School of Computer Science and Engineering

Program- BTech-3rd Semester
Course Code- CSET213
Year- 2024
Date- 21/10/2024

Type- Sp. Core-I
CourseName-Linux and Shell Programming
Semester- Odd
Batch- BXX-BXX (Cyber Security)

Lab Assignment 8

Exp No	Name	CO1	CO2	CO3
8	Shell programming environments- filters and pipe Grep its use and commands. Using of Grep with pipe and filters	✓ -	✓	-

Objective:
1. To learn the usage of pipes and filters by executing some interesting examples.
2. To understand the implementation of use of grep in a pipe chain

Outcomes: After hands-on, you can differentiate between redirection and pipe. Moreover, you can apply the concept of pipe when the output of one command can be the input of another command, and we need to do it without storing the output of the former command in a temporary file. In addition, the student will use regular expression, implement pattern matching in shell scripts using metacharacters, and implement a pipe chain for developing the application.

Hands-on Learning on pipes and filters (60 minutes)

Redirecting Terminal I/O: Whenever a shell command runs, three standard file streams are opened like **"standard in (stdin)" for input; "standard out (stdout)"; and "standard error(stderr)" for output.** The shell feature known as redirection is used to assign files other than the user terminal to a standard stream. The formats for redirection are:

<*command*> < <*file-name*>	[<*command*> gets input from <*file-name*> rather than the terminal]
<*command*> > <*file-name*>	[<*command*> sends output to <*file-name*> rather than the terminal]
<*command*> >> <*file-name*>	[<*command*> appends output to <*file-name*>]

Examples:

- sort > mysortedfile // instructs the sort command to sort data from the **standard input** (stdin) and redirect the sorted output to a file named mysortedfile.
- sort file1 file2 > mysortedfile // This sorts the **combined contents** of file1 and file2 and stores the sorted result in mysortedfile.
- sort 0<filein 1>mysortedfile 2>errlist
- sort filein > mysortedfile 2> errlist
- sort filein >mysortedfile 2>&1
- (date; cal) > myfile

1. Why Pipes?

Redirection provide a shell command with an alternative to the user terminal for standard input, standard output, or standard error. Redirection can’t be used to take standard output and turn it directly into standard input for another command without going through an intermediate file.

Let us consider the example shown below. The objective of doing "sort < ls -l" would need to be accomplished by

```
ls -l > temp

sort < temp
```

Example1:

By the use of pipe (|), the shell provides a means of taking standard output from one command and making it standard input for another without using any temporary file.

The notation of pipe is:

<command-1> | <command-2> **(1)**

Equation-1 is used to denote that standard output from <command-1> is to be piped to <command-2> as standard input.

For the example 1, using pipe, you can simply execute:

ls -l | sort

2. Filters

A filter is a command that takes data from a file and performs some simple transformation on it, the result of which is sent on to some other file.

Examples of filter:

sort - sort files

grep (and its derivatives) - search for keyword information in the file

head - output lines from the front end of the file

tail - output lines from the tail end of the file

wc - count words, lines, and/or characters in the file

2.1. 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?

- ^ (caret) match beginning of line. Anchors match.
- $ (dollar sign) match end of line. Anchors match.
- . (dot) match any character. Beware, command line globbing uses? instead.
- * (star) matches zero or more of preceding characters. Beware, command line uses * as in.*.
- [] (square brackets) set of characters inside braces, match any one-off.
- [^] (carat at first character inside braces), match any character except those inside braces
- [a-z] (use of dash inside braces) match a range. If - is to be matched, must be first character, to avoid misinterpretation as range operator.
- () {parenthesis, must be escaped with backslash), save match for later use with \n, where n is a number.
- {m}, {m,} and {m,n} (braces, which must be escaped with a backslash), matched m, more than m, or between m and n repetitions of preceding character.
- & (ampersand) expands to the matched string, used in sed.

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.

2.1.1 Metacharacters used with grep

Metacharacters are used for the mechanisms employed by regular expressions to represent complex patterns; e.g.

^	for the beginning of a line	'^t' (lines beginning with "t")
$	for the end of a line	't$' (lines ending with "t")
.	matches any single character	'^.t' (lines with 1st character anything and 2nd character "t")
*	goes with the preceding character to represent 0 or more repetitions of the character	
+	is like *, but is for 1 or more repetitions	
\	turns off any special meaning for the character that follows it	

2.1.2 Construction rules for regular expressions provide means for using simpler regular expressions to define more complex regular expressions. Example:

```
[...] match is a regular expression: match is to any character listed; allows ranges such as a-x.
[^...] not match is a regular expression: match is to any character not listed; also allows ranges.
```

2.2 head

Example: **head -20 myFile.txt** //lists the first 20 lines of myFile.txt
Note: If no number is specified, the default is 10.

2.3 tail

Example: **tail -20 myFile.txt** //lists the last 20 lines of myFile.txt
Note: If no number is specified, the default is 10.

2.4 wc

Example: **wc -lwc myFile.txt** //counts lines, words, and characters in the file.

Note: If any one of l, w, or c is omitted, that count is not provided.

Scripting Problems for Assessment (60 Minutes)

1. Create a file name “moviefileName.txt” having the names of the Science Fiction movies.
   i. Consider some redundant data is present in the file. So, remove redundancy and store that in moviefileName_RemovedRedundancy.txt
   ii. The data in moviefileName_RemovedRedundancy.txt is unordered. So, order them in ascending order and preserve output in moviefileNameSortedInAscending.txt.
   iii. Similarly, do the same in descending order and preserve output in moviefileNameSortedInDescending.txt.
   iv. Display the movie names having alphabets ‘d’ and ‘e’ combined in the word. Example “Spi**de**rhead” (15 Minutes)
2. Create a file with name “bigdata.csv”. Keep 10000 numeric values in it. You need to show and preserve in stdout and file respectively:
   i. First 50 values of 10000 data
   ii. Last 50 values of 10000 data

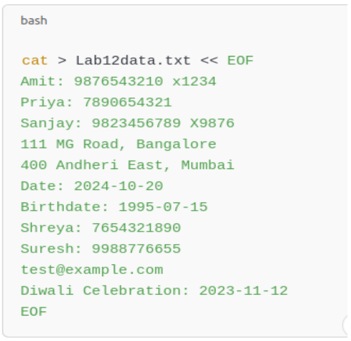
   Note: You have to create a file using Linux command. You need to put 10000 random numeric data using the code that can be in any language. (20 Minutes)
3. Display all the output including hidden files of /etc at a time on screen. Along with that give the total number of files present in /etc. (10 Minutes)
4. Display all the link files present in the current directory. (10 Minutes)

   Note: You have to create 100 subdirectories and create links for each of them and store it in different directory (let us suppose Desktop). So, in total you must have 100 links in the Desktop directory.
5. Fetch all the files (with an extension .bak) in the system. (5 Minutes)

Grep Questions:

1. Write a shell script using series of grep statements to do the following using the file Lab12data.txt:
   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.

   Note: Create the file Lab12data.txt using:



bash

```bash
cat > Lab12data.txt << EOF
Amit: 9876543210 x1234
Priya: 7890654321
Sanjay: 9823456789 X9876
111 MG Road, Bangalore
400 Andheri East, Mumbai
Date: 2024-10-20
Birthdate: 1995-07-15
Shreya: 7654321890
Suresh: 9988776655
test@example.com
Diwali Celebration: 2023-11-12
EOF
```

2. Write a shell script using grep with pipe to do the following:

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.

Submission Instructions:

1. Submission requires the screen shots of all the incurred steps to execute a shell script.
2. Use the naming convention: Prog_CourseCode_RollNo_LabNo.docx.
   Submission is through LMS only