Filters and Pipes

Session Objectives

- **Input Redirection**
- **Output Redirection**
- **Error Redirection**
- **Filters**
  - **Grep**
  - **Tr**
  - **Wc**
  - **Cut**
  - **Sort**
- **Pipes**
- **Advanced file operations**
  - **Find**
  - **Locate**
  - **Uniq**
  - **Comm**
  - **Cmp**
  - **xargs**

Introduction

The commands we saw in the previous chapters were created to perform single tasks only. If we want to perform multiple tasks in one command, we go in for the concept of redirection. Redirection creates a lot of temporary files, which are redundant and occupy disk space. Pipes and filters are used to overcome this.

To understand the concept of I/O redirection, let us first understand what standard input and standard output mean.

The keyboard is treated as the standard input file in Linux. When a user executes a command that requires input, the shell assigns the keyboard as the default source of input.

The VDU (monitor) is treated as the standard output file in Linux. The output of any command is sent to the monitor.

The VDU is also treated as the standard error file in Linux. The error messages are sent to the monitor.

Input can be taken from sources other than the Standard Input and output can be passed to any source other than the Standard Output. This is called Redirection. Redirection can be of three types:

- Input redirection
- Output redirection
- Error redirection

Input redirection

Input redirection implies taking input from a source other than the keyboard. The syntax for input redirection is

```
$ command<file
```

In the above case, file is the input source. If the file does not exist, the shell will issue an error and abort the operation.

The following command takes input from the file demo and displays the contents on the monitor.

```
$ cat<demo
rose
lotus
marigold
$_
```

Output redirection

Output redirection implies redirecting output to a source other than the monitor. The syntax for output redirection is

```
$ command >file
```

In the above case, the output of the command is sent to the file instead of to the monitor. If the file does not exist, it is created on the disk as an empty file and the output is sent to the file. If the file already exists, its contents are deleted before the output is written to it. If we do not want the existing contents to be deleted but new contents to be only appended, we must go for the following command:

```
$ command >>file
```

For example,

```
$ cat  demo>test
```

The above command sends the output of the cat command to the file test. Now display the contents of test:

```
$ cat test
rose
lotus
marigold
$_
```

I/O redirection

It is possible to redirect both the standard input and standard output for a command. The syntax is

```
$ command<source>destination
```

For example,

```
$ cat<demo>example
```

The above command takes input from demo file and passes it to the example file.

```
$ cat example
rose
lotus
marigold
$_
```

Error redirection

Consider the following command

```
$ cat test>mesg
```

If the file `test` does not exist, the shell prints an error message on the screen. But in the above case, the error message is written to a file `mesg` instead of on the monitor.

Instead if the command is

```
$ cat  test 2>errfile
```

The error message of the system is not displayed on the standard output i.e., the VDU but is redirected to errfile. On displaying the contents of errfile the error message of the system can be found.

Filters

A filter is a program that takes input from the Standard Input, processes it and then sends the output to the Standard Output. Filters can also take input from a file and send the output to another file.

Filters are used to perform the following:

- Extract lines containing a specific pattern
- Sort the contents of a file
- Replace existing characters with other characters

The various filters available in Linux are:

- `grep`
- `wc`
- `cut`
- `tr`
- `sort`

Let us explore each of these filters.

The grep filter

The grep utility is one of the most useful filters in Linux. Grep searches line-by-line for a specified pattern, and outputs any line that matches the pattern. Grep stands for *globally* search for *regular expression* and *print* out.

The basic syntax for the grep command is `grep [-options] pattern [file]`.

If the file argument is omitted, grep will read from standard input. It is always best to enclose the pattern within single quotes, to prevent the shell from misinterpreting the command.

Here are some of the characters you can use to build grep expressions:

- The carat (^) matches the beginning of a line.
- The dollar sign ($) matches the end of a line.
- The period (.) matches any single character.
- The asterisk (*) matches zero or more occurrences of the previous character.
- The expression [a-b] matches any characters that are lexically between a and b.

The various options available with the grep command are given in the table below:

Option	Function
-v	Displays only those lines which do not match the pattern
-c	Displays the count of the lines that match the pattern

-n	Displays the lines that match the pattern along with the line number
-I	Displays the lines that match the pattern ignoring case distinction

Example 1

Consider a file called emp whose contents are as given below

```
$ cat emp
100   ram   india
101   ravi  china
102   raja  america
104   rani  India
$_
```

If we want to display details of employees in india, then

```
$ grep “india” emp
100   ram   india
$_
```

If the case is to be ignored, then

```
$ grep  -i  “india” emp
100   ram   india
104   rani  India
$_
```

Example 2

Consider a file test as shown below:

```
$ cat test
transed is in tnagar
transed offers courses in ecommerce
transed has its centre at adyar
$_
```

