Chapter 11:
Text Processing Utilities

Learning objectives

Upon completing this chapter, the learner should be able to:

- Work with text files
- Search text files with grep and displaying text files or part of them with different utilities
- Understand how regular expressions can be used to make the search results more specific
- Learn about the very sophisticated utilities awk and sed
- Perform more advanced operations on text files

Key terms

Text processing

less

cat

head

tail

cut

sort

wc

Text fields

Regular expressions

regex engine

Basic Regular Expression (BRE)

Extended Regular Expression (ERE)

Line anchor

Stream Editor (SED)

AWK

Table of content

1. Using Common text file–related tools 4
   1.1. Doing more with less 5
   1.2. Showing file contents with cat 6
   1.3. Using head and tail 7
   1.4. Filtering specific columns with cut 8
   1.5. Sorting file contents and output with sort 9
   1.6. Counting lines, words, and characters with wc 10
2. A primer to using regular expressions 11
   2.1. Types of regex 12
   2.2. Using line anchors 13
   2.3. Using escaping in regular expressions 13
   2.4. Using wildcards and multipliers 14
   2.5. Using grep to analyze text 16
3. Other useful text processing utilities 18
   3.1. Learning Awk basics 18
      3.1.1. Awk syntax and options 18
      3.1.2. Common use and examples 19
   3.2. Learning Sed basics 20
      3.2.1. Sed syntax and options 20
      3.2.2. Common used examples 21

1. Using Common text file-related tools

Before we start talking about the best possible way to find text files containing specific text, let's take a look at how you can display text files in an efficient way. Table provides an overview of some common commands often used for managing text file contents.

Command	Explanation
less	Opens the text file in a pager, which allows for easy reading of the text file.
cat	Dumps the contents of the text file on the screen.
head	Shows the first 10 lines of the text file.
tail	Shows the last 10 lines of the text file.
cut	Used to filter specific columns or characters from a text file.
sort	Sorts contents of a text file.
wc	Counts the number of lines, words, and characters in a file.

Apart from using these commands on a text file, they may also prove very useful when used in pipes, where you can use the command `ps aux | less`, which sends the output of the command `ps aux` to the pager `less` to allow for easy reading.

1.1. Doing more with less

In many cases, as a Linux administrator you'll need to read the contents of text files, the less utility offers a convenient way to do so. You can use the command less /etc/passwd, for example, to open the contents of the /etc/passwd file in the less pager.

From less, you can use the PageUp and PageDown keys on your keyboard to browse through the file contents. Then you can press q to quit less. Also, very useful is that you can easily search for specific contents in less using /sometext for a forward search and? sometext for a backward search. Repeat the last search by using n. You have seen similar behavior in vim and man. That is because all of these commands are based on the same code.

Exercise: Applying basic less skills

In this exercise, you apply some basic less skills working with file contents and command output.

1. From a terminal, type less /etc/passwd. This opens the /etc/passwd file in the less pager.
2. Type G to go to the last line in the file.
3. Type /root to look for the text root. You'll see that all occurrences of the text root are highlighted.
4. Type q to quit less.
5. Type ps aux | less. This sends the output of the ps aux command (which shows a listing of all processes) to less. Browse through the list.
6. Press q to quit less.

1.2. Showing file contents with cat

The less utility is useful to read long text files. If a text file is not that long, you are probably better off using cat. This tool just dumps the contents of the text file on the terminal it was started from, and if the text file is long, however, you'll see all contents scrolling on the screen, and only the lines that fit on the terminal screen are displayed.

Using cat is simple, just type cat followed by the name of the file you want to see. For instance, use cat /etc/passwd to show the contents of this file on your computer screen.

```
[root@server1:~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
...
sshd:x:74:74:Privilege-separated
u1:x:1000:1000:User 1:/home/u1:/bin/bash
u2:x:1001:1002::/home/u2:/bin/bash
```

If you are interested in the first lines, you can use the tac utility, which gives the inversed result of cat.

```
[root@server1:~]# tac /etc/passwd
u2:x:1001:1002::/home/u2:/bin/bash
u1:x:1000:1000:User 1:/home/u1:/bin/bash
sshd:x:74:74:Privilege-separated
SSH:/var/empty/sshd:/sbin/nologin
…
bin:x:1:1:bin:/bin:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
```

1.3. Using head and tail

If a text file contains much information, it can be useful to filter the output a bit. You can use the head and tail utilities to do that.

