

CHAPTER 6

File Filters

Introduction

Linux/UNIX operating system supports a variety of file processing utilities called filters. This chapter explores the filters along with some other useful commands.

6.1 THE UNIQ COMMAND

This command displays uniq lines of the given files. That is if successive lines of a file are same then they will be removed. By default output will be on to the screen. This can be used to remove successive empty lines in a given file. The syntax of usage of this command is given as :

uniq [OPTIONS] INPUT [OUTPUT]

The available options are :

- c** Prefix lines by their occurrences
- d** Only print repeated lines
- D** Print All duplicate lines
- fN** Avoid first N fields while comparing
- i** Ignore case while comparing the lines
- sN** Skip N chars while comparing the lines
- u** Only print unique lines
- z** end lines with 0byte not new line
- wN** compare no more than N characters in the lines

Now see our workout at the command prompt on a sample file, testfile.

```
$ cat testfile
```

```
AAAA  
AAAA  
BBB  
CCCC  
CCCC
```

```
DDDD
EEDD
LLLL
$ uniq testfile (or uniq -u testfile)
AAAA
BBB
CCCC
CCCC
DDDD
EEDD
LLLL
```

This has printed the string AAAA only once. Also, as we have two lines with CCCC separated by an empty line, they are not eliminated.

```
$ uniq -d testfile
AAAA
```

It displayed only uniq lines.

```
$ uniq -s 2 testfile
AAAA
BBB
CCCC
CCCC
DDDD
LLLL
```

We have asked to skip two characters while comparing. Thus, this time DDDD is only selected while EEDD is not. This is because DDDD is considered same as EEDD if we skip first two characters. Thus, first of them, which is DDDD is printed.

```
$ uniq -s 2 -d testfile
AAAA
DDDD
$ uniq -D testfile
AAAA
AAAA
$ uniq -D -c testfile
uniq: printing all duplicated lines and repeat counts is meaningless
Try `uniq --help' for more information.
```

The useful -c option prefixes each line of the input file with its number of occurrences. Let the file "testfile" contains the following lines.
This line occurs only once.
This line occurs twice.
This line occurs twice.

This line occurs three times.
 This line occurs three times.
 This line occurs three times.
 Then, the following command is executed the result is as displayed below.

```
$uniq -c testfile
1 This line occurs only once.
2 This line occurs twice.
3 This line occurs three times.
$
```

Similarly, when the following command is executed the result is displayed as below. Probably, you can come back to these commands after reading piping.

```
$sort testfile | uniq -c | sort -nr
3 This line occurs three times.
2 This line occurs twice.
1 This line occurs only once.
$cat list-1 list-2 list-3 | sort | uniq final.list
$
```

The above command concatenates the list files, sorts them, removes duplicate lines, and finally writes the result in to an output file.

6.2 THE GREP COMMAND

This command is used to select lines from a file which are having some specified string (which is called as search pattern). The syntax of this command is :

grep [OPTIONS] pattern filename(s)

For example, the following command displays those lines of the file xyz having string rao.

```
grep "rao" xyz
```

The grep command accepts regular expressions as search patterns. For example, the following command displays those lines of the file xyz having strings either "Rao", or "rao". Here, [rR] is a regular expression.

```
grep "[rR]ao" xyz
```

```
grep "[rR]a[uo]" xyz
```

Above command displays those lines of the file xyz having strings either "Rao", or "Rau", or "rao", or "rau".

The following command displays those lines of the file xyz which starts with string "rao"

```
grep "^rao" xyz
```

The following command displays those lines of the file xyz which ends with string "rao".

```
grep "rao$" xyz
```

The following command displays those lines of the file xyz which contains the string "rao" only. No more characters in the line.

```
grep "^rao$" xyz
```

If we want to display only empty lines in the file xyz, we can run the following command.

```
grep "^$" xyz
```

```
grep "^[rR]ao" xyz
```

Above command displays those lines of the file xyz which starts with either "Rao" or "rao".

```
grep "[rR]ao$" xyz
```

Above command displays those lines of the file xyz which ends with "Rao" or "rao".

The following command displays those lines of the give file which contains strings of form Ra followed by any character. Here, . indicates any single character.

```
grep Ra. filename
```

```
grep "Ra." filename
```

The following command displays those lines of the given file which contains amount in dollars.

```
grep "\$[0-9]" filename
```

The following command displays those lines of the given file which contains words starting with li.

```
grep "\>li" filename
```

The following command displays those lines of the given file which contains words ending with li.

```
grep "\<li" filename
```

The following command displays those lines of the given file which contains words only "like"

```
grep "\blike\b" filename
```

The following command displays those lines of the given file which contains strings of nature 10, 100, 1000, etc.,

```
grep "10+" filename
```

The following command displays those lines of the given file which contains strings of nature 1000. Here, 10{3} indicates three 0's after 1.

```
grep "10{3}" filename
```

The following command displays those lines of the given file which contains strings of nature 1000, 10000, 100000. Here, 10{3,} indicates at least three 0's after 1.

```
grep "10{3,}" filename
```

The following command displays those lines of the given file which contains strings of nature 1000, 10000, and 100000. Here, 10{3,5} indicates three to five 0's after 1.

grep "10{3,5}" filename

The following command displays those lines of the given file which contains words which ends with 1000. Here, 10{3}\b indicates three 0's after 1 and at the end of the word. Similarly, we can use >, < etc., also.

grep "10{3}\b" filename

If we want those lines of the given file which contains "red color", "blue color" or "white color", then the grep can be executed as :

grep "(red|blue|white) color" filename

The following grep command displays those lines having patterns 15, 1515 or 151515.

