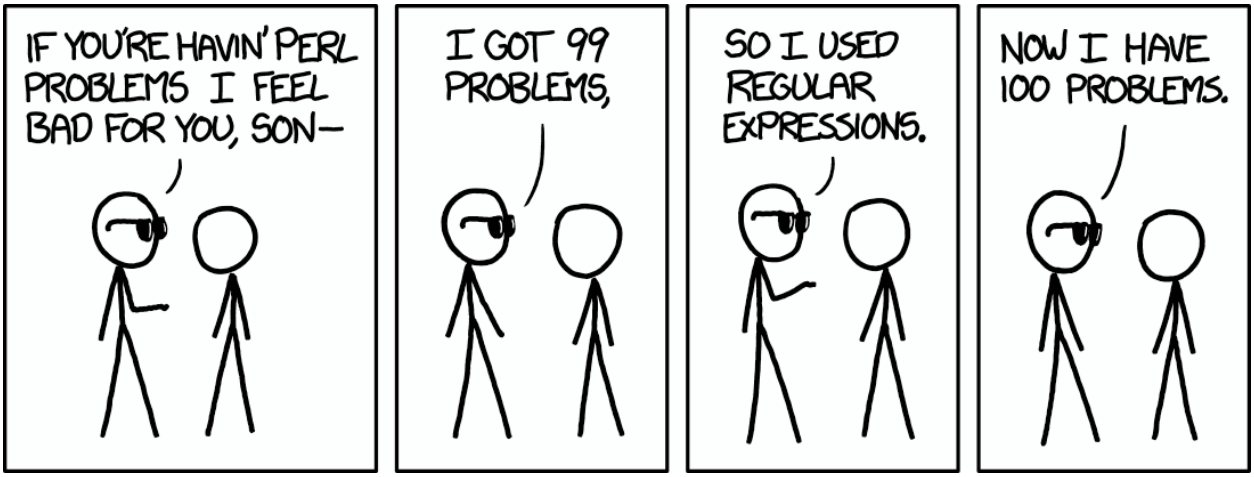
Exercises Fourth Grouping

Prerequisites

- Internet Connection
- Some serious Perseverance
- Ubuntu Server 22.04
- SSH from your Host OS to your VM Linux Server
- Download the password.txt file from the BTA github to your host OS
- Get the password.txt file to your Linux server Home Folder

What's all this about REGEX?

OR



GREP

grep (Global Regular Expression Print) is an essential command-line utility in Unix and Linux environments, widely used for searching within text files or output streams for lines that match a specific pattern defined by a regular expression. Its versatility and power make it an invaluable tool for text processing, data extraction, log analysis, and a wide range of other tasks in system administration, programming, and data analysis.

Grep is a tool that originated from the UNIX world during the 1970's. It can search through files and folders (directories in UNIX) and check which lines in those files match a given regular expression. Grep will output the filenames and the line numbers or the actual lines that matched the regular expression. All in all, a very useful tool for locating information stored anywhere on your computer, even (or especially) if you do not really know where to look.

Text Search

The primary use of grep is to search through text. It can sift through files, directories, or the output of other commands, looking for lines that match a given regex pattern. This is incredibly useful for finding specific entries in log files, searching for occurrences of a term in source code, or filtering lists with specific criteria.

Complex Pattern Matching

With its support for regular expressions, grep allows for complex pattern matching. This includes:

- Searching for variations of a word (e.g., color, colour).
- Matching patterns with optional characters or sequences.
- Extracting lines that match specific numeric ranges or formats.

Pipelines and Redirection

grep is commonly used in pipelines (chaining commands using the pipe | operator) and redirections, allowing for powerful command-line workflows. For example, you can pipe the output of a command through grep to filter for relevant lines, or redirect the output of grep to a file.

File and Directory Searches

grep can recursively search through directories using the -r or -R options, making it easy to perform wide-ranging searches across many files and subdirectories. This is particularly useful in projects with complex directory structures.

Line Numbering

When reviewing matches, it can be helpful to know where they are located. The -n option adds line numbers to grep's output, providing context for each match.

Context Control

grep offers options like -B (before), -A (after), and -C (context) to display lines around the matching line. This is useful for understanding the context of matches within files, such as error messages in log files.

Counting

With the -c option, grep can count the number of lines that match a pattern rather than displaying the lines themselves. This can be useful for quantifying occurrences.

Inverting Matches

The -v option inverts the search, showing lines that do *not* match the given pattern. This is useful for filtering out unwanted information.

Highlighting Matches

Some implementations of grep support highlighting the matching text, making it easier to spot matches among longer lines of text. This can often be achieved with the --color option.

Efficiency and Performance

grep is designed to be efficient with system resources, allowing for the rapid searching of large files or data streams. Its performance is one of the reasons it remains a staple in text processing tasks.

Integration with Other Tools

grep is often used in conjunction with other Unix/Linux command-line tools like awk, sed, sort, and uniq for complex text processing and data manipulation tasks.

In summary, grep's usefulness in the Linux CLI comes from its powerful pattern matching capabilities, flexibility in processing text from files or other commands, and its wide array of options for customizing output. Whether you're a system administrator, programmer, or data analyst, grep is a tool that can significantly streamline your text processing and data analysis workflows.

MAGNETS? HOW DO THEY WORK?

The grep utilities are a family of Unix tools, including grep, egrep, and fgrep, that perform repetitive searching tasks. The tools in the grep family are very similar, and all are used for searching the contents of files for information that matches particular criteria.

