Chapter 13

Command line tools for manipulating text files

Irrespective of the role you play with Linux (system administrator, developer or user), you often need to browse through and parse text files, and/or extract data from them. These are file manipulation operations. Thus, it is essential for the Linux user to become adept at performing certain operations on files.

Most of the time, such file manipulation is done at the command line, which allows users to perform tasks more efficiently than while using a GUI.

cat

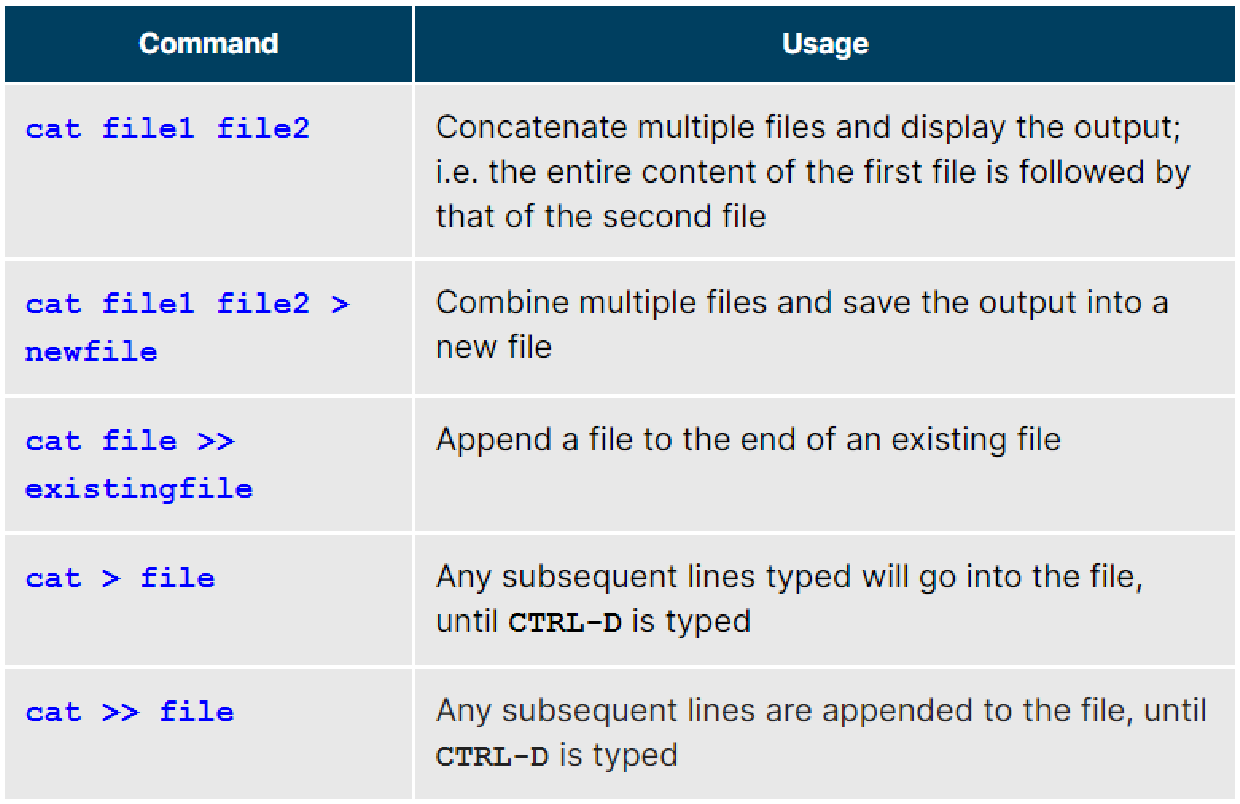
cat is short for ***concatenate*** and is one of the most frequently used Linux command line utilities. It is often used to read and print files, as well as for simply viewing file contents. To view a file, use the following command:

```
$ cat <filename>
```

For example, **cat readme.txt** will display the contents of **readme.txt** on the terminal. However, the main purpose of **cat** is often to combine (concatenate) multiple files together. You can perform the actions listed in the table using **cat**.