Grep "(15){3}" filename

The following extracts the credit card numbers

grep -E '\b[0-9]{4}((|-|)[0-9]{4}){3}\b' filename

Sample output :

1234 5678 9012 3456

1234567890123456

1234-5678-9012-3456

-n option if we use with grep command it displays line numbers also.

grep -n "rao" xyz

This displays those lines of file xyz which are having the string "rao" along with their line numbers.

-v option if we use with grep command it displays those lines which does not have the given search pattern.

grep -v "rao" xyz

This displays those lines of the file xyz which does not contain the string "rao".

-f option followed by a filename file takes search patterns from *file*. This option allows you to input all the patterns you want to match into a file, called *pattern.txt* here. Then, *grep* searches for all the patterns from *pattern.txt* in the designated file *searchhere.txt*. The patterns are additive; that is, *grep* returns every line that matches any pattern. The pattern file must list one pattern per line. If *pattern.txt* is empty, nothing will match.

-i option is used to indicate the ignore the case while carrying out pattern matching.

-w option Matches only when the input text consists of full words.

grep -w 'xyz' filename

In this example, it is not enough for a line to contain the three letters "xyz" in a row; there must actually be spaces or punctuation around them. Letters, digits, and the underscore character are all considered part of a word; any other character is considered a word boundary, as are the start and end of the line. This is the equivalent of putting \b at the beginning and end of the regular expression.

-x option is like -w, but must match an entire line.

grep -x 'Hello, World!' filename

This example matches only lines that consist entirely of "Hello, world!". Lines that have additional content will not be matched. This can be useful for parsing logfiles for specific content that might include cases you are not interested in seeing.

In the following command, first the command `whoami` is executed as it is enclosed in the back quotes and its results, i.e., username of the current user is taken as search pattern. Thus, it displays those lines of the given file which contains the username of the current user.

`-c` option with `grep` gives only the number lines in which matching pattern are found. For example, the following command displays how many times the file `tv.html` is accessed in a web site.

grep -c "tv.html" access.log

`-l` option can be used to print filenames only if the matching pattern is found. `grep` stops when it finds first match.

`-L` option is also used to print filenames not having matching pattern. `grep` commands stops if it finds a file having the given pattern.

`-m` option followed by an integer number (N) indicates the `grep` to work on search on first N lines of the given file.

`-o` option will print the only matched portion, not the whole line.

`-q` option is used to set `grep` in silent mode. It will not display the matched lines or words. In order to know whether the file is having given pattern, we can see the commands exit status by displaying the environment variable `$?`. If its value is 0 then `grep` is succeeded.

grep `whoami` filename

The `fgrep` (fixed `grep`) and `egrep` (extended `grep`) commands

`fgrep` is used search for a group of strings. One string has to be separated from other by a new line.

```
$fgrep `rao`  
>ram  
>raju` filename
```

This command displays those lines having either `rao` or `ram` or `raju`.

`fgrep` will not accept regular expressions.

The `egrep` is little more different. It also takes a group of strings. while specifying strings piping (`|`) can be also used as a separator.

Example :

egrep `rao|ram|raju` filename

In addition it accepts regular expressions also.

In fact, `grep` with `-e` option works like `egrep` which we have discussed in the previous examples.

6.3 THE SED COMMAND

`Sed` is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as `ed`), `sed` works by making only one pass over the input(s), and is consequently more efficient. But it is `sed`'s ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

We have seen vi editor in previous chapters. Sed uses almost same commands while processing the files.

The following options are used with sed.

-n shows only those lines on which sed actually acted. Sed's default behavior is to display the line what it has read from the file and also to display the line after applying the command. However, if we specify the -n option it displays only the lines after applying the action.

-f Usually the commands are enclosed in between single quotes. However, if we want the sed command to be stored in a file (like shell or awk script) and sed to use it then we will use this option.

In the following pages, we use the file 'ABC' with the following content for experimentation purpose.

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

Make sure you take things with whole heart.

sed -n 'ABC

This command displays nothing.

sed 'ABC

This command gives the following output which we can find as the file content. Actually, sed's default behavior is displaying the lines what it has read. Thus, we see the file content as it is. However, when we use -n option (above command), nothing is displayed as -n option displays lines content after applying our command; which is nothing in this case.

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

Make sure you take things with whole heart.

sed -n '1,2p' ABC

This command displays first two lines of the file as shown below.

I am not happy with your Progress.

I think you have to improve a lot.

sed '1,2p' ABC

This command displays first two lines two times and next lines once. This is because, sed's default behavior is to display the lines read. In addition, we have asked to print first two lines. Thus, first two lines are printed twice.

I am not happy with your Progress.

I am not happy with your Progress.

I think you have to improve a lot.

I think you have to improve a lot.

Why you are not serious?.

Make sure you take things with whole heart.

sed '1,2d' ABC

This command displays 3rd and fourth lines of the files as shown below.

Why you are not serious?.

Make sure you take things with whole heart.

sed -n '1,2d' ABC

This command displays nothing. Why?.

Now, run the following commands and see the content of the file 'pp'.

sed '1,2d' ABC>pp

cat pp

Why you are not serious?.

Make sure you take things with whole heart.

sed '1,/Why/p' ABC

This command displays lines from 1st line to the (first) line having the pattern 'Why'.

