School of Computer Science Engineering and Technology
Bennett University (The Times Group)

Program- BTech-3rd Semester
Course Code- CSET213
Year- 2025
Date- 14/10/25

Type- Sp. Core-I
Course Name-Linux and Shell Programming
Semester- Odd
Batch- Cyber Security (B1-B14)

Lab Assignment 11

Exp No	Name	CO1	CO2	CO3	CO4
11	Grep its use and commands. Using of Grep with pipe and filters	✓	✓		

Objective: To understand and practice the grep tool for searching patterns, filtering text, and using regular expressions in shell scripts in a pipe chain.

Outcomes: After executing this assignment, the students will be able to

- Implement pattern matching in shell scripts using meta characters
- Regular expression
- Implement a pipe chain using

Hands-on Learning on operators (60 minutes)

Why grep?

- For Searching a pattern/data in a large file
- `$grep [options] pattern [file]`
- To see if a particular process is running or not. For this purpose, we must use ps command in combination with the grep command. For e.g., you want to see whether Apache web server process is running or not then give command:

```
$ ps ax | grep httpd
```

- grep is a Unix utility that searches through either information piped to it or files in the current directory.
- egrep is extended grep, same as grep -E
- Use zgrep for compressed files. Usage: grep

Commonly used grep options:

- -i : ignore case during search
- -r : search recursively
- -v : invert match i.e. match everything except pattern
- -l : list files that match pattern
- -L : list files that do not match pattern
- -n : prefix each line of output with the line number within its input file.
- -A num: print num lines of trailing context after matching lines.
- -B num : print num lines of leading context before matching lines.

What is a regular expression (regex)?

- Combination of pattern and metacharacters is known as regular expressions. Regular expressions are used by different Linux utilities like ***grep, awk, and sed***
- A regular expression (regex) is a method of representing a string-matching pattern.
- Regular expressions enable strings that match a particular pattern within textual data records to be located and modified and they are often used within utility programs and programming languages that manipulate textual data.
- Regular expressions are extremely powerful.
- Supporting Software and Tools are:
    1. Command Line Tools: ***grep, egrep, sed***
    2. Editors: ***ed, vi, emacs***
    3. Languages: awk, perl, python, php, ruby, tcl, java, javascript, .NET

How to implement pattern matching in shell scripts using meta characters?

Character	Meaning
.	Match any *single* character except newline.
*	Match any number (or none) of the single character that immediately precedes it. Thepreceding character also can be a regular expression (e.g., since **.** (dot) means any character, **.*** means *match any number of any character* -- except newlines).
^	Match the beginning of the line or string.
$	Match the end of the line or string.
[ ]	Match any *one* of the enclosed characters. A hyphen (**-**) indicates a range of consecutivecharacters. A circumflex (**^**) as the first character in the brackets reverses the sense: it matches any one character *not* in the list. A hyphen or close bracket (**]**) as the first character istreated as a member of the list. All other metacharacters are treated as members of the list.
[^]	Match anything except enclosed characters.
\{*n*,*m*\}	Match a range of occurrences of the single character that immediately precedes it. The preceding character also can be a regular expression. \{*n*\} matches exactly *n* occurrences, \{*n*,\} matches at least *n* occurrences, and \{*n*,*m*\} matches any number of occurrencesbetween *n* and *m*.
{*n*,*m*}	Like \{*n*,*m*\}. Available in **grep** by default and in **gawk** with the **-Wre-interval** option.
\	Turn off the special meaning of the character that follows.
\(\)	Save the matched text enclosed between \(and \) in a special holding space. Up to nine patterns can be saved on a single line. They can be "replayed" in the same pattern or withinsubstitutions by the escape sequences \1 to \9.
n	Reuse matched text stored in *n*th \(\).
()	In **egrep** and **gawk**, save the matched text enclosed between \(and \) in a holding space to bereplayed in substitutions by the escape sequences \1 to \9.
\<\>	Match the beginning (\<) or end (\>) of a word.
+	Match one or more instances of preceding regular expression.

?	Match zero or one instance of preceding regular expression.
|	Match the regular expression specified before or after.
()	Group regular expressions.