The general syntax of the grep commands is:

grep [-options] pattern [filename]

You can use fgrep to find all the lines of a file that contain a particular word. For example, to list all the lines of a file named myfile in the current directory that contain the word "dog", enter at the Unix prompt:

fgrep dog myfile

This will also return lines where "dog" is embedded in larger words, such as "dogma" or "dogged". You can use the -w option with the grep command to return only lines where "dog" is included as a separate word:

grep -w dog myfile

To search for several words separated by spaces, enclose the whole search string in quotes, for example:

fgrep "dog named Checkers" myfile

Please provide the following for your body of work:

- screenshots
    - Commands
    - Starts of outputs
- An explanation of
    - The command
    - The output

Exercise 16

Let's nuke your Terminal (Just joking, I mean I'm going to send a lot of data to standard ouput.)

Task #1

cat password.txt

- Was that a fun file to watch go by?
- Its Big

Task #2

less password.txt

- Its Big
- When your bored of checking it out
- Hit q to quit

Task #3

fgrep "123456" passwords.txt

- What did that do?
- Explain what just happened.

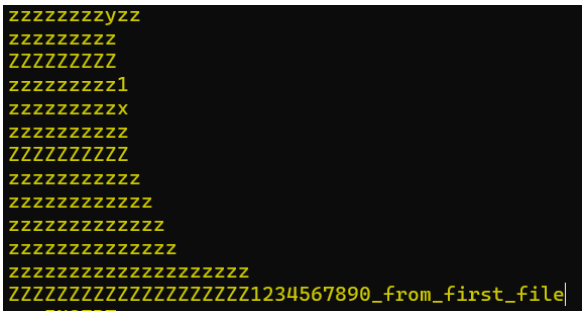
Exercise 17

Let's add a nice little thing at the very end of these sets of exercises.

Task #1

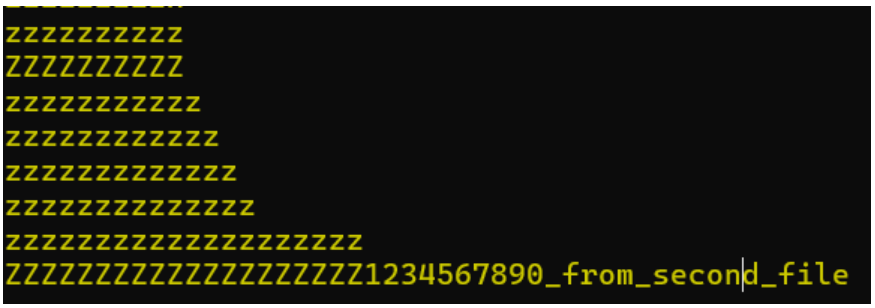
vim passwords.txt

- Add "1234567890_from_first_file" to the end of the file after the zzz not a new line.



Task #2

- Create a copy of the first file, as passwords2.txt
- Edit the second file.
    - Change the last line to "1234567890_from_second_file"



Task #3

fgrep 1234567890_ passw*

- What did that do?
- Explain what just happened.

Exercise 18

Task #1

fgrep 1234567890_ passwords.txt > passwords3.txt

- What did that do?
- Explain what just happened.

Task #2

cat passwords3.txt

- What did that do?
- Explain what just happened.

Task #3

fgrep -c 1234567890_ passw*

- What did that do?
- Explain what just happened.

Task #5

fgrep -w 1234567890_ passw*

- What did that do?
- Explain what just happened.

Task #6

fgrep -n 1234567890_ passw*

- What did that do?
- Explain what just happened.

The `grep` command in Linux is used for searching text using patterns. When used with the `-E` option, `grep` interprets the pattern as an Extended Regular Expression (ERE). This mode enhances `grep`'s capabilities, allowing it to understand a broader set of regex constructs without needing to escape them, which is often required in Basic Regular Expressions (BRE).

Syntax of `grep -E`:

```
grep -E pattern [file...]
```

Extended Regular Expressions (ERE)

Extended Regular Expressions introduce additional regex features and simplify the syntax for others. For example, in ERE:

- Parentheses `()` for grouping and pipes `|` for alternation can be used without backslashes.
- Plus `+` and question mark `?` quantifiers, and curly braces `{}` for specifying exact or ranged occurrences, are available directly.

Common Use Cases for `grep -E`

- Searching for lines that match one of several patterns.
- Finding patterns where a certain character or sequence repeats a specific number of times.
- Complex pattern matching that involves optional elements and alternations.
- **Hyphen Character**: Ensure you're using the correct hyphen - for options (not an en dash – or em dash —), which can happen due to auto-formatting in some text editors.
- **Quotation Marks Around the Pattern**: When using shell commands, it's a good practice to enclose regex patterns in quotes to prevent the shell from interpreting special characters. Without quotes, characters like `{`, `}`, `[`, `]`, and others might be interpreted by the shell or not passed correctly to `grep`.
- **Regex Syntax**: Ensure the regex accurately represents what you're trying to match. For example, if you're looking to match exactly ten digits followed by "_from", the pattern looks correct, but it needs to be enclosed in quotes.

```
grep -E '[0-9]{10}_from' passwords.txt
```

PROTIP: Enclosing the pattern in single quotes `'` prevents the shell from interpreting the curly braces and other special characters, ensuring `grep` receives the pattern exactly as intended.

Exercise 19

Task #1

grep -E "[0-9]{10}_from" passwords.txt

- What did that do?
- Explain what just happened.

Task #2

grep -E "[0-9]{10}_from" passwords*.txt

- What did that do?
- Explain what just happened.

Exercise 20

Task #1

- Using authy.log determine why there was a service interruption on our website.
- Explain what just happened.

Task #2

- Using authy.log determine by who was this done?
- Explain what just happened.

Task #3

- Using authy.log determine from where was this done from?
- Explain what just happened.