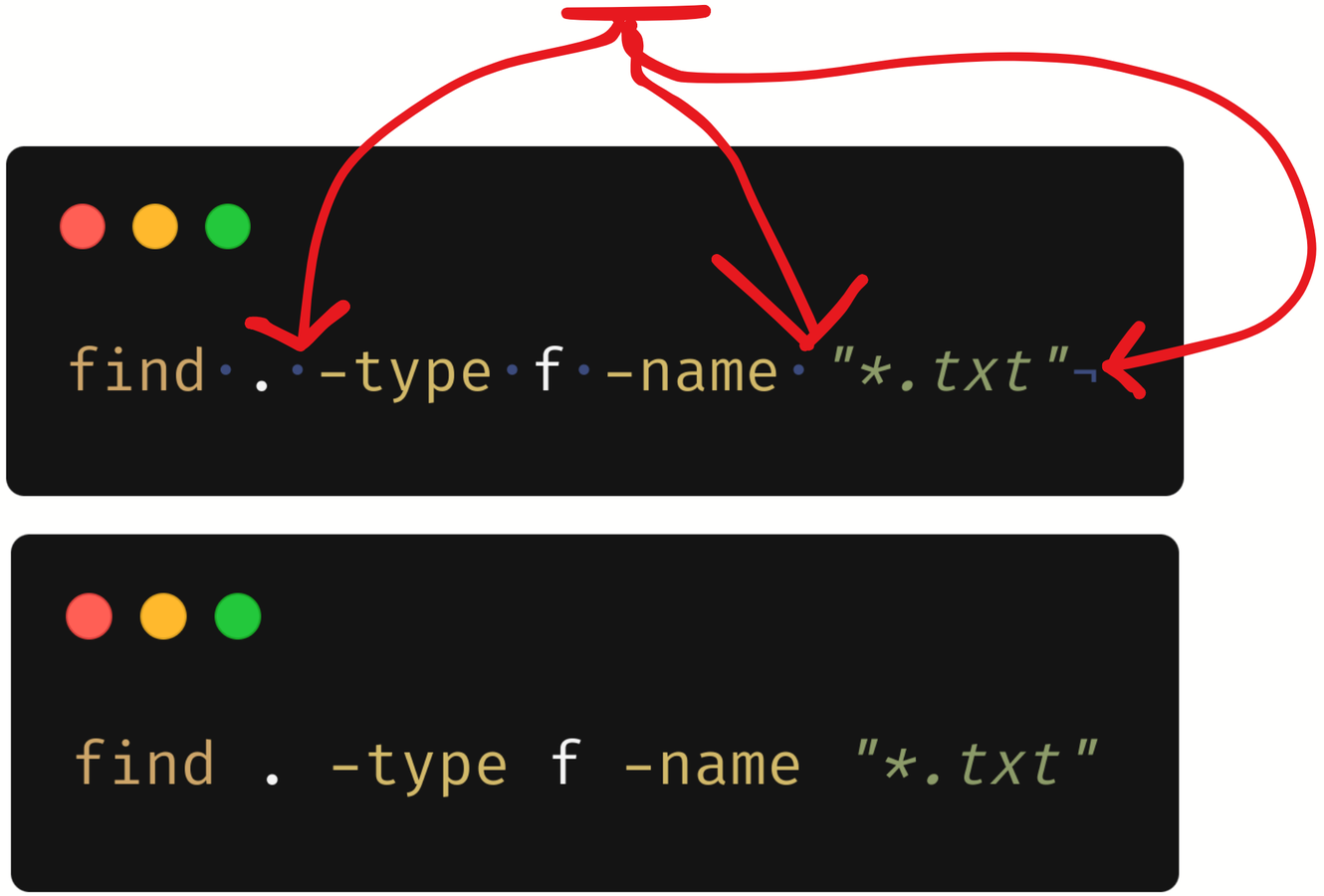
**find . -type f -name "*.txt"**




find . -type f -name "*.txt"


find . -type f -name "*.txt"