To display the lines that end in r

```
$ grep “r$”  test
transed is in tnagar
transed has its centre in adyar
$_
```

To display the lines that begin with t

```
$ grep “^t”  test
transed is in tnagar
transed offers courses in ecommerce
transed has its centre in adyar

$_
```

The wc filter

The wc filter can be used to count the number of lines, words or characters in a file. The syntax is

```
$ wc [-lwc] [filename]
```

For example,

```
$ wc test
3 16 88
```

The above command gives the count of the number of lines, words and characters in the file demo. To count only characters, words or lines, we can set the –c, - w or –l option respectively.

```
$ wc –l test
3
```

If no name is provided to the wc filter, it takes input from the standard input,

```
$ wc
transed is in Tnagar
Ctrl+d
1 4 20
$_
```

The Cut filter

The cut filter can be used to extract specific columns from the output of certain commands. The syntax of the cut filter is

```
$ cut [options][filename]
```

The options are

```
-d<column-delimiter>          specifies the column delimiter
-f<column-number>             displays the specified columns
-c<character-number>          displays the specified characters
```

Example 1

Consider the following file:

```
$ cat movie
m001:the mask:sathyam
m002:gladiator:sree
m003:speed:santham
m004:omen:studio5
$_
```

To display only the movie name(second column), the following command can be used

```
$ cut –d”:” –f2 movie
the mask
gladiator
speed
omen
```

Example 2

Consider the following command

```
$ cut –c1-3 demo
```

The above command displays the first three characters from the file demo.

The tr filter

The tr filter is used to translate one set of characters from the standard input to another.

Example 1

Consider the following command

```
$ tr “[a-z]”  “[A-Z]”
This is a test file
THIS IS A TEST FILE
$_
```

The above command converts all lowercase to uppercase.

The tr filter when used with the –s option can be used to squeeze multiple spaces into a single space.

Example 2

```
$ who
abc   tty0  Mar 27 16:54
$_
```

To make the delimiter a single space,

```
$ who >temp

$ tr –s “ “ <temp
abc tty0 Mar 27 16:54
$_
```

Example 3

Consider the following file:

```
$ cat movie
m001:the mask:sathyam
m002:gladiator:sree
m003:speed:santham
m004:omen:studio5
```

```
$_
```

The following command replaces all occurrences of : with a space

```
$ tr “:” ’ ‘ <movie
m001 the mask sathyam
m002 gladiator sree
m003 speed santham
m004 omen studio5
```

The sort filter

The sort filter arranges the input taken from the standard input in alphabetical order.

Example

```
$ sort
arun
raja
kavi
Ctrl+d
arun
kavi
raja
$_
```

The sort filter comes with various options like –r, -f, -n, -b, -t etc. Let us look at these options one by one.

The –n option

The sort filter arranges numbers and alphabets in ascending order according to the ASCII value. But, sometimes the ordering is not correct, for example, the ASCII value of 10 is less than the ASCII value of 2. So 10 will be placed before 2. In order to overcome this problem, we can go in for the –n option of the sort filter.

Example

```
$ sort –n
34
3
23
1
Ctrl+d
1
3
23
34
$_
```

The +pos1 –pos2 option

Consider a file having three columns like empno, ename and location.

```
$ cat emp
100   ram   india
101   ravi  china
102   raja  america
104   rani  singapore
$_
```

These columns are called fields. If we want to sort on any one field, then the +pos1 –pos2 option can be used.

Assume we want to sort employees by their names, then

Example

```
$ sort +1 –2 emp
raja
ram
rani
ravi
$_
```

If we want to sort by the location then,

```
$ sort +2 –3 emp
america
china
india
singapore
$_
```

The –t option

If we use a field separator other than a tab or a blank space, we can specify that in our sort command using the –t option.

Example

Consider the following file emp

```
$ cat emp
100:meera:india
101:prabha:china
102:rajesh:america
104:rani:singapore
$_
```

Now if we want to sort on the country field

Example

```
$ sort –t  “:” +1 –2 emp
102:rajesh:america
101:prabha:china
```

```
100:meera:india
104:rani:singapore
$_
```

The –r option

The –r option does the sorting in reverse alphabetical order.

Consider the following file course

```
$ cat course
java
ecom
wml
xml
Ctrl+d
```

Example

```
$ sort –r course
xml
wml
java
ecom
```

The –f option

The –f option when used with the sort command does the sorting in alphabetical order but ignores the case distinction.

Example

```
$ sort –f
NIIT
aptech
transed
radiant
Ctrl+d

The output will be

aptech
NIIT
radiant
transed
```

The –u option

The sort command when used with the –u option removes duplicate lines and does the sorting.