Using head on a text file will show by default the first 10 lines of that file. Using tail on a text file shows the last 10 lines by default. You can adjust the number of lines that are shown by adding -n followed by the number you want to see. So, tail -n 5 /etc/passwd shows the last five lines of the /etc/passwd file.

On older versions of head and tail, you had to use the –n option to specify the number of lines you wanted to see. On current versions of both utilities, you may also omit the -n option. So, tail -5 /etc/passwd or tail -n 5 /etc/passwd gives you the exact same results.

Another useful option that you can use with tail is -f. This option starts by showing you the last 10 lines of the file you've specified, but it refreshes the display as new lines are added to the file. This is convenient for monitoring log files. The command tail -f /var/log/messages is a common command to show in real – time messages that are written to the main log file /var/log/messages. When combining tail and head, you can do smart things as well, specially to extract parts of specific text file.

Exercise: Using basic head and tail operations

In this exercise, you learn how to use head and tail to get exactly what you want.

1. Type tail –f /var/log/messages. You'll see the last lines of /var/log/messages being displayed. The file doesn't close automatically.
2. Type Ctrl+C to quit the previous command.
3. Type head –n 5 /etc/passwd to show the first five lines in /etc/passwd.
4. Type tail –n 2 /etc/passwd to show the last two lines of /etc/passwd.
5. Type head –n 5 /etc/passwd | tail –n 1 to show only line number 5 of the /etc/passwd file.

1.4. Filtering specific columns with cut

When working with text files, it can be useful to filter out specific fields. Imagine that you need to see a list of all users in the /etc/passwd file. In this file, several fields are defined, of which the first contains the name of the users who are defined. To filter out a specific field, the cut command is useful. To do this, use the -d option to specify the field delimiter followed by -f with the number of the specific field you want to filter out. So, the complete command is cut -d:-f 1 /etc/passwd if you want to filter out the first field of the /etc/passwd file. You can see the result in Listing.

```
[root@server1 ~]# cut -f 1 -d : /etc/passwd
root
bin
...
u1
u2
```

1.5. Sorting file contents and output with sort

Another very useful command to use on text file is sort, this command sorts text. If you type sort /etc/passwd, for instance, the content of the /etc/passwd file is sorted in alphabetical order. You can use the sort command on the output of a command also, as in cut -f 1 -d:/etc/passwd | sort, which sorts the contents of the first column in the /etc/passwd file.

By default, the sort command sorts in alphabetic order. In some cases, that is not convenient because the content that needs sorting may be numeric or in another format, for instance, cut -f 2 -d:/etc/passwd | sort -n sorts the second field of the /etc/passwd file in numeric order. It can be useful also to sort in reverse order; if you use the command du -h | sort -rn, you get a list of files sorted with the biggest file in that directory listed first.

You can also use the sort command and specify which column you want to sort. To do this, use sort -k3 -t:/etc/passwd, for instance, which uses the field separator: to sort the third column of the /etc/passwd file.

You might also like the option to sort on a specific column of a file or the output of a command. An example is the command `ps aux`, which gives an overview of the busiest processes on a Linux server. The listing below shows the command ps aux gives an overview of the busiest processes on a Linux server by sorting the output of this command directly on the third column, use the command `ps aux | sort -k3`.

```
[root@server1:~]# ps aux | sort -k3 | tail -n 5
polkitd     1194   0.0   1.1  615104  11944 ?          Ssl
22:57   0:00 /usr/lib/polkit-1/polkitd --no-debug
root         1689   0.0   1.9  583876  19716 ?          Ssl
22:57   0:00 /usr/bin/python2 -Es /usr/sbin/tuned -l -P
root         1248   0.0   2.9  358760  29516 ?          Ssl
22:57    0:00 /usr/bin/python2 -Es /usr/sbin/firewalld --
nofork --nopid
root         2092   0.1   0.0       0      0 ?          S
23:47   0:00 [kworker/0:2]
USER          PID %CPU %MEM     VSZ    RSS TTY        STAT
START   TIME COMMAND
```

1.6. Counting lines, words, and characters with wc

When working with text files, you sometimes get a large amount of output. Before deciding which approach works best in a specific case, you might want to have an idea about the amount of text you are dealing with. In that case, the `wc` command is useful. In its output, this command gives three different results: the number of lines, the number of words, and the number of characters.