# What do these characters represent? ...



Here's an example using regular expressions for foo and bar in sed. This command replaces any word starting with "foo" (e.g., foo123, foo_test) with "bar" followed by the same suffix in all .txt files in the current directory:

**sed -i 's/foo[0-9a-zA-Z_]*/bar/g' *.txt**

```
sed -i 's/foo[0-9a-zA-Z_]*/bar/g' *.txt
```

```
[0-9a-zA-Z_]*
```

[]: Denotes a set of characters to match

**[0-9a-zA-Z_]***

0-9: Matches any digit from 0 to 9

```
[0-9a-zA-Z_]*
```

a-zA-Z_: Matches any lowercase letter, uppercase letter, or an underscore

**[0-9a-zA-Z_]***

-: Used inside [] to specify a range of characters, like 0-9 for digits or a-z for lowercase letters

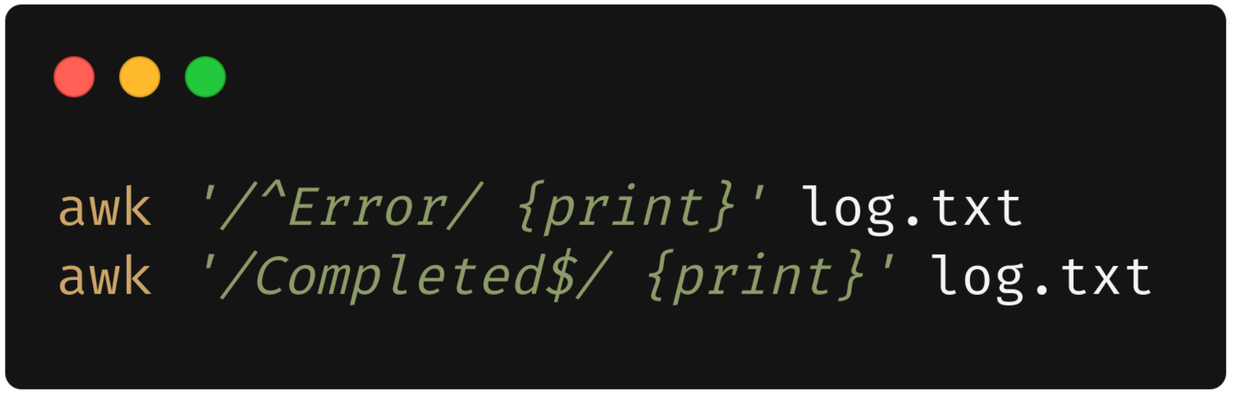
**[0-9a-zA-Z_]***

*: Matches zero or more repetitions

**[0-9a-zA-Z_]***

two examples using awk with ^ (start of line) and $ (end of line):

**awk '/^Error/ {print}' log.txt**
**awk '/Completed$/ {print}' log.txt**



^ (start of line) and $ (end of line)

**^Error**

**Completed$**

The command below extracts and prints all IPv4 addresses from logfile.log:

- -o: Prints only the matched part of each line (the IP address).
- -E: Enables extended regular expressions.

**grep -oE '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' logfile.log**

```
grep -oE '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' logfile.log
```

\b: Word boundary to ensure complete IP addresses are matched

**\b([0-9]{1,3}\.){3}[0-9]{1,3}\b**

- The \b in regular expressions represents a word boundary, which matches the position where a word starts or ends. It doesn't consume any characters but ensures that the match occurs at the edge of a word.

(): groups expressions together, allowing repetition or capturing matched text

**\b([0-9]{1,3}\.){3}[0-9]{1,3}\b**

{1,3}: specifies that the preceding element should appear between 1 and 3 times

**\b([0-9]{1,3}\.){3}[0-9]{1,3}\b**

192.168.1.41

\.: matches a literal dot (period) character, since the dot is a special character in regex.

**\b([0-9]{1,3}\.){3}[0-9]{1,3}\b**

192.168.1.41

{3}: specifies that the preceding element must appear exactly 3 times

```
\b([0-9]{1,3}\.){3}[0-9]{1,3}\b
```

192.168.1.41

192.168.1.41

```
\b([0-9]{1,3}\.){3}[0-9]{1,3}\b
```

192.168.1.41

```
\b([0-9]{1,3}\.){3}[0-9]{1,3}\b
```

To extract email addresses from a file using grep, you can use the following command:

- -o: Prints only the matched email addresses
- -E: Enables extended regular expressions, allowing the use of advanced regex features like +, ?, and | without needing to escape them, making the pattern more readable and flexible.