I am not happy with your Progress.

I am not happy with your Progress.

I think you have to improve a lot.

I think you have to improve a lot.

Why you are not serious?.

Why you are not serious?

Make sure you take things with whole heart.

sed -n '1,/Why/p' ABC

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

sed '1,/Why/w pp1' ABC

The above command writes the selected lines into the file pp1.

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

Make sure you take things with whole heart.

\$cat pp1

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

Run the following commands and check the difference.

sed -n '1,/Why/w pp2' ABC

cat pp2

I am not happy with your Progress.

I think you have to improve a lot.

Why you are not serious?.

sed '/not/s/not/NOT/g' ABC

The above command replaces all the occurrences of the string 'not' with 'NOT'. Remember, original file is not changed.

```
I am NOT happy with your Progress.  
I think you have to improve a lot.  
Why you are NOT serious?.  
Make sure you take things with whole heart.
```

sed '1 r pp1' ABC

The above command reads the content from the file 'pp1'. That is, it displays 1'st line of the file ABC and then the content of the file pp1 and then remaining content of file ABC.

Thus, the result is as follows.

```
I am not happy with your Progress.  
I am not happy with your Progress.  
I think you have to improve a lot.  
Why you are not serious?.  
I think you have to improve a lot.  
Why you are not serious?.  
Make sure you take things with whole heart.
```

We can ask sed to take instructions from a file. To explain, we now create a file say, 'ff' with the sed command in it.

cat >ff

```
/not/s/not/NOT/g  
^d
```

sed -f ff ABC

Now with the help of -f option, we are informing sed to take commands from the file 'ff'. The result is as follows.

```
I am NOT happy with your Progress.  
I think you have to improve a lot.  
Why you are NOT serious?.  
Make sure you take things with whole heart.
```

sed -n '1,3 w pp2' ABC

This command prints first three lines to file pp2, which we can check by running the following command.

cat pp2**sed -n '1~2 w pp2' ABC**

This command prints every alternative line from first line. We can check the same by running the following command.

cat pp2**sed '1~2d' ABC**

This command prints 2nd and fourth lines while deleting 1st, 3rd, etc., line.

sed '1,/Why/d' ABC

This command displays all the lines other than the lines from 1st line to the line which contains the pattern 'Why'.

sed '1,/Why/s/not/NOT/' ABC

This command replaces the pattern 'not' with 'NOT' in the lines starting from 1st line to the line having the pattern 'Why'.

sed '1,2s/not/NOT/' ABC

This command replaces the pattern 'not' with 'NOT' in the lines 1 to 2.

6.4 THE CUT COMMAND

This is used to split files vertically. Remember, tail and head can be used to split files horizontally. The cat command can be used to join the file horizontally.

cut -f1,3 filename

The above command displays 1st and 3rd words of each line of the given file. Input file is assumed to be having TAB as the field (word) separator.

cut -d":" -f1,3 /etc/passwd

The file /etc/passwd contains the details of all the legal users of the machine. One line of this file contains details of one user on the machine. The details include: user name, encrypted password, UID, group, real name, home directory of the user and default shell. All are separated by colons. This file can be seen by any one. However, we can not directly edit the same through editors such as vi.

The above command displays username, UID of each legal user of the machine. Here, with **-d** option we are specifying that : is the field separator between word to word in the file.

Cut command can not change the natural order of the fields. That is, the following command also gives same result as that of the above command.

cut -d":" -f3,1 /etc/passwd

cut -d":" -f1,3 filename

Both the commands display first and third word from each line of the given file.

cut -f":" -f3- filename

This displays third word to till last word of each line of the given file.

cut -c3-5 filename

This displays 3rd character to 5th character of each line of the given file.

cut -d":" -f1 /etc/passwd > a1

File a1 contains usernames of all the legal users of the machine.

cut -d":" -f3 /etc/passwd > a3

File a3 contains UID's of each legal user of the machine.

The following command extracts 3rd to 5th characters from each line of the given file xyz.

cut -c3-5 xyz

The following command extracts 3rd and 5th characters from each line of the given file xyz.

cut -c3,5 xyz

The following command extracts 3rd character to last character from each line of the given file xyz.

cut -c3- xyz

The following command extracts first two characters from each line of the file.

cut -c-2 xyz

Here, -2 indicate first character to till character 2 in each line.

6.5 THE PASTE COMMAND

This is used to join files vertically. Assuming that the file a3 contains UID's and file a1 contains user names, the following command joins them vertically and stores the results in the file a31.

```
paste a3 a1 >a31
```

```
cat a31
```

This display

```
0  root
1  bin
2  daemon
3  adm
```

The file a3 contains :

```
0
1
2
3
```

The file a1 contains:

```
root
bin
daemon
adm
```

```
paste -d"|" a3 a1 >a13
```

The above command places the given field separator while joining the files contents vertically. The content of a13 will be as shown below.

```
cat a13
```

```
0|root
1|bin
2|daemon
3|adm
```

The following workout shows the effect of -s option with paste command.

```
$ cat a3
```

```
Abhi
```

```
Anna
```

```
Bob
```

```
Hi
```

```
Ramya
```

```
$ cat a4
```

```
cat <<-ENDOFMESSAGE
```

```
This is line 1 of the message.
```

```
This is line 2 of the message.
```

```
This is line 3 of the message.
```

```
This is line 4 of the message.
```

```
This is the last line of the message.
```

```
ENDOFMESSAGE
```

```
# The output of the script
```

```
$ paste -s a3 a4
```

```
Abhi Anna Bob Hi Ramya
```

```
cat <<-ENDOFMESSAGE This is line 1 of the message. This is line 2 of the message. This is line 3 of the message. This is line 4 of the message. This is the last line of the message. ENDOFMESSAGE # The output of the script
```

The paste command works like cat command if we don't specify any argument. See the following work out.

```
$ paste >hh
```

```
Hello How
```

```
are you
```

```
dear
```

```
^d
```

```
$ cat hh
```

```
Hello How
```

```
are you
```

```
dear
```

6.6 JOIN COMMAND

This is used to join files. Unlike paste it works similar to join operation of DBMS.