Consider, for example, the `ps aux` command. When executed as `root`, this command gives a list of all processes running on a server. One solution to count how many processes there are exactly is to pipe the output of `ps aux` through `wc`, as in `ps aux | wc`. You can see the result of the command in the listing below, where you can see that the total number of lines is 90 and that there are 1,045 words and 7,583 characters in the command output.

```
[root@server1 ~]# ps aux | wc
90 1045 7583
```

2. A primer to using regular expressions

Working with text files is an important skill for a Linux administrator. You not only have to know how to create and modify existing text files, but it is also very useful if you can find the text file that contains specific text.

It will be clear sometimes which specific text you are looking for. Other times, it might not. For example, are you looking for color or colour? Both spellings might give a match. This is just one example of why using flexible patterns while looking for text can prove useful. These flexible patterns are known as regular expressions in Linux.

To understand regular expressions a bit better, let's take a look at an example with the last six lines from the /etc/passwd file, and suppose that you are looking for the user anna. In that case, you could use the General Regular Expression Parser (grep) to look for that specific string in the file by using the command grep anna /etc/passwd. Listing below shows the results of that command, and as you can see, too many results are shown.

```
[root@server1 ~]# grep anna /etc/passwd
anna :x:1000:1000::/home/ anna :/bin/bash
rih anna :x:1001:1001::/home/rih anna :/bin/bash
anna bel:x:1002:1002::/home/ anna bel:/bin/bash
joanna:x:1004:1004::/home/jo anna :/bin/bash
```

So, a regular expression is a search pattern that allows you to look for specific text in an advanced and flexible way.

2.1. Types of regex

There are many different applications that use different types of regex in Linux, like the regex included in programming languages (Java, Perl, Python and so on) and Linux programs like (sed, awk, grep,) and many other applications.

A regex pattern uses a regular expression engine which translates those patterns. Linux has two regular expression engines:

- The Basic Regular Expression (BRE) engine
- The Extended Regular Expression (ERE) engine

Most Linux programs work well with BRE engine specifications, but some tools like sed understand some of the BRE engine rules. The awk command uses the ERE engine to process its regular expression patterns.

Since there are many regex implementations, it's difficult to write patterns that work on all engines. Hence, we will focus on the most commonly found regex.

2.2. Using line anchors

What if you want to look for lines that are starting with the text anna. The type of regular expression that specifies where in a line of output the result is expected is known as a line anchor. To show only lines that start with the text you are looking for, you can use the regular expression ^ (in this case, to indicate that you are looking only for lines where anna is at the beginning of the line). Listing below looks for lines starting with a specific pattern.

```
[root@server1 ~]# grep ^anna /etc/passwd
anna:x:1000:1000::/home/anna:/bin/bash
annabel:x:1002:1002::/home/annabel:/bin/bash
```

Another regular expression that relates to the position of specific text in a specific line is $, which states that the line ends with some text. For instance, the command grep ash$ /etc/passwd shows all lines in the /etc/passwd file that end with the text ash.

2.3. Using escaping in regular expressions

Although not mandatory, when using regular expressions, it is a good idea to use escaping to prevent the regular expression from being interpreted by the shell. In many cases, it is not really necessary to use escaping, in some cases, the regular expression fails without escaping. To prevent this from ever happening, it is a good idea to put the regular expression between quotes. So, instead of typing grep ^anna /etc/passwd, it is better to use grep '^anna' /etc/passwd, even if in this case both examples work.

2.4. Using wildcards and multipliers

In some cases, you might know which text you are looking for, but you might not know howm the specific text is written. Or you just want to use one regular expression to match different patterns. In those cases, wildcards and multipliers come in handy.