The **tac** command (**cat** spelled backwards) prints the lines of a file in reverse order. Each line remains the same, but the order of lines is inverted. The syntax of **tac** is exactly the same as for **cat**, as in:

```
$ tac file
$ tac file1 file2 > newfile
```



Command	Usage
`cat file1 file2`	Concatenate multiple files and display the output; i.e. the entire content of the first file is followed by that of the second file
`cat file1 file2 > newfile`	Combine multiple files and save the output into a new file
`cat file >> existingfile`	Append a file to the end of an existing file
`cat > file`	Any subsequent lines typed will go into the file, until `CTRL-D` is typed
`cat >> file`	Any subsequent lines are appended to the file, until `CTRL-D` is typed

Using cat interactively

cat can be used to read from standard input (such as the terminal window) if no files are specified. You can use the **>** operator to create and add lines into a new file, and the **>>** operator to append lines (or files) to an existing file. We mentioned this when talking about how to create files without an editor.

To create a new file, at the command prompt type **cat > <filename>** and press the **Enter** key.

This command creates a new file and waits for the user to edit/enter the text. After you finish typing the required text, press **CTRL-D** at the beginning of the next line to save and exit the editing.

Another way to create a file at the terminal is **cat > <filename> << EOF**. A new file is created and you can type the required input. To exit, enter **EOF** at the beginning of a line.

Note that **EOF** is case sensitive. One can also use another word, such as **STOP**.

echo

echo simply displays (echoes) text. It is used simply, as in:

$ echo string

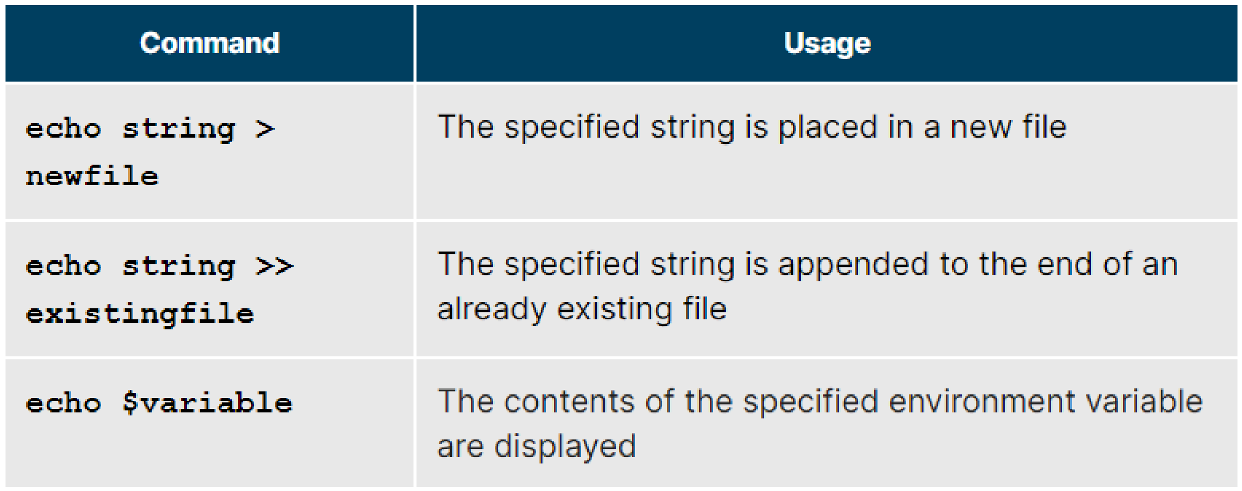
echo can be used to display a string on standard output (i.e. the terminal) or to place in a new file (using the **>** operator) or append to an already existing file (using the **>>** operator).

The **–e** option, along with the following switches, is used to enable special character sequences, such as the newline character or horizontal tab.