Let the files content are :

File aa1 contains

```
111|NBV Rao
```

```
121|PP Raj
```

```
116|Teja
```

```
119|Rani
```

File aa2 contains

```
111|Prof
112|Asst Prof
121|lecturer
116|Prof
```

join -t'|' -j 1 1 aa1 aa2

This command produces the following result :

```
111|NBV Rao|Prof
121|PP Raj|lecturer
116|Teja|Prof
```

join -t'|' -j 1 1 -o 1.1 2.2 aa1 aa2

This command produces output such as the following. That is, first field from the first file and second field from the second file is displayed.

```
111|Prof
121|lecturer
116|Prof
```

join -t'|' -a1 -o 1.1 2.2 aa1 aa2

This command gives the following results.

```
111|Prof
121|lecturer
116|Prof
119|
```

join -t'|' -a2 -o 1.1 2.2 aa1 aa2

This command gives the following results.

```
111|Prof
|Asst Prof
121|lecturer
116|Prof
```

6.7 THE SPLIT COMMAND

The split command can be used to split a text file into smaller files. The syntax of this command is given as :

split [*OPTION*] [*INPUT* [*PREFIX*]]

Output fixed-size pieces of INPUT to PREFIXaa, PREFIXab, ...; default PREFIX is 'x'.
With no INPUT, or when INPUT is -, read standard input.

The available options are :

-a, --suffix-length=N

use suffixes of length N (default 2)

-b, --bytes=SIZE

put SIZE bytes per output file

-C, --line-bytes=SIZE

put at most SIZE bytes of lines per output file

-l, --lines=NUMBER

put NUMBER lines per output file

--verbose

print a diagnostic to standard error just before each output file is opened

--help

display this help and exit

--version

output version information and exit

SIZE may have a multiplier suffix: b for 512, k for 1K, m for 1 Meg.

For example, we have a file aa and I wanted to split that file into a set of files such that each file contains one line of this file. It can be achieved by executing the following command.

split -l 1 aa test

If we observe in the current directory we may find files with the names testaaa, testaab, etc.,

If you open these files, they exactly contains single line of the file aa.

If we want to join all these files, we can happily use cat command such as :

cat test*>ppp

The content of the file ppp will be exactly same as aa. You can verify by running the diff command such as :

diff aa ppp

6.8 THE SORT COMMAND

The sort command can be used to sort contents of file or files. The syntax is as follows :

sort options filename(s)

Sort sorts, merges, or compares all the lines from the given files, or the standard input if no files are given. sort has three modes of operation: sort (the default), merge, and check if the file is sorted.

OPTIONS

-c

Check whether the given files are already sorted: if they are not all sorted, print an error message and exit with a status of 1.

-m

Merge the given files by sorting them as a group. Each input file should already be individually sorted. It always works to sort instead of merge; merging is provided because it is faster, in the case where it works.

-b

Ignore leading blanks when finding sort keys in each line.

-d

Sort in 'phone directory' order: ignore all characters except letters, digits and blanks when sorting.

-f

Fold lower case characters into the equivalent upper case characters when sorting so that, for EXAMPLE, 'b' is sorted the same way 'B' is.

-i

Ignore characters outside the ASCII range 040-0176 octal (inclusive) when sorting.

-M

An initial string, consisting of any amount of white space, followed by three letters abbreviating a month name, is folded to UPPER case and compared in the order 'JAN' < 'FEB' < ... < 'DEC.' Invalid names compare low to valid names.

-n

Compare according to arithmetic value an initial numeric string consisting of optional white space, an optional - sign, and zero or more digits, optionally followed by a decimal point and zero or more digits.

-r

Reverse the result of comparison, so that lines with greater key values appear earlier in the output instead of later.

Other options are :

-o output-file

Write output to output-file instead of to the standard output. If output-file is one of the input files, sort copies it to a temporary file before sorting and writing the output to output-file.