To start with, there is the. regular expression, this is used as a wildcard character to look for one specific character. So, the regular expression r.t would match the strings rat, rot, and rut.

In some cases, you might want to be more specific about the characters you are looking for. If that is the case, you can specify a range of characters that you are looking for. For instance, the regular expression r[aou]t matches the strings rat, rut, as well as rot.

Another useful regular expression is the multiplier *. This matches zero or more of the previous character. That does not seem to be very useful, but indeed it is. If you know exactly how many times the previous character will appear, you can specify this number also, as in re\{2\}d, which would match red as well as reed but not rered.

The last regular expression that is useful to know about is `the?`, which matches zero or one of the previous character. Table below provides an overview of the most important regular expressions.

Regular Expression	Use
^text	Line starts with text.
text$	Line ends with text.
.	Wildcard. (Matches any single character).
[abc]	Matches a, b, or c.
*	Match 0 to an infinite number of the previous character.
\{2\}	Match exactly 2 of the previous character.
\{1,3\}	Match a minimum of 1 and a maximum of 3 of the previous character.
colou?r	Match 0 or 1 of the previous character. This makes the previous character optional, which in this example would match both color and colour.

Let's take a look at an example of a regular expression that comes from the man page semanage-fcontext and relates to managing SELinux. The example line contains the regular expression: "/web (/.*)?"

In this regular expression, the text /web is referred to. This text string can be followed by the regular expression (/.*)?, which means zero or one of (/.*), which in fact means that it can be followed by nothing or (/.*). The (/.*) refers to a slash which may be followed by an unlimited number of characters. To state it differently, the regular expression refers to the text /web which may or may not be followed by any characters.

2.5. Using grep to analyze text

The ultimate utility to work with regular expressions is grep, quite a few examples that you have seen already were based on the grep command. The grep command has a couple of useful options to make it even more efficient. Table describes some of the most useful options.

Option	Use
–i	Not case sensitive. Matches uppercase as well as lowercase.
–v	Only show lines that do not contain the regular expression.
–r	Search files in the current directory and all subdirectories.
–e	Use this to search for lines matching more than one regular expression.
–A	<number> Show <number> of lines after the matching regular expression.
–B	<number> Show <number> of lines before the matching regular expression.

Exercise: Using common grep options

In this exercise, you work through some common grep options.

1. Type grep '^#' /etc/sysconfig/sshd. This shows that the file /etc/sysconfig/sshd contains a number of lines that start with the comment sign #.
2. To view the configuration lines that really matter, type grep -v '^#' /etc/sysconfig/sshd. This shows only lines that do not start with a #.
3. Now type grep -v '^#' /etc/sysconfig/sshd -B 5. This shows lines that are not starting with a # sign but also the five lines that are directly before that line, which is useful because in these lines you'll typically find comments on how to use the specific parameters. However, you'll also see that many blank lines are displayed.
4. Type grep -v -e '^#' -e '^$' /etc/sysconfig/sshd. This excludes all blank lines and lines that start with #.

3. Other useful text processing utilities

The grep utility is a powerful utility that allows you to work with regular expressions. It is not the only utility, though. Some even more powerful utilities exist, like awk and sed. Both utilities are extremely rich and merit a book by themselves.

3.1. Learning Awk basics

Awk is a scripting language, abbreviated from the names of the developers–Aho, Weinberger, and Kernighan, and is used for manipulating data and generating reports. The awk command programming language allows the user to use variables, numeric functions, string functions, and logical operators to transform data files and produce formatted reports.

3.1.1. Awk syntax and options

To define an awk script, use braces surrounded by single quotation marks like this: awk OPTIONS… 'selection_criteria {action}' input-file > output-file.

For example: $ awk '{print "Hello AWK"}' prints the "Hello AWK" statement to the screen. Awk can take the following options:

- -F fs: To specify a file separator
- -f file: To specify a file that contains awk script
- -v var=value: To declare a variable

3.1.2. Common use and examples

The most useful use cases are summarized in the following examples:

awk –F: '{print $4}' /etc/passwd This command shows the fourth column from /etc/passwd. This is something that can be done by using the cut utility as well, but the awk utility is more successful in distinguishing the fields that are used in command output of files. The bottom line is that if cut does not work, you should try the awk utility.