- **\n** represents newline
- **\t** represents horizontal tab.

echo is particularly useful for viewing the values of environment variables (built-in shell variables). For example, **echo $USERNAME** will print the name of the user who has logged into the current terminal.

The following table lists **echo** commands and their usage:



Command	Usage
`echo string > newfile`	The specified string is placed in a new file
`echo string >> existingfile`	The specified string is appended to the end of an already existing file
`echo $variable`	The contents of the specified environment variable are displayed

Working with large files

System administrators need to work with configuration files, text files, documentation files, and log files. Some of these files may be large or become quite large as they accumulate data with time. These files will require both viewing and administrative updating. In this section, you will learn how to manage such large files.

For example, a banking system might maintain one simple large log file to record details of all of one day's ATM transactions. Due to a security attack or a malfunction, the administrator might be forced to check for some data by navigating within the file. In such cases, directly opening the file in an editor will cause issues, due to high memory utilization, as an editor will usually try to read the whole file into memory first. However, one can use **less** to view the contents of such a large file, scrolling up and down page by page, without the system having to place the entire file in memory before starting. This is much faster than using a text editor.

Viewing **somefile** can be done by typing either of the two following commands:

```
$ less somefile
$ cat somefile | less
```

By default, **man** pages are sent through the **less** command. You may have encountered the older **more** utility which has the same basic function but fewer capabilities:
i.e. **less** is **more**!

head

head reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines in an option.

For example, if you want to print the first 5 lines from **grub.cfg**, use the following command:

```
$ head –n 5 grub.cfg
```

You can also just say **head -5 grub.cfg**.

tail

tail prints the last few lines of each named file and displays it on standard output. By default, it displays the last 10 lines. You can give a different number of lines as an option. **tail** is especially useful when you are troubleshooting any issue using log files, as you probably want to see the most recent lines of output.

For example, to display the last 15 lines of **somefile.log**, use the following command:

```
$ tail -n 15 somefile.log
```

*You can also just say **tail -15 somefile.log**.

To continually monitor new output in a growing log file:

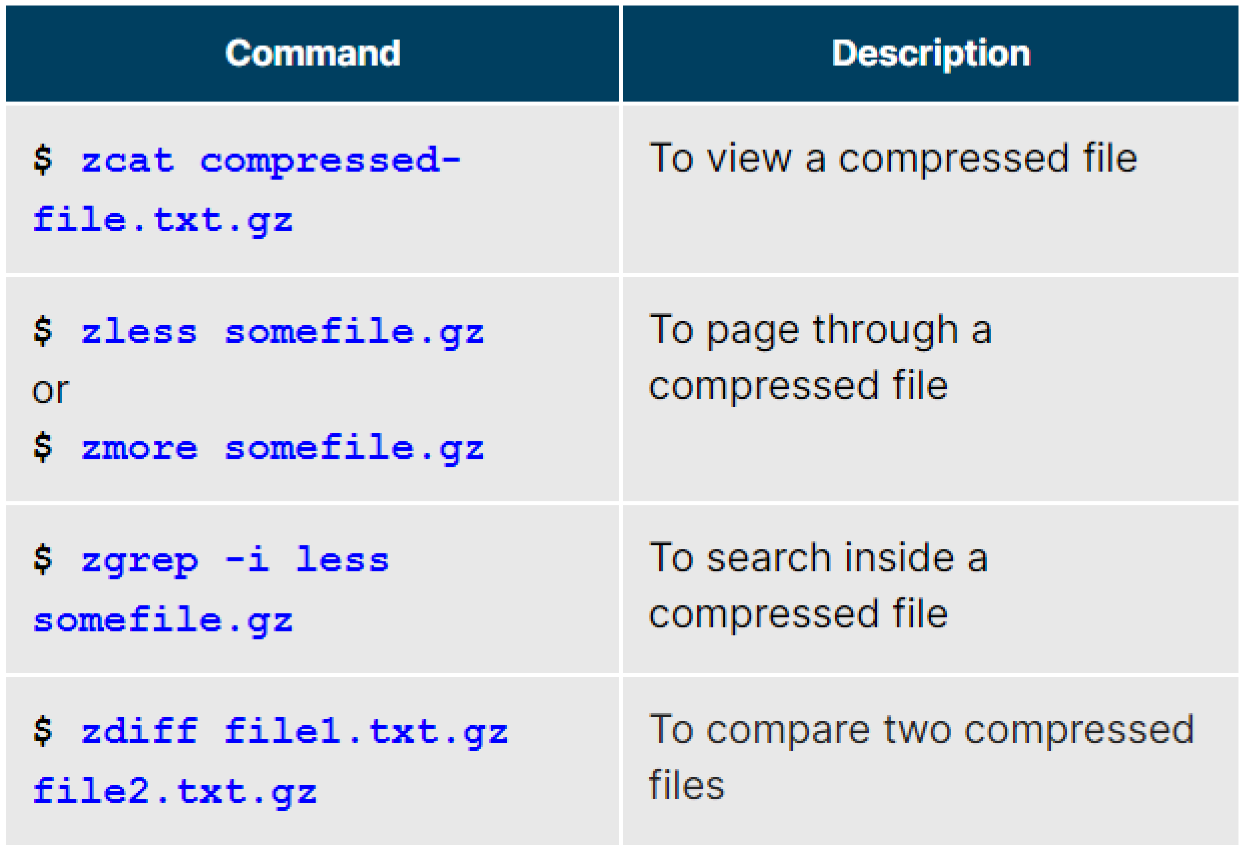
```
$ tail -f somefile.log
```

This command will continuously display any new lines of output in **atmtrans.log** as soon as they appear. Thus, it enables you to monitor any current activity that is being reported and recorded.

Viewing compressed files

When working with compressed files, many standard commands cannot be used directly. For many commonly-used file and text manipulation programs, there is also a version especially designed to work directly with compressed files. These associated utilities have the letter "z" prefixed to their name. For example, we have utility programs such as **zcat**, **zless**, **zdiff** and **zgrep**.

Here is a table listing some **z** family commands:



Command	Description
`$ zcat compressed-file.txt.gz`	To view a compressed file
`$ zless somefile.gz` or `$ zmore somefile.gz`	To page through a compressed file
`$ zgrep -i less somefile.gz`	To search inside a compressed file
`$ zdiff file1.txt.gz file2.txt.gz`	To compare two compressed files

Note that if you run **zless** on an uncompressed file, it will still work and ignore the decompression stage. There are also equivalent utility programs for other compression methods besides **gzip**, for example, we have **bzcat** and **bzless** associated with **bzip2**, and **xzcat** and **xzless** associated with **xz**.

Introduction to sed and awk

It is very common to create and then repeatedly edit and/or extract contents from a file. Let's learn how to use **sed** and **awk** to easily perform such operations.

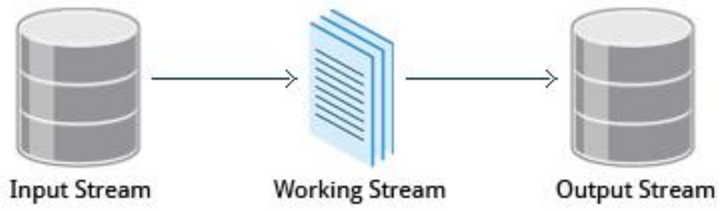
Note that many Linux users and administrators will write scripts using comprehensive scripting languages such as Python and perl, rather than use **sed** and **awk** (and some other utilities we will discuss later). Using such utilities is certainly fine in most circumstances; one should always feel free to use the tools one is experienced with. However, the utilities that are described here are much lighter; i.e. they use fewer system resources, and execute faster. There are situations (such as during booting the system) where a lot of time would be wasted using the more complicated tools, and the system may not even be able to run them. So, the simpler tools will always be needed.

sed

sed is a powerful text processing tool and is one of the oldest, earliest and most popular UNIX utilities. It is used to modify the contents of a file or input stream, usually placing the contents into a new file or output stream. Its name is an abbreviation for **s**tream **ed**itor.

sed can filter text, as well as perform substitutions in data streams.