Many utilities support POSIX character lists, which are useful for matching non-ASCII characters inlanguages other than English. These lists are recognized only within **[]** ranges. A typical use would be **[[:lower:]]**, which in English is the same as **[a-z]**. The following table lists POSIX character lists:

Notation	Action
[:alnum:]	Alphanumeric characters
[:alpha:]	Alphabetic characters, uppercase and lowercase
[:blank:]	Printable whitespace: spaces and tabs but not control characters
[:cntrl:]	Control characters, such as ^A through ^Z
[:digit:]	Decimal digits
[:graph:]	Printable characters, excluding whitespace
[:lower:]	Lowercase alphabetic characters
[:print:]	Printable characters, including whitespace but not control characters
[:punct:]	Punctuation, a subclass of printable characters
[:space:]	Whitespace, including spaces, tabs, and some control characters
[:upper:]	Uppercase alphabetic characters
[:xdigit:]	Hexadecimal digits

The following characters have special meaning in replacement patterns

Character	Meaning
\	Turn off the special meaning of the character that follows.
\n	Restore the *n*th pattern previously saved by \(and \). *n* is a number from 1 to 9, matching thepatterns searched sequentially from left to right.
&	Reuse the search pattern as part of the replacement pattern.
~	Reuse the previous replacement pattern in the current replacement pattern.
\e	End replacement pattern started by \L or \U.
\E	End replacement pattern started by \L or \U.
\l	Convert first character of replacement pattern to lowercase.
\L	Convert replacement pattern to lowercase.
\u	Convert first character of replacement pattern to uppercase.
\U	Convert replacement pattern to uppercase.

Examples of Searching

When used with grep or egrep, regular expressions normally are surrounded by quotes to avoid interpretation by the shell. (If the pattern contains a $, you must use single quotes, as in '$200', or escape the $, as in "\$200".) When used with ed, vi, sed, and awk, regular expressions usually are surrounded by / (although any delimiter works). Here are some sample patterns:

Pattern	What does it match?
bag	The string *bag*.
^bag	"bag" at beginning of line or string.
bag$	"bag" at end of line or string.
^bag$	"bag" as the only text on line.
[Bb]ag	"Bag" or "bag."
b[aeiou]g	Second character is a vowel.
b[^aeiou]g	Second character is not a vowel.
b.g	Second character is any character except newline.
^...$	Any line containing exactly three characters.
^\.	Any line that begins with a dot.
^\.[a-z][a-z]	Same, followed by two lowercase letters (e.g., troff requests).
^\.[a-z]\{2\}	Same as previous, **grep** or **sed** only.
^[^.]	Any line that doesn't begin with a dot.
bugs*	"bug," "bugs", "bugss", etc.
"word"	A word in quotes.
"*word"*	A word, with or without quotes.
[A-Z][A-Z]*	One or more uppercase letters.
[A-Z]+	Same, **egrep** or **awk** only.
[A-Z].*	An uppercase letter, followed by zero or more characters.
[A-Z]*	Zero or more uppercase letters.
[a-zA-Z]	Any letter.
[0-9A-Za-z]+	Any alphanumeric sequence.

School of Computer Science Engineering and Technology
Bennett University (The Times Group)

Scripting Problems for Assessment (60 Minutes)

1. Write a shell script using series of grep statements to do the following using the file Lab11data.txt: (30 Minutes)
    a. Print all lines that contain a phone number with an extension (the letter x or X followed by four digits).
    b. Print all lines that begin with three digits followed by a blank.
    c. Print all lines that contain a date.
    d. Print all lines that do not begin with a capital S.
2. Write a shell script using grep with pipe to do the following: (20 Minutes)
    a. Make a pipe that counts number of files/directories (including hidden files) in your directory
    b. Search directories out of ls -l
    c. Using pipes and commands echo/tr/uniq, find duplicate words in a file.
3. Write a shell script using grep with pipe to do the following using the file Lab11data.txt
    a. Find all occurrences of the word '2013'.
    b. Search for the word '@example.com' in the file ignoring case sensitivity.
    c. Display how many times the word '@example.com' appears in the file.
4. Write a shell script using grep with pipe to do the following using the file Lab11data.txt
    a. Show all lines that begin with a number.
    b. print all lines containing email addresses.
    c. Find lines that end with a period (.).
    d. Print all lines that contain a 4-digit year (e.g., 2013).
5. Write a shell script finduser.sh that accepts a username as an argument and checks whether that user exists in /etc/passwd.
6. Write a script errorlog.sh that takes a log file as input and prints all lines containing ERROR, WARNING, or FAILURE.
7. Write a shell script search.sh that asks the user for a filename and a keyword, then uses grep to display all matching lines.

Submission Instructions:

1. Submission requires the screen shots of all the incurred steps to execute a shell script or a video showing the whole process.
2. All these files are in single zip folder.
3. Use the naming convention: Prog_CourseCode_RollNo_LabNo.docx (Example: BTech3rdSem_CSET213_ E21CSEU002_Lab1.1)
4. Submission is through LMS only