Example

```
$ sort –u
```

```
100   ram   india
101   ravi  china
102   raja  america
104   rani  singapore
102   raja  america
101   ravi  china
Ctrl+d

The output will be

100   ram   india
101   ravi  china
102   raja  america
104   rani  singapore
```

The –o option

The output of the sort command is generally sent to the standard output. If the sort command is used with the –o option, the output can be sent to the file specified instead the standard output.

Example

```
$ sort –o newemp emp
$_
$ cat newemp
100:meera:india
101:prabha:china
102:rajesh:america
104:rani:singapore
```

The –b option

The –b option can be used to remove leading blank spaces in the input. If blank spaces are present, it will cause problems in the output since the sort command uses the ASCII value for sorting.

Example

```
$ sort –b
 mani
priya
jaya
kala
Ctrl+d

The output will be

jaya
kala
mani
priya
```

Pipes

A pipe is a mechanism in which the output of one command is sent as the input for another command.

Example 1

```
$ who am i | wc-l
1
```

In the above example, the output of `who am i` is passed as input to the `wc-l` command and the final result is displayed.

Pipes are temporary files that store the output of one command in memory and pass it as input to the other command.

Example 2

Consider another example,

```
$ cat demo |tail –4
```

The above command displays the last 4 lines of the file demo.

Example 3

Consider the following file:

```
$ cat movie
m001:the mask:sathyam
m002:gladiator:sree
m003:speed:santham
m004:omen:studio5
$_
```

The third column alone can be displayed using pipes as shown below:

```
$ cat movie | cut –d “:” –f3
sathyam
sree
santham
studio5
$_
```

The tee command

The intermediate output in a pipe is discarded by Linux. Sometimes it is necessary to pipe the standard output of a command to another command and also save it on disk for later use. It is also possible that we may want the output of a particular command in a long pipeline to be stored for later use. The tee command is used for solving such problems.

```
$ cat emp | sort | tee temp | cut –d “:” –f1
100
101
```

```
102
104
$_
$ cat temp
100:meera:india
101:prabha:china
102:rajesh:america
104:rani:singapore
$_
```

☞ TOTAL RECALL

1. What is the significance of redirection?
2. What is the function of a pipe?
3. What does the sort –f option do?
4. How does the cut filter function?

Advanced file operations

The *find* command

The find command is used to locate files in a directory and all its subdirectories. The syntax is

```
$ find [path] [expression]
```

There are various options available with the find command. Let us explore each option one by one.

The –name option

The –name option lists out the specific files in all directories beginning from the named directory. For example, consider the following command

```
$ find . –name "emp" –print
```

- the search is to begin from the current directory which is indicated by the **.** (dot)
- the –name "emp" option indicates that the files with name "emp" have to be searched
- -print option is used to display the full path name of the files on the standard output device

The –type option

The –type option is used to locate a file of a specific type. Consider the command

```
$ find /home/mydir –type f –print
```

- mydir is the directory from which the search has to begin
- -type f indicates that the search is for ordinary files

To search for directory files, the following command can be used,

```
$ find /home/mydir –type d –print
```

The –mtime option

The –mtime option allows to find files that have been modified before or after a specified time. The parameters that can be used with the –mtime option and their significance is explained in the table given below:

Option	Significance	Example
`-mtime n`	Finds a file modified n days before the current date	If the current date is 10-5-2001, then the option `–mtime 5` will find files modified on 5-5-2001
`-mtime +n`	Finds files modified in more than n days	If the current date is 10-5-2001, then the option `–mtime +5` will find files modified before 5-5-2001
`-mtime –n`	Finds files modified in less than n days	If the current date is 10-5-2001, then the option `–mtime -5` will find files modified after 5-5-2001

The –exec option

The –exec option is used to execute commands on files found by the find command.

```
$ find .  –name “temp” –type f –exec  rm{} \;
```

In the above example, we are trying to delete ordinary files named temp in the current directory. The {} must be specified in the exec command. The find command replaces the {} with the path names of the files located by it and the action `rm` is executed on each file. The backslash followed by semicolon is part of the syntax.

The –ok option

The –ok option is similar to the exec option. But for each file it asks the user for confirmation before executing the command.

```
$ find .  –name “temp” –type f –ok  rm{} \;
```

The *locate* command

The locate command is the fastest way to locate a file. It is faster than other commands because it looks for files in a single database containing file locations. The database is the `slocate.db` database under the `/var/libs/slocate` directory. The database is updated automatically by the

system every night at a scheduled time. The system administrator can also update the database using the `updatedb` command.