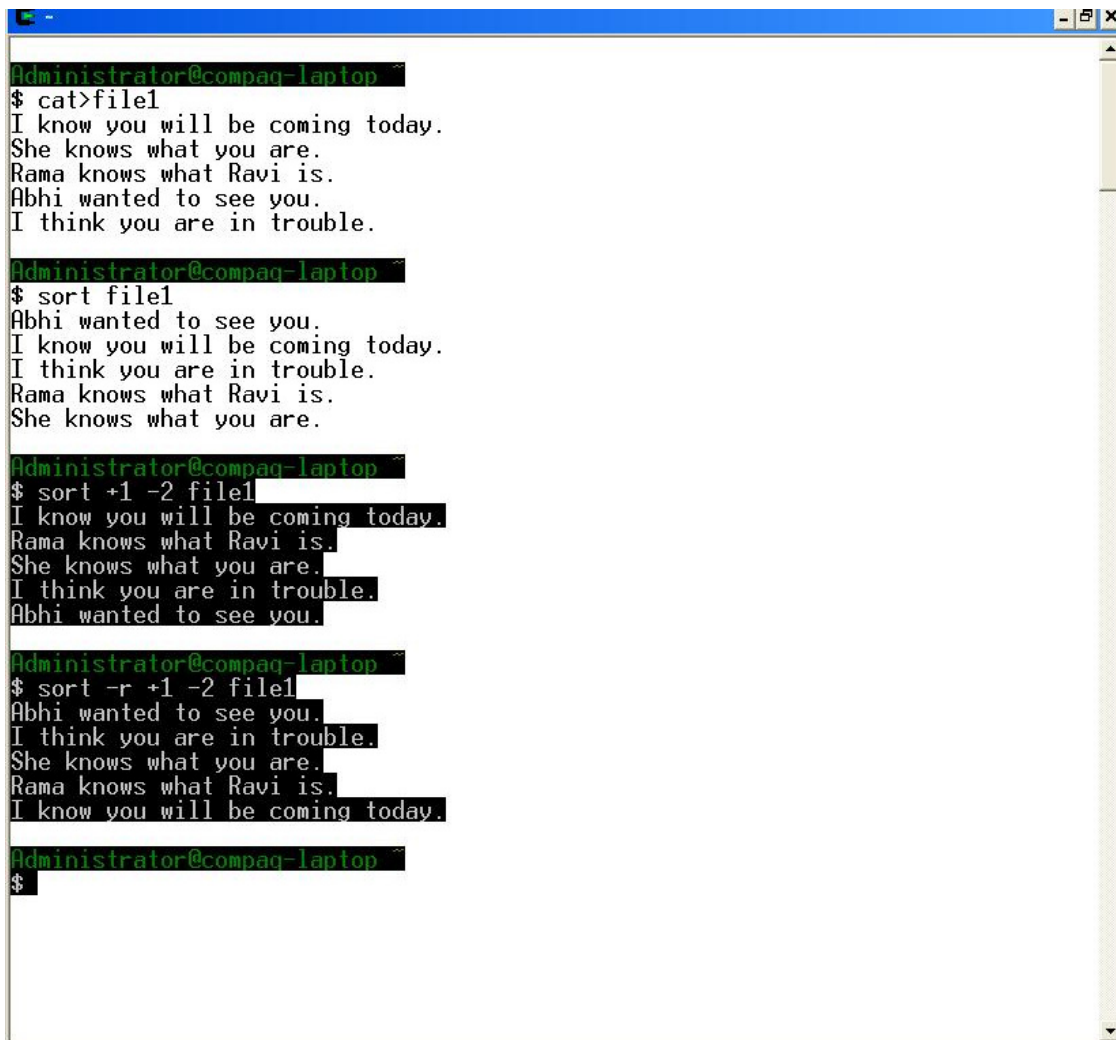
-t separator

Use character separator as the field separator when finding the sort keys in each line. By default, fields are separated by the empty string between a non-whitespace character and a whitespace character. That is to say, given the input line 'foobar', sort breaks it into fields 'foo' and 'bar'. The field separator is not considered to be part of either the field preceding or the field following it.

-u

For the default case or the -m option, only output the first of a sequence of lines that compare equal. For the -c option, check that no pair of consecutive lines compares equal.