Data from an input source/file (or stream) is taken and moved to a working space. The entire list of operations/modifications is applied over the data in the working space and the final contents are moved to the standard output space (or stream).



sed commands syntax

Command	Usage
`sed -e command <filename>`	Specify editing commands at the command line, operate on file and put the output on standard out (e.g. the terminal)
`sed -f scriptfile <filename>`	Specify a scriptfile containing **sed** commands, operate on file and put output on standard out

The **-e** option allows you to specify multiple editing commands simultaneously at the command line. It is unnecessary if you only have one operation invoked.

sed basic operations

Now that you know that you can perform multiple editing and filtering operations with **sed**, let's explain some of them in more detail. The table explains some basic operations, where **pattern** is the current string and **replace_string** is the new string:

Command	Usage
`sed s/pattern/replace_string/ file`	Substitute first string occurrence in every line
`sed s/pattern/replace_string/g file`	Substitute all string occurrences in every line
`sed 1,3s/pattern/replace_string/g file`	Substitute all string occurrences in a range of lines
`sed -i s/pattern/replace_string/g file`	Save changes for string substitution in the same file

You must use the **-i** option with care, because the action is not reversible. It is always safer to use **sed** without the **–i** option and then replace the file yourself, as shown in the following example:

```
$ sed s/pattern/replace_string/g file1 > file2
```

The above command will replace all occurrences of **pattern** with **replace_string** in **file1** and move the contents to **file2**. The contents of **file2** can be viewed with **cat file2**. If you approve you can then overwrite the original file with **mv file2 file1**.

Example: To convert **01/02/… to JAN/FEB/…**

```
sed -e 's/01/JAN/' -e 's/02/FEB/' -e 's/03/MAR/' -e 's/04/APR/' -e 's/05/MAY/' \
   -e 's/06/JUN/' -e 's/07/JUL/' -e 's/08/AUG/' -e 's/09/SEP/' -e 's/10/OCT/' \
   -e 's/11/NOV/' -e 's/12/DEC/'
```

awk

awk is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

awk has the following features:

- It is a powerful utility and interpreted programming language.
- It is used to manipulate data files, retrieving, and processing text.
- It works well with fields (containing a single piece of data, essentially a column) and records (a collection of fields, essentially a line in a file).

As with **sed**, short **awk** commands can be specified directly at the command line, but a more complex script can be saved in a file that you can specify using the **-f** option.

Command	Usage
`awk 'command' file`	Specify a command directly at the command line
`awk -f scriptfile file`	Specify a file that contains the script to be executed

awk basic operations

The table explains the basic tasks that can be performed using **awk**. The input file is read one line at a time, and, for each line, **awk** matches the given pattern in the given order and performs the requested action. The **-F** option allows you to specify a particular field separator character. For example, the **/etc/passwd** file uses "**:**" to separate the fields, so the **-F:** option is used with the **/etc/passwd** file.

The command/action in **awk** needs to be surrounded with apostrophes (or single-quote (')). **awk** can be used as follows:

Command	Usage
`awk '{ print $0 }' /etc/passwd`	Print entire file
`awk -F: '{ print $1 }' /etc/passwd`	Print first field (column) of every line, separated by a space
`awk -F: '{ print $1 $7 }' /etc/passwd`	Print first and seventh field of every line