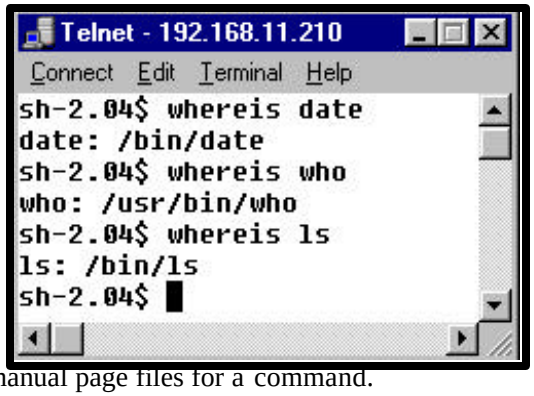
For example.

```
$ locate temp
```

The above command will display a list of the path names of all the files in which the string temp appears anywhere in the file name.

The *whereis* command

The whereis command is used to display full path name for executable command. It locates the binary, source, and manual page files for a command.



The *uniq* command

The uniq command manipulates lines, which occur more than once in a file. The file must be sorted, since uniq only compares adjacent lines. The various options available with the uniq command are:

Option	Meaning
-d	Displays only the lines that are duplicated in the input file
-u	Displays only lines that are unique
-c	Precedes each output line with the number of times that line occurred in the input.

Example

```
$ cat emp
100:meera:india
100:meera:india
101:prabha:china
102:rajesh:america
102:rajesh:america
104:rani:singapore
```

```
$_

$ uniq emp
100:meera:india
101:prabha:china
102:rajesh:america
104:rani:singapore
$_
```

The cmp command

The cmp command is used to compare two files. It displays the line number and character number at which the two files differ.

Consider the two files given below.

```
$ cat f1
This is the first file
$_

$ cat f2
This is the second file
$_
```

These two files are now compared using the cmp command as shown below:

```
$ cmp f1 f2
f1 f2 differ: char 13, line 1
```

The cmp command indicates the first character at which the files differ.

The comm command

The comm command is generally used to compare only data files. The comm command locates identical lines within files sorted in the same collating sequence and produces three columns; the first contains lines found only in the first file, the second contains lines found only in the second file and the third contains lines, which are in both files.

Consider the two files given below

```
Java
Ecommerce
Xml
```

f1

```
Java
WML
ASP
```

f2

Then the following command

```
$ comm f1 f2
```

produces the following output

```
Ecommerce          WML          Java
Xml                ASP
```

The following options may be used with the comm command

```
-1          displays only the 2nd and 3rd column
-2          displays only the 1st and 3rd column
-3          displays only the 1st and 2nd column
```

Consider the following command

```
$ comm –12 f1 f2
Java
```

The comm command can also compare the standard input with a disk file and display the results on the VDU.

```
$ comm – t1
```

The hyphen (-) indicates that the first file to be compared is the standard input. The above command implies that each line entered on the keyboard is compared with the lines from the file t1 and results are displayed on the monitor.

xargs

xargs is a command that accepts a list of words from standard input and provides those words as arguments to a given command:

```
$cat filelist | xargs rm
```

The output of cat cannot be piped directly to rm because rm does not look for filenames on standard input. xargs reads the files being passed by cat and builds up a command line beginning with rm and ending with the filenames. If there are a large number of files, xargs runs the rm command multiple times, deleting some of the files each time. The number of arguments can be specified from standard input to build up on the command line with the -n option:

```
$cat filelist | xargs -n 20 rm
```

-n 20 says to put only 20 arguments on each command line, so you delete only 20 files at a time. Here is a different example to give more insight into how xargs works:

```
$ ls
acme
report16
report3
report34
report527

$ ls | xargs -n 2 echo ===
=== acme report16
=== report3 report34
=== report527
$
```

The first ls command shows us that there are only five files in the current directory. (These five can be regular files or directories; it does not matter for this example.) Next pipe the output of ls to xargs, which composes and executes this command (the first of several):

```
echo === acme report16
```

The command begins with echo === because these are the arguments given to the xargs command. The command then contains two filenames read from standard input. -n 2 tells xargs to add only two words from standard input to each echo command. I added === as the first echo argument so you can visually find the output from each separate echo command. You can see that xargs called echo three times to process all the standard input.

xargs can be used to solve this problem: No continuity .

```
$ rm abc*
rm: arg list too long
```

The current directory contained too many filenames starting with *abc*, and the command buffer overflowed, so an error message was printed, and none of the files were deleted. xargs can solve this buffer overflow problem:

```
ls | grep '^abc' | xargs -n 20 rm
```

REVIEW QUESTIONS

1. The find command when followed by –exec option should have _____ at the end.
2. The _____ command is generally used to compare only data files.
3. A _____ is a mechanism in which the output of one command is sent as the input for another command.
4. The _____ is treated as the standard error file in Linux.
5. Grep stands for _____.
6. The _____ filter is used to translate one set of characters from the standard input to another.
7. The sort –b option helps in ignoring _____.