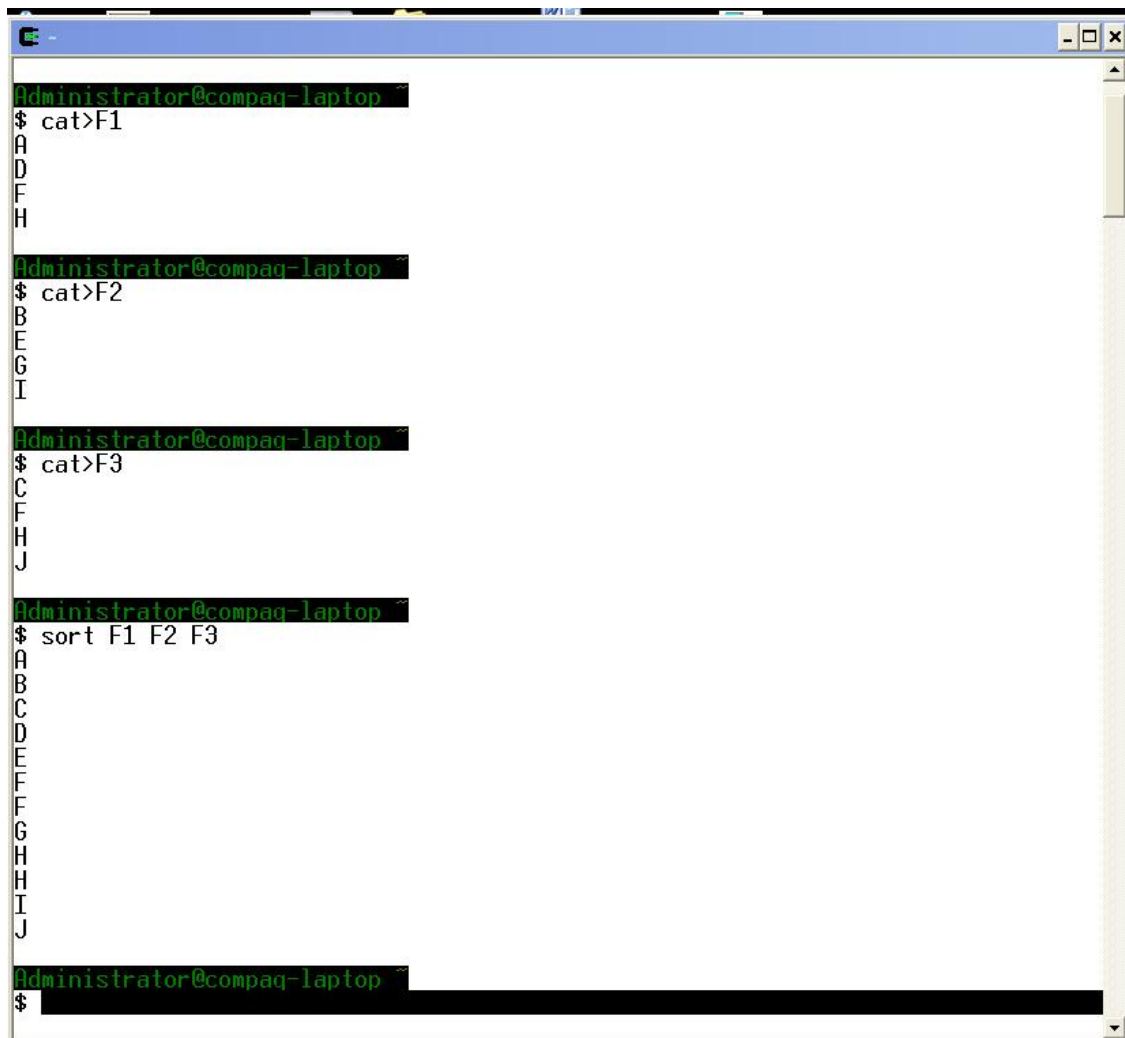
Now see Figure 1 for an interactive session of sort on a file "file1".



```
Administrator@compaq-laptop ~  
$ cat>file1  
I know you will be coming today.  
She knows what you are.  
Rama knows what Ravi is.  
Abhi wanted to see you.  
I think you are in trouble.  
  
Administrator@compaq-laptop ~  
$ sort file1  
Abhi wanted to see you.  
I know you will be coming today.  
I think you are in trouble.  
Rama knows what Ravi is.  
She knows what you are.  
  
Administrator@compaq-laptop ~  
$ sort +1 -2 file1  
I know you will be coming today.  
Rama knows what Ravi is.  
She knows what you are.  
I think you are in trouble.  
Abhi wanted to see you.  
  
Administrator@compaq-laptop ~  
$ sort -r +1 -2 file1  
Abhi wanted to see you.  
I think you are in trouble.  
She knows what you are.  
Rama knows what Ravi is.  
I know you will be coming today.  
  
Administrator@compaq-laptop ~  
$
```

Figure 6.1 A Sample Working with sort

Now see the following session output (Figure 6.2). We have created three files F1, F2 and F3 and then executed the command `sort F1 F2 F3`. We may find sorted output of all the three files content. That is sort can be used to merge files while sorting.



```
Administrator@compaq-laptop ~  
$ cat>F1  
A  
D  
F  
H  
  
Administrator@compaq-laptop ~  
$ cat>F2  
B  
E  
G  
I  
  
Administrator@compaq-laptop ~  
$ cat>F3  
C  
F  
H  
J  
  
Administrator@compaq-laptop ~  
$ sort F1 F2 F3  
A  
B  
C  
D  
E  
F  
F  
G  
H  
H  
I  
J  
  
Administrator@compaq-laptop ~  
$
```

Figure 6.2 Another working example of sort command

Assume that a file abc contains numerical values like the following

```
10  
92  
1  
111  
300
```

Then, **sort abc** command gives

```
1  
10  
111  
300  
92
```

This happens sort will be sorting the strings based on ASCII values. Now, if we want to sort in numerical order, we have to use `-n` option. That is the following command gives us numerically ordered output of file `abc`.

sort -n abc

Output :

```
1
10
92
111
300
```

Similarly, if we run **sort -r -n abc** command, we will get numbers in reverse order.

Some times, we may encounter to sort data base files with special character as field separators. To inform about the field separator, we use `-t` option. To experiment, let us create a file with `|` (piping character) as field separator.

cat employee

```
111|NBV Rao|Prof.|CSE|34900|43|2
123|PN Rao|Prof.|ECE|40220|40|3
121|CS Reddy|Prof|CIV|34650|45|4
122|Mrs. Radha|Lect|CSE|19870|28|1
119|PP Raja|Asst Prof.|ECE|23460|29|1
```

The following command sorts the content of the file based on second word of each line.

sort -t'|' +2 -3 employee

Output :

```
119|PP Raja|Asst Prof.|ECE|23460|29|1
122|Mrs. Radha|Lect|CSE|19870|28|1
121|CS Reddy|Prof|CIV|34650|45|4
111|NBV Rao|Prof.|CSE|34900|43|2
123|PN Rao|Prof.|ECE|40220|40|3
```

See the difference the following command and the above command.

sort employee

Output:

```
111|NBV Rao|Prof.|CSE|34900|43|2
119|PP Raja|Asst Prof.|ECE|23460|29|1
121|CS Reddy|Prof|CIV|34650|45|4
122|Mrs. Radha|Lect|CSE|19870|28|1
123|PN Rao|Prof.|ECE|40220|40|3
```