You can also use the awk utility to do tasks that you might be used to using grep for. Consider the following example: awk -F: '/user/ {print $4}' /etc/passwd. This command searches the /etc/passwd file for the text user and will print the fourth field of any matching line. It's important to mention that you can search any regular expressions using awk instead of just words.

The awk scripting language supports if conditional statement, suppose a file contains random numbers each in a line, the command: $ awk '{if ($1 > 30) print $1}' testfile will print out only the lines that are greater than 30. And so far, awk supports much more complex conditional and loop statements.

3.2. Learning Sed basics

SED command in UNIX stands for stream editor and it can perform lots of functions on file like, searching, find and replace, insertion or deletion. Though the most common use of SED command in UNIX is for substitution or for find and replace. By using SED you can edit files even without opening it, it can do insertion, deletion, search and replace(substitution). Also, it supports regular expression which allows it perform complex pattern matching.

3.2.1. Sed syntax and options

General syntax is: sed OPTIONS... [SCRIPT] [INPUTFILE...]

For example: $sed 's/unix/linux/' testfile.txt, replaces the word “unix” with “linux” in the testfile.txt and puts the output to the screen. Most common options SED can take are:

- -i: To specify files are to be edited in-place
- -n: To suppress automatic printing of patterns
- -f file: To specify a file that contains sed script
- -E -r: To use extended regular expressions rather than basic regular expressions

3.2.2. Common used examples

In this example, sed -n 5p /etc/passwd, sed is used to print the fifth line from the /etc/passwd file. The sed utility is a very powerful utility for filtering text from text files (like grep), but it has the benefit that it also allows you to apply modifications to text files, as shown in the following example: sed -i s/old-text/new-text/g ~/myfile. In this example the sed utility is used to search the text old-text in ~/myfile and on all occurrences replace it with the text new-text.

Notice that the default sed behavior is to write the output to STDOUT, but the option -i will write the result directly to the file. Make sure that you know what you are doing before using this command, because it might be difficult to revert file modifications that are applied in this way.

With this command: sed -i -e '2d' ~/myfile, you can delete a line based on a specific line number. You can also make more complicated references to line numbers. Use, for instance, sed -i -e '2d, 20, 25d' ~/myfile to delete lines 2 and 20 through 25 in the file ~/myfile. Also you can delete lines according to matching patterns: $ sed '/second/, /fourth/d' myfile, which deletes lines from the second to the fourth in a file contains the following:

```
$ cat myfile
This is a first line test.
This is a second line test.
This is a third line test.
This is a fourth line test.
This is a fifth line test.
```

When using sed to substitute texts within a file, a substitution flags are used in the syntax: sed 's/pattern/replacement/flags', where the flags might be one of the following: (g) to replace all occurrences, (A number) to replace the occurrence number, (p) to print the original content and (w) to write the results to a file. Example, $ sed 's/test/another test/2' myfile replaces the second occurrence of "test" word on each line.

Sed command processes your entire file. However, you can limit the sed command to process specific lines, there are two ways:

- A range of lines
- A pattern that matches a specific line

You can type one number to limit it to a specific line like this: $ sed '2s/test/another test/' myfile. Or you can specify a pattern like this: $ sed '/root/s/bash/csh/' /etc/passwd to substitute strings in the line containing the root key term. Surely, you can use regular expressions to write this pattern to be more generic and useful.

You can insert or append text lines using the following flags:

- **The (i) flag**: enables inserting text before a specific line in a file, for example: $ sed '2i\This is the inserted line.' myfile, will insert the text before the second line in the file
- **The (a) flag**: will append text after a specific line in a file, for example: $ sed '2a\This is the inserted line.' myfile, will insert the text after the second line in the file

Finally, you can modify a specific line using the (c) flag like this: $ sed '3c\This is a modified line.' myfile to modify the 3rd line with "This is a modified line." phrase. Much more, you can modify lines according to matching patterns like in deletion: $ sed '/This is/c Line updated.' myfile, which modifies each line contains "This is" with a line of the "Line updated." phrase.