**grep -oE '\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b' logfile.log**

```
grep -oE '\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b' logfile.log
```

```
\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b
```

What's the difference between the two - symbols?

```
\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b
```

What's the difference between the two + symbols?

```
\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b
```

+: matches one or more repetitions of the preceding character or group

```
\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b
```

# REGEX IN PYTHON

**Python’s re Module:**

Python’s re library provides powerful regex functions for matching, searching, and modifying strings.

Common Functions:

- **search():** Finds the first match of a pattern.
- **match():** Checks if the pattern matches the start of the string.
- **findall():** Finds all instances of a pattern.
- **sub():** Substitutes occurrences of a pattern with a replacement.

**re.function(pattern, string, flags=optional_flags)**

- pattern: The regular expression to match.
- string: The text where the pattern is searched.
- flags (optional): Modifiers like **re.IGNORECASE**, **re.MULTILINE**, etc., that change the behavior of the regex.

search() - match() - findall() - sub()

## pattern

```
# Using raw string
result = re.search(r'\d+', 'abc123xyz')
print(result) # Output: <re.Match object; span=(3, 6), match='123'>
print(result.group()) # Output: 123
```

The **r** prefix before the pattern string makes it a raw string literal. This prevents Python from interpreting backslashes (\) as escape characters (e.g., **\n** for newline, **\t** for tab, etc.).

search() - match() - findall() - sub()

# Common Escape Sequences

| Escape Sequence | Description |
| --- | --- |
| `\\` | Backslash (`\`) |
| `\'` | Single quote (`'`) |
| `\"` | Double quote (`"`) |
| `\n` | Newline |
| `\t` | Horizontal tab |
| `\r` | Carriage return |
| `\b` | Backspace |
| `\f` | Form feed |
| `\v` | Vertical tab |
| `\a` | Bell (alert) |
| `\N{name}` | Unicode character by name |
| `\uXXXX` | 16-bit Unicode (e.g., `\u03A9` for Ω) |
| `\UXXXXXXXX` | 32-bit Unicode (e.g., `\U0001F600` for 😀) |
| `\xXX` | Character with hex value (e.g., `\x41` for 'A') |
| `\ooo` | Character with octal value (e.g., `\101` for 'A') |

## pattern

```
# Without raw string
result = re.search('\\d+', 'abc123xyz')
print(result.group()) # Output: 123
```

search() - match() - findall() - sub()

## pattern

```
# Without raw string
import re
# Match an email-like pattern without using raw strings
pattern = '[a-zA-Z0-9\\.\\_]+@[a-zA-Z]+\\.[a-zA-Z]{2,}'
text = 'Contact us at support.email@example.com for help.'
result = re.search(pattern, text)
print(result.group())  # Output: support.email@example.com
```

search() - match() - findall() - sub()

## pattern

```
# Without raw string
import re
# Match an email-like pattern without using raw strings
# pattern = '[a-zA-Z0-9\\.\\_]+@[a-zA-Z]+\\.[a-zA-Z]{2,}'
pattern = r'[a-zA-Z0-9\.\_]+@[a-zA-Z]+\.[a-zA-Z]{2,}'
text = 'Contact us at support.email@example.com for help.'
result = re.search(pattern, text)
print(result.group())  # Output: support.email@example.com
```

search() - match() - findall() - sub()

**search():** Finds the first match of a pattern **anywhere** in the string.

- \w+ matches one or more word characters ([a-zA-Z0-9_])
- \d+ matches one or more digits

# Finds the first word followed by digits

```
import re
result = re.search(r'\w+\d+', 'abc123xyz')
print(result.group())  # Output: abc123
```

search() - match() - findall() - sub()