The following command sorts the content of the file `employee` based on their salaries (6th field).

sort -t'|' -n +5 -6 employee**Output:**

```
122|Mrs. Radha|Lect|CSE|19870|28|1
119|PP Raja|Asst Prof.|ECE|23460|29|1
123|PN Rao|Prof.|ECE|40220|40|3
111|NBV Rao|Prof.|CSE|34900|43|2
121|CS Reddy|Prof|CIV|34650|45|4
```

We can use `-o` option to save sorted output. For example, the following command saves the sorted output in a file `aaaa`.

sort -t'|' -o aaaa +2 -3 employee

We can check whether file content is already in sorted order or not.

sort -c aaaa

```
sort: aaaa:3: disorder: 121|CS Reddy|Prof|CIV|34650|45|4
```

The following command displays nothing indicating that the file `aaaa` contains the sorted data based on second word of each line.

sort -t'|' -c +2 -3 aaaa

In the following, `sort` takes interactive input.

\$ sort -o a3

```
Hi
Ramya
Abhi
Anna
Bob
```

\$ cat a3

```
Abhi
Anna
Bob
Hi
Ramya
```

The `sort` command can accept standard input through redirection operators. Here, we specify the input through the here the document.

\$ sort -o aa3 <<END

```
> Hello How are You
> I guess you are fine.
> Ramu is coming today.
> Abhi needs you.
> Bob may kill you.
> END
```

\$ cat aa3

Abhi needs you.
 Bob may kill you.
 Hello How are You
 I guess you are fine.
 Ramu is coming today.

6.9 THE TR COMMAND

This command can be used for transliteration. That is replacing a character with another character. It accepts standard input and gives standard output.

```
tr '*' '-' <xyz
```

This command replaces all the occurrences of character `*` with `-` in the given file `xyz`.

```
tr '*' '/' '-' '?' <xyz
```

This command replaces all the occurrences of `*` with `-` and `/` with `?` in the given file. In both the situations output appears on the screen. By standard redirection operator output can be stored in a file.

Example

```
tr '*' '-' <xyz >pqr
```

```
tr '[a-z]' '[A-Z]' < xyz
```

This command replaces all lower case characters of the file `xyz` to uppercase.

```
tr -d '*' <xyz
```

This command removes all occurrences of `*` in the given file `xyz`.

```
tr -s '*' <xyz
```

This command replaces multiple consecutive `*`'s with a single `*` in the given file.

6.10 THE DIFF COMMAND

This is used to compare the contents of two files in general. The syntax of the `diff` command is given as :

diff OPTIONS from-file to-file

In the simplest case, *diff* compares the contents of the two files *from-file* and *to-file*. A file name of `-` stands for text read from the standard input. As a special case, **diff - -** compares a copy of standard input to itself.

If *from-file* is a directory and *to-file* is not, *diff* compares the file in *from-file* whose file name is that of *to-file*, and vice versa. The non-directory file must not be `-`.

If both *from-file* and *to-file* are directories, *diff* compares corresponding files in both directories, in alphabetical order; this comparison is not recursive unless the **-r** or **--recursive** option is given. *diff* never compares the actual contents of a directory as if it were a file. The file that is fully specified may not be standard input, because standard input is nameless and the notion of "file with the same name" does not apply.

diff options begin with `-`, so normally *from-file* and *to-file* may not begin with `-`. However, `—` as an argument by itself treats the remaining arguments as file names even if they begin with `-`.

Example**diff p1 p2**

If the content of p1 and p2 are exactly same it displays nothing. Otherwise it displays the difference information in a special format such as the following :

```
1c1
< ass
—
> add
3d2
< ass
```

The diff command uses some special symbols and instructions to indicate the changes that are required to be made two files such that one may become another.

For example, in the above output 1c1 indicates that line 1 of first file to be changed. The following line contains <ass. This has to be changed with add (see 4th line in the output). Similarly, 3rd line in the first file to be removed. Like this, if we some operations file1 content becomes file2 content and vice versa.

Consider another example.

Suppose we have two files with the following content.

1. txt :

```
aaa
bbb
ccc
ddd
eee
fff
ggg
```

2. txt :

```
bbb
c c
ddd
eee
fff
ggg
hhh
```

```
$ diff 1.txt 2.txt
```

```
1d0
< aaa
3c2
< ccc
—
> c c
7a7
> hhh
```

Lines like "**1d0**" and "**3c2**" are the coordinates and types of the differences between the two compared files, while lines like "**< aaa**" and "**> hhh**" are the differences themselves.

Diff change notation includes 2 numbers and a character between them. Characters tell us what kind of change was discovered :

- d** - a line was deleted
- c** - a line was changed
- a** - a line was added

Number to the left of the character gives you the line number in the original (first) file, and the number to the right of the character tells you the line number in the second file used in comparison.

So, looking at the two text files and the diff output above, we can see what happened:

This means that 1 line was deleted. **< aaa** suggests that the aaa line is present only in the original file :

```
1d0
< aaa
```

And this means that the line number 3 has changed. We can see how this confirms that in the first file the line was "ccc", and in the second it now is "c c".

```
3c2
< ccc
—
> c c
```

Finally, this confirms that one new line appeared in the second file, it's "hhh" in the line number 7 :

```
7a7
> hhh
```

Some of the available options with diff are :

-a Treat all files as text and compare them line-by-line, even if they do not seem to be text.