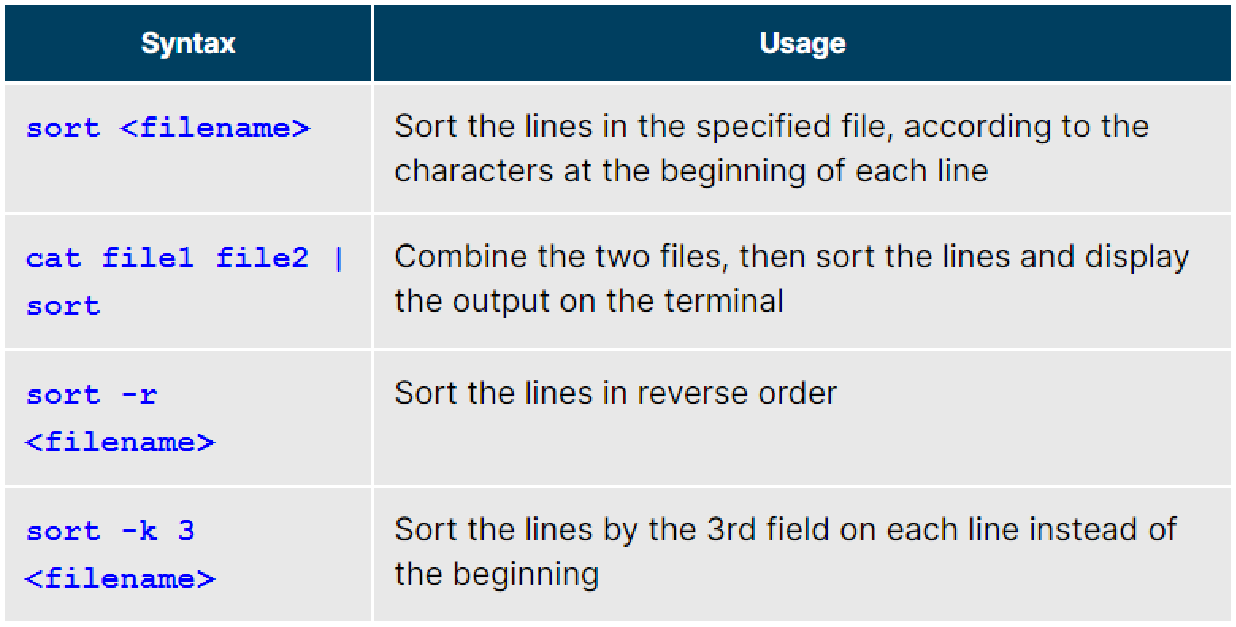
File manipulation utilities

In managing your files, you may need to perform tasks such as sorting data and copying data from one location to another. Linux provides numerous file manipulation utilities that you can use while working with text files. In this section, you will learn about the following file manipulation programs:

- **sort**
- **uniq**
- **paste**
- **join**
- **split**.

sort

sort is used to rearrange the lines of a text file, in either ascending or descending order according to a sort key. You can also sort with respect to particular fields (columns) in a file. The default sort key is the order of the ASCII characters (i.e. essentially alphabetically).



Syntax	Usage
`sort <filename>`	Sort the lines in the specified file, according to the characters at the beginning of each line
`cat file1 file2 | sort`	Combine the two files, then sort the lines and display the output on the terminal
`sort -r <filename>`	Sort the lines in reverse order
`sort -k 3 <filename>`	Sort the lines by the 3rd field on each line instead of the beginning

When used with the **-u** option, **sort** checks for unique values after sorting the records (lines). It is equivalent to running **uniq** (which we shall discuss) on the output of sort.

uniq

uniq removes duplicate consecutive lines in a text file and is useful for simplifying the text display.

Because **uniq** requires that the duplicate entries must be consecutive, one often runs sort first and then pipes the output into **uniq**; if sort is used with the **-u** option, it can do all this in one step.

To remove duplicate entries from multiple files at once, use the following command:

```
sort file1 file2 | uniq > file3
```

or

```
sort -u file1 file2 > file3
```

To count the number of duplicate entries, use the following command:

```
uniq -c filename
```