**match():** Checks for a match only at the **start** of the string.

```python
import re

# Using re.match
result = re.match(r'\w+\d+', 'abc123xyz')
print(result.group())  # Output: abc123
print(result)
# Output: <re.Match object; span=(0, 6), match='abc123'>
```

**findall():** returns a **list** of all matches, not a match object, so .group() isn't used.

```python
import re

# Using re.findall
result_findall = re.findall(r'\w+\d+', 'abc123xyz')
print(result_findall)  # Output: ['abc123']
```

**sub():** returns a **string** where the pattern has been substituted, so .group() isn't applicable either

```python
import re

# Using re.sub (substitute numbers with '#')
result_sub = re.sub(r'\d+', '#', 'abc123xyz')
print(result_sub)  # Output: abc#xyz
```

Checks if an email is valid

```python
import re
email = "user@example.com"
if re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', email):
    print("Valid email")  # Output: Valid email
```

search() - match() - findall() - sub()

Extracts phone numbers in the format xxx-xxx-xxxx

```python
import re
text = "Contact us at 123-456-7890 or 987-654-3210."
phone_numbers = re.findall(r'\b\d{3}-\d{3}-\d{4}\b', text)
print(phone_numbers)  # Output: ['123-456-7890', '987-654-3210']
```

search() - match() - findall() - sub()

# Extracting URLs

```python
import re
text = "Visit us at https://example.com or http://example.org"
urls = re.findall(r'https?://[^\s]+', text)
print(urls)  # Output: ['https://example.com', 'http://example.org']
```

search() - match() - **findall()** - sub()

# Removing HTML tags

```python
import re
html = "<p>This is <b>bold</b> text</p>"
clean_text = re.sub(r'<.*?>', '', html)
print(clean_text)  # Output: This is bold text
```

# <.*?>

This is the regular expression used to match HTML tags

- **<** and **>**: Match the opening and closing angle brackets of HTML tags.
- **.*?**: Matches any characters (.) in a non-greedy way (*?), ensuring it matches the shortest possible content between the tags (to avoid matching content across multiple tags).

# KEY TAKEAWAYS

Regex enables powerful text processing in both Bash and Python, each providing specific tools and methods.

- **Bash:** Ideal for quick command-line operations and simple text manipulations.
- **Python:** Versatile for complex pattern matching in applications and scripts.

^ - $ - . - [] - * - + - ? - () - {} - |

- **Web Scraping:** Use re.findall() to extract emails or phone numbers from web page content.
- **Form Validation:** Implement regex to validate email, phone, and postal code inputs in forms.
- **Log Filtering:** Parse log files to filter out IP addresses using re and save to a report.
- **Data Extraction:** Extract key metrics (like prices) from text-heavy product descriptions.
- **Sentiment Analysis:** Use regex to identify sentiment words within customer reviews.
- **Text Normalization:** Clean up social media text by removing URLs, hashtags, and mentions.
- **CSV Processing:** Extract specific data patterns from a CSV file (like product codes).
- **PDF Data Mining:** Apply regex to extract structured data (like dates) from OCR text in PDFs.
- **Keyword Search:** Highlight specific keywords in large documents using regex for emphasis.
- **Data Transformation:** Use re.sub() to format inconsistent data, such as phone numbers.

^ - $ - . - [] - * - + - ? - () - {} - |

# TEXT EDITORS

VIM, VS Code, Sublime Text

...

# SOLVE WORDLE

# SOURCES

- https://regex101.com/
- https://docs.python.org/3/library/re.html
- https://www.kaggle.com/datasets/mostafafathy4869/english-words/data
- https://www.kaggle.com/datasets/lennartluik/all-english-words-csv

^ - $ - . - [] - * - + - ? - () - {} - |

# Q&A

VanLUG

LinkedIn

Ubuntu

Vancouver Linux Users Group (VanLUG)

^[Reg]ular [Ex]pressions$

^ - $ - . - [] - * - + - ? - () - {} - |

VanLUG

LinkedIn

Ubuntu

Vancouver Linux Users Group (VanLUG)

^[Reg]ular [Ex]pressions$

^ - $ - . - [] - * - + - ? - () - {} - |