-b Ignore changes in amount of white space.

-i Ignore case

-e Give output suitable for ed script

-q Report whether there is a difference or not. No other difference output.

-r When comparing directories, compare the directories recursively.

-w Ignore white space while comparing.

-x pattern When comparing directories ignore filenames whose basename follows pattern.

-X file While comparing directories ignore files or directories matching with file.

Sometime we need to compare the contents of the directories. We can do so with the help of diff command. For example, the following command compares dirA and dirB.

diff -qr dirA dirB

Why to compare directories?

We need compare the directories because of the following reasons:

- **identifying if some files are missing from one of the directories** - can be useful when we want to make sure two directories with configuration files for a

certain package are identical - files can be different, but the same files are present in the same locations for both directories

- **confirming if files in two directories are the same** - a typical task when comparing your actual data against a backup copy. When something goes wrong, this is one of the first things we do to make sure all the important files are not only present, but are actually the same as they have been when we took the last backup copy
- **highlighting textual differences between files in directories** - this is a useful exercise when we are looking at two similar directories and expect only minor changes between the files - version numbers, different file or directory names hard coded in various scripts, etc.

Assume that we have two directories under /tmp.

```
ls /tmp/dir1
/tmp/dir1
/tmp/dir1/file1
/tmp/dir1/file2
/tmp/dir1/dir11
/tmp/dir1/dir11/file11
/tmp/dir1/dir11/file12
ls /tmp/dir2
/tmp/dir2
/tmp/dir2/file1
/tmp/dir2/dir11
/tmp/dir2/dir11/file11
/tmp/dir2/dir11/file12
/tmp/dir2/file3
```

We have dir11 subdirectory in both. There's also a few files here and there, some of them missing from one of the directories specifically to be highlighted by our comparison exercises.

We ran diff command as given below and its output is also shown.

```
$diff /tmp/dir1 /tmp/dir2
Common subdirectories: /tmp/dir1/dir11 and /tmp/dir2/dir11
diff /tmp/dir1/file1 /tmp/dir2/file1
1d0
< New line
Only in /tmp/dir1: file2
Only in /tmp/dir2: file3
```

This output confirms that **/tmp/dir1** and **/tmp/dir2** both contain a dir11 directory, and also shows that **/tmp/dir1/file1** and **/tmp/dir2/file1** are actually different files even though they have the same name. By default, diff compares such files and we can see the result of each comparison in the output. Also included are pointers to the files which are present only in one of the compared directories: we can see that **file2** can only be found in **/tmp/dir1** and **file3** was present only in **/tmp/dir2**.

From the example below, it is easy to deduct that the command line for identifying files missing in one of the directories will be this one :

```
diff /tmp/dir1 /tmp/dir2 | grep Only
```

```
Only in /tmp/dir1: file2
```

```
Only in /tmp/dir2: file3
```

If we were only interested in files which exist in both directory structures, but are different - we can use a special command line option. It will simply point the files out, without getting into any further details. We will probably notice how this output is very similar to the default one :

```
diff -brief /tmp/dir1 /tmp/dir2
```

```
Common subdirectories: /tmp/dir1/dir11 and /tmp/dir2/dir11
```

```
Files /tmp/dir1/file1 and /tmp/dir2/file1 differ
```

```
Only in /tmp/dir1: file2
```

```
Only in /tmp/dir2: file3
```

Note how instead of showing the difference between *file1* in */tmp/dir1* and */tmp/dir2*, this time you only get told that these two files are different.

Recursively comparing directories

We can compare directories recursively. Incidentally, the example directories which we have taken are having simple structure. Thus, we don't find much difference in the following output.

```
$diff -recursive -brief /tmp/dir1 /tmp/dir2
```

```
Files /tmp/dir1/dir11/file12 and /tmp/dir2/dir11/file12 differ
```

```
Files /tmp/dir1/file1 and /tmp/dir2/file1 differ
```

```
Only in /tmp/dir1: file2
```

```
Only in /tmp/dir2: file3
```

6.11 THE CMP COMMAND

The `cmp` utility compares two files of any type and writes the results to the standard output. By default, `cmp` is silent if the files are the same; if they differ, the byte and line number at which the first difference occurred is reported.

Bytes and lines are numbered beginning with one.

Example : `cmp file1 file2`

6.12 THE COMM COMMAND

`comm` - compare two sorted files line by line

Compare sorted files `LEFT_FILE` and `RIGHT_FILE` line by line.

1. suppress lines unique to left file
2. suppress lines unique to right file
3. suppress lines that appear in both files

Example : `comm p1 p2`

6.13 SOFTWARE PATCHING

Normally SW products are supplied either as binary distribution or source distribution. In source distribution, all the source program files are supplied to the customer and the customer is required to compile on his target machine to get binary or executable code of the SW. Moreover, it is common that SW systems are released in incremental fashion. When a new release is made, a patch file (difference file) is prepared by comparing with the previous release files. This can be downloaded by the customer who is having previous release and by applying the SW patching he can get recent version of the SW source which he can compile to get updated version of SW running on his machine.

Example :

diff p1 p2>p3

Here p3 can be called as patch file.

patch p1 <p3 will change the content of the file p1 as p2.

patch p2 <p3 will change the p2 file content as p1.

6.14 CONCLUSIONS

We have introduced to commands such as uniq, grep, sort, cut, paste, split, diff, etc., in this chapter with many live examples. Also, we have explained about SW patching which is needed in the daily life of a programmer.