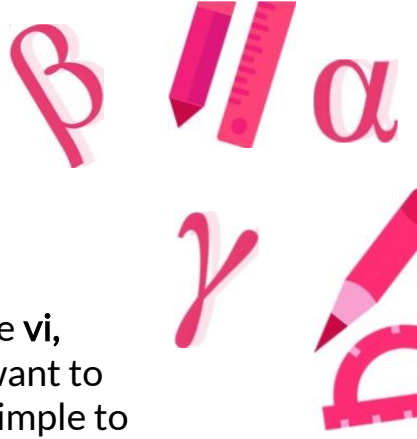
MODULE 5 :
Regular Expressions



Regular Expressions in Unix

- Regular expressions can be used with text processing commands like **vi, grep, sed, awk,** and others. Regular expressions are used when you want to search for specific lines of text containing a particular pattern. It is simple to search for a specific word or string of characters. Almost every editor on every computer system can do this.

- Regular expressions are more powerful and flexible. You can search for words of a certain size.

- Regular expressions are shortened as '**regexp**' or '**regex**'.

Regular Expressions in Unix (contd.)

Some of the commonly used commands with Regular expressions are tr, sed, Vi and grep. Listed below are some of the basic **Regex**.

Symbol	Descriptions
.	matches any character
^	matches start of string
$	matches end of string
*	matches up zero or more times the preceding character
\	Represent special characters
()	Groups regular expressions
?	Matches up exactly one character

grep Command in Unix

The ***grep*** filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

```
grep [options] pattern [files]
```

`-c` : This prints only a count of the lines that match a pattern

`-h` : Display the matched lines, but do not display the filenames.

`-i` : Ignores, case for matching

`-l` : Displays list of a filenames only.

`-n` : Display the matched lines and their line numbers.

Grep Command in Unix(contd.)

```
grep [options] pattern [files]
```

-v : This prints out all the lines that do not matches the pattern

-e exp : Specifies expression with this option. Can use multiple times.

-f file : Takes patterns from file, one per line.

-w : Match whole word

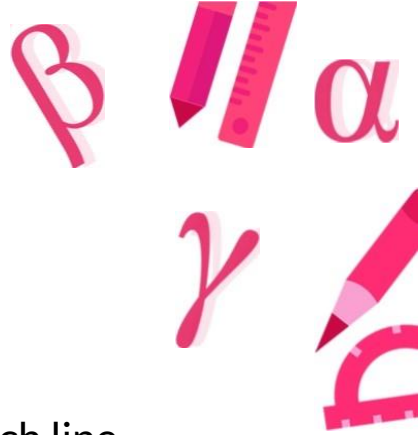
-o : Print only the matched parts of a matching line, with each such part on a separate output line.

grep with Regular Expressions

grep can also use regular expressions to match a group of similar patterns.

- `[ ]`: Matches any one of a set characters
- `[ ] with hyphen`: Matches any one of a range characters
- `^`: The pattern following it must occur at the beginning of each line
- `^ with [ ]` : The pattern must not contain any character in the set specified

grep with Regular Expressions(contd.)



- `$`: The pattern preceding it must occur at the end of each line
- `. (dot)`: Matches any one character
- `\ (backslash)`: Ignores the special meaning of the character following it
- `*`: zero or more occurrences of the previous character
- `(dot).*`: Nothing or any numbers of characters.

Wildcards

Wildcards are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories.

Here is the basic set of wildcards:

- `*` - represents zero or more characters
- `?` - represents a single character
- `[]` - represents a range of characters

Shell Metacharacters

The command options, option arguments and command arguments are separated by the space character. However, we can also use special characters called metacharacters in a Linux command that the shell interprets rather than passing to the command.

Shell Metacharacters (contd.)

Symbol	Meaning
>	Output redirection
>>	Output redirection (append)
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; one character
[ ]	File substitution wildcard; any character between brackets
`cmd`	Command Substitution
$(cmd)	Command Substitution
|	The Pipe (|)
;	Command sequence, Sequences of Commands
[ ]	File substitution wildcard; any character between brackets

Shell Metacharacters (contd.)

Symbol	Meaning
||	OR conditional execution
&&	AND conditional execution
( )	Group commands, Sequences of Commands
&	Run command in the background, Background Processes
#	Comment
$	Expand the value of a variable
\	Prevent or escape interpretation of the next character
<<	Input redirection

How to Avoid Shell Interpretation of Metacharacters

Sometimes we need to pass metacharacters to the command being run and do not want the shell to interpret them. There are three options to avoid shell interpretation of metacharacters.

- Escape the metacharacter with a backslash (\).
- Use single quotes (' ') around a string. Single quotes protect all characters except the backslash (\).

How to Avoid Shell Interpretation of Metacharacters(contd.)

- Use double quotes (" "). Double quotes protect all characters except the backslash (\), dollar sign ($) and grave accent (`). Double quotes is often the easiest to use because we often want environment variables to be expanded.

Thank You!