Quiz

Chapter review questions

1. Which command was developed to show only the first 10 lines in a text file?
   a. head
   b. top
   c. first
   d. cat
2. Which command enables you to count the number of words in a text file?
   a. count
   b. list
   c. ls –l
   d. wc
3. Which key on your keyboard do you use in less to go to the last line of the current text file?
   a. End
   b. PageDown
   c. q
   d. G
4. Which option is missing from the following command, assuming that you want to filter the first field out of the /etc/passwd file and assuming that the character that is used as the field delimiter is a:? cut ….: –f 1 /etc/passwd
   a. – d
   b. – c
   c. – t
   d. – x

5. Which option is missing if you want to sort the third column of the output of the command ps aux?

   ps aux | sort

   a. – k 3

   b. – s 3

   c. – k f 3

   d. – f 3

6. Which of the following lines would only show lines in the file /etc/passwd that start with the text anna?

   a. grep anna /etc/passwd

   b. grep –v anna /etc/passwd

   c. grep $ anna /etc/passwd

   d. grep ^ anna /etc/passwd

7. Which regular expression do you use to make the previous character optional?

   a. ?

   b. .

   c. *

   d. &

8. Which regular expression is used as a wildcard to refer to any single character?

   a. ?

   b. .

   c. *

   d. &

9. Which command prints the fourth field of a line in the /etc/passwd file if the text user occurs in that line?

   a. awk '/user/ {print $4}' /etc/passwd

   b. awk -d: '/user/ {print $4}' /etc/passwd

   c. awk -F: '/user/ $4' /etc/passwd

   d. awk -F: '/user/ {print $4}' /etc/passwd

10. Which option would you use with grep to show only lines that do not contain the regular expression that was used?

   a. -x

   b. -v

   c. -u

   d. -q

11. Which command enables you to see the results of the ps aux command in a way that you can easily browse up and down in the results?

   a. ps aux | cat

   b. ps aux | grep

   c. ps aux | awk

   d. ps aux | less

12. Which command enables you to show the last five lines from ~/samplefile?

   a. cat ~/samplefile -n 5

   b. more -5 ~/samplefile

   c. tail -5 ~/samplefile

   d. head 5 ~/samplefile

13. Which command do you use if you want to know how many words are in ~/samplefile?

a. wc –w ~/samplefile

b. head ~/samplefile

c. more –l ~/samplefile

d. cat w ~/samplefile

14. After opening command output using tail –f ~/mylogfile, how do you stop showing output?

a. Ctrl+v

b. Ctrl+c

c. q

d. Alt+C

15. Which grep option do you use to exclude all lines that are starting with either a # or a?

a. grep –v –e '^#' –e '^$'

b. grep –e '^#' –e '^;'

c. grep –v –e '^#;'

d. grep –v –e '^#' –e '^;'

16. Which regular expression do you use to match one or more of the preceding characters?

a. ?

b. .

c. *

d. +

17. Which grep command enables you to see text as well as TEXT in a file?

a. grep –i text file

b. grep –r text file

c. tai text file

d. more text file

18. Which grep command enables you to show all lines starting with PATH, as well as the five lines just before that line?

a. grep $PATH –B 5

b. grep ^PATH –A 5

c. grep ^PATH –B 5

d. grep –v PATH –n 5

19. Which sed command do you use to show line 9 from ~/samplefile?

a. sed –n 9p simplefile

b. sed 9p simplefile

c. sed 9 simplefile

d. sed nine simplefile

20. Which command enables you to replace the word user with the word users in ~/samplefile?

a. awk 's/user/users/' ~/samplefile

b. sed 's/user/users/' ~/samplefile

c. sed –v /user/users/ ~/samplefile

d. sed 's/users/user/' ~/samplefile

Answers to chapter Review Questions:

1. a
2. d
3. d
4. a
5. a
6. d
7. a
8. b
9. d
10. b
11. d
12. c
13. a
14. b
15. d
16. d
17. a
18. c
19. a
20. b