Unit 12 : Pipes and Filters

Lesson 1 : Standard files and Redirection

1.1. Learning Objectives

On completion of this lesson you will be able to know:

- ❖ About standard input, output and error files
- ❖ How the assignments for standard input, output and error will be redirected

1.2. Standard files:

Standard input file:

In Linux, when a user executes a command that requires input, the shell interprets the command and assigns the keyboard as the default source of input. The keyboard is referred to as the *standard input file.*

Let us take an example of the cat command. When followed by a file name, all the lines in the file are displayed. Without the file name, however, the cat command takes its input from the standard input File as shown below:

```
$ cat < Enter>
```

The cat command waits for input from the keyboard. As you enter characters from the keyboard and press **<Enter>,** the characters are displayed on the screen as shown below:

```
$ cat < Enter>
Oyonti is the student of DCSA <Enter>
Oyonti is the student of DCSA
_                               [The cursor waits on the next line]
```

You can enter as many lines as you wish. To indicate the end of input, you have to press the **<Ctrl> d** keys. The **$** prompt then appears on the screen.

Standard input and output file.

Not all commands expect input. For example, the **cd** command does not use the standard input file.

In Linux, all open files, including the standard files, are assigned a number called the file descriptor. The file descriptor, 0, is assigned to the standard input file.

Standard output file:

In Linux, the shell assigns the monitor as the default destination for the output of any command that it executes. The monitor is referred to as the standard output file.

For example, when you issue the command,

```
    $ 1s
```

The shell executes the command and sends its output – the directory listing – to the standard output file.

Not all commands generate output, For example, the **mkdir** command does not use

the standard output file.
The file descriptor, 1, is assigned to the standard output file.

Standard Error file:

Shell utilities display error messages on the monitor. There could be many reasons for error messages, such as typing an invalid command, or a command for which the user does not have permission. The monitor is thus also the standard error file.
For example, the **cat** command followed by a file name that does not exist will generate an error message on the monitor. Another example could be the command, **cp** file1 dir1, which displays an error message on the standard error file if you do not have write permission on dir1. or read permission on file1. Although the command is correct, this will generate an error since you may not have permissions on the file.
The file descriptor, 2, is assigned to the standard error file.

1.3. Redirection:

Redirection changes the assignments for the standard input, standard output, and standard error. Using redirection, the input to a command can be taken from a file other than the keyboard. Similarly, the output of a command or the error can be written to a disk file or printed, instead of the monitor. The three types of redirection are:

- Input redirection
- Output redirection
- Error redirection

Input Redirection:
The following example illustrates the usage of input redirection:

```
$ cat < test 1 <Enter>
```

Here, the less than symbol, <, implies input redirection from the file, test1. The cat command will take each line of the file, test1, as input and display it on the monitor. Thus, it is possible to specify that the standard input, instead of coming from the standard input file, comes from a disk file.
For better clarity, the above command can also be written using the file descriptor as:

```
$ cat 0 < test1 <Enter>
```

Here, 0 indicates input redirection.

Output Redirection:
The following example illustrates the usage of output redirection:

```
$ cat test1 > test2
```

Here, the greater than symbol, >, implies redirection of output to the file, `test2`. The output of the cat command is written to a disk file, `test2`.
In output redirection, the file to which the output is redirected is first created on the disk as an empty file and then the output is sent to this file. However, if the file already exists, its contents are deleted before the output is written to it.
If you want to append the output to the file, test2, the command is:

```
$ cat testl >> test2
```

The previous commands can also be written using the file descriptor as:

```
      $ cat test1 1 > test2        and
      $ cat testl l >> test2
```

Here, 1> indicates output redirection. Although the file descriptor, 1, is not necessary, it can be included for better clarity.

Error Redirection:

The following example illustrates the usage of error redirection:

```
      $ cat test 2> error-mesg <Enter>
```

Assume that the file, test, does not exist in the current directory. So, when a user tries to execute this command, Linux will generate an error message since the execution is unsuccessful. This message, which would otherwise be displayed on the monitor (the standard error file), will now be written to the file, error-mesg. As in the case of output redirection, error redirection also first creates the file to which the error messages are redirected and then writes the error output to the file.

1.4. Exercises

1.4.1. Multiple choice questions

a. The **cat** command takes its input from

(i) standard input file
(ii) standard output file
(iii) input redirection
(iv) output redirection.

b. **`mkdir`** does not use

(i) standard input file
(ii) standard output file
(iii) standard error file
(iv) none of above.

c. How many types of redirection are there?

(i) Three
(ii) Four
(iii) Five
(iv) Six.

1.4.2. Questions for short answers

a. What is the device considered as standard input file?
b. Mention the device name considered as standard error file.
c. Classify redirection.

1.4.3. Analytical questions

a. Explain the concept of standard input file with example.
b. Explain output redirection with example.

Lesson 2 : Filters

2.1. Learning Objectives

On completion of this lesson you will be able to know:

- ❖ About filters
- ❖ About different types of filters

2.2. Definition of filters

Definition of filters

A filter is a program that takes its input from the standard input file, processes (or filters) it, and sends its output to the standard output file, Linux has a rich set of filters that can be used to work on data in an effective way. Some examples of filters are grep, cat, wc, tr, and cut.

2.3. Classification of filters

Classification of filters

The grep filter
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).

Syntax
grep regular_expression [filename]
The file name is optional in the grep command. Without a file name, grep expects standard input. As a line is input, grep searches for the regular expression in the line and displays the line if it contains that particular expression. Execution stops when the user indicates end of input (by pressing **<CtrI>d**).

Specifying Regular Expressions
Regular expressions can be used to specify simple patterns of characters to highly complex ones.
Some simple patterns are shown in the following table:

Regular	Expression Pattern
'A'	The character A
'F'	The character F
'New'	The word New

Simple Regular Expressions

The wc filter

The **wc** filter is used to count the number of lines, words, and characters in a disk file or in the standard input.

Syntax

```
     wc [-lwc] [filename/s]
```

Example

```
     $    wc    test  <Enter>
     2    7     29
```

The file test has 2 lines, 7 words, and 29 characters.

Option	Function
-1	Displays the number of lines
-w	Displays the number of words
-c	Displays the number of character

Options of the wc filter

The cut filter

The **cut** filter is useful when specific columns from the output of certain commands (such as ls, who) need to be extracted.

Syntax

```
    cut [options [filename/s]
```

Option	Function
-f<colum_number (s)>	Displays the specified columns
-c<character_number(s)>	Displays the specified characters
-d <colum_delimiter>	Specifies the column delimiter

Options of the cut filter

Example

```
$ cut -cl-5 /etc/passwd
```

This command will display the first five characters from the file, `/etc/passwd`

The tr filter

The **tr** filter can be used to translate one set of characters to another.

Example

```
$ tr "[a-z]" "[A-Z]"
We are an independent nation
WE ARE AN INDEPENDENT NATION
```

The above command converts all lower-case letters to upper-case

The sort filter

The **`sort`** filter arranges each line of the standard input in ascending order.

Example

```
$ sort
Odroho
Nazrul
Oyonti
Meherun
Ruhul
```

```
 Odroho
Adnan
 <Ctrl> d
Adnan
Odroho
Meherun
Nazrul
Odroho
Oyonti
Ruhul
$ -
```

2.4. Exercises

2.4.1. Multiple choice questions

a. The **grep** filter

(i) Searches a file for particular pattern of characters, and displays all lines that contain that pattern.
(ii) Is useful when specific column from the output of certain commands need to be extracted.
(iii) Is used to count the number of lines
(iv) Can be used to translate one set of characters to another.

b. Option –w is used in wc filter

(i) To display the number of lines
(ii) To display the number of words
(iii) To display the number of characters
(iv) To specify the column delimiter.

c. –f <column_number(s)> option of the cut filter is used

(i) To display the specified characters.
(ii) To display the specified columns.
(iii) To specify the column delimiter.
(iv) To display the number of lines.

2.4.2. Questions for short answers

a. Define filters.
b. Write down the options of wc filter.
c. What does tr filer do?

2.4.3. Analytical questions

a. Explain grep filter with it's expression pattern.
b. Write down the purpose of sort filter with example.

Lesson 3 : Pipes

3.1. Learning Objectives

On completion of this lesson you will be able to know:

- ❖ A method used to pass information from one program process to another (Pipes).

3.2. Pipes:

A pipe is a method used to pass information from one program process to another.

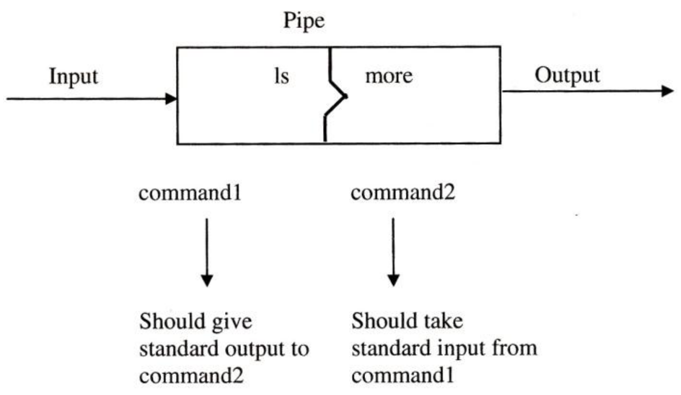
A pipe is a method used to pass information from one program process to another. Linux has a feature by which filters and other commands can be combined in such a way that the standard output of one filter or command can be sent as standard input to another filter or command.
For example, to display the contents of the current directory, a screen-full at a time, you can give the following commands:

```
$ ls > tempfile
$ more tempfile
```

Here, the listing of the directory is stored in the file, tempfile, by the first command. This file is then used as input by the more command.
Through the Linux pipe feature, these two steps can be combined and executed as a single command, without creating a temporary file, as shown below:

```
$ ls | more
```

The vertical bar (|) is the pipe character, which indicate to the shell that the output of the command before the ' | ' is to be sent as input to the command after the '|'. (Note that on the keyboard, the pipe character appears as a broken vertical line).
Another advantage of the pipe feature is that utilities do not have to be rewritten to perform complex tasks. Instead, the Linux tools (command) can be combined. There is no limit to the number of filters or commands in a pipe. Consider the example in the figure below to understand how a pipe works.


Pipe
Input
ls
more
Output
command1
command2
Should give standard output to command2
Should take standard input from command1

Working of a Pipe

Examples

The following is a sample output of the **Is -1** command:

```
-rw-rw-r--      1   Odroho   Odroho     1384    nov  26  06:32   DCSA
drwxr-xr-x      5   Odroho   Odroho     1024    nov  22  23:30   Desktop
drwx------      2   Odroho    BDTECH    1024    nov  24  09:00   Mail
-rwxr-xr-x      1   root     root       12691   nov  28  01:46   a.out
drwx------      2   Odroho   BDTECH     1024    nov  29  23:20   mail
drwx------      2   Odroho   BDTECH     1024    nov  30  22:21   nsmai1
-rw-rw-r--      1   Odroho   Odroho     58      nov  24  03:44   program.cc
-rwxrwxrwx      1   Odroho   BDTECH     691     nov  24  23:55   test
```

Some examples of the pipe feature, based on the above output, are given below.

Examples

1. To display the names of all the ordinary files in the current directory for which the File Owner has execute permission, you need to give the following command:

```
$ Is -1 | grep "^-. . x" | tr -s " " | cut -d" " – f9
```

Output

```
out
test
```

Here, the output of the **ls -1** command is given as input to the grep command. The **ls – 1** command gives a detailed listing of the files in your current directory. The **grep** command extracts all the ordinary files that have execute permission. This output is given to the **tr** command, which replaces multiple spaces with a single space. The **cut** command takes this as input and extracts the ninth field, which is the file name, and displays it.

2. To display the file names in the current directory along with the file sizes, a screen-full at a time, you need to give the command:

```
$ Is -1 | tr -s " " | cut – d " " – f 9, 5
```

Output

```
1384 DCSA
1024 Desktop
1024 Mail
12691 a.out
1024 mail
1024 nsmail
58 program.ee
691 test
```

3. If you want to display the names of all ordinary files in the current directory, the command will be:

```
$ 1s -1 | grep "^_" | tr -s " " | cut -d" " -f9
```

Output

```
DCSA
a.out
program.ee
test
```

4. The command to display the total number of files in your current directory is:

```
$ 1s | we -1
```

Output

```
        8
```

5. The command to display the number of lines where the word, "and", occurs in a file, test, will be:

```
$ grep -c "and" test
```

Output

```
        4
```

Some common situation involving pipes and filters are given below:

I. Consider the contents of the file named test:

```
Shafiq   25   Male     DP
Shaon    20   Male     RJ
Ruhul    26   Male     BR
Sania    27   Female   GP
Odroho   22   Male     SJ
```

The above file has records in plain text and has fixed length columns. There is a utility, which understands only comma-delimited files. You have to store the contents of this file in another file, in which the delimiter is a comma. The following command accomplishes this.

```
$ tr –s ‘ ’ ‘,’ <test> test2
```

After this command is executed, the content of the file, test2, will be:

I. Consider the contents of the file named test:

Shafiq, 25, Male, DP
Shaon , 20, Male , RJ
Ruhul, 26, Male, BR
Sania, 27, Female, GP
Odroho, 22, Male, SJ

You want to scan the file, tran, and search for the string, "error". You want to display all such occurrences on the screen, page-wise. The following command accomplishes this.

```
$ cat tran | grep "error" |  more
or
$ grep "error" < tran | grep | more
```

3.3. Exercises

3.3.1. Multiple choice questions

a. The '**more'** filter is used

(i) To display text one screen full at a time.
(ii) To display the file names in the current directory along with the file sizes.
(iii) To display the names of all ordinary files in the current directory.
(iv) To display the number of lines.

b. **'$ 1s | we -1'** command is used

(i) To display the names of all ordinary files
(ii) To display text
(iii) To display the name of all ordinary files in the current directory.
(iv) To display the total number of files in you directory.

3.3.2. Questions for short answers

a. Write the function of '**more'** filter.
b. Through diagram explain how a pipe works.

3.3.3. Analytical questions

a. Explain the command '**ls | more'** with the help of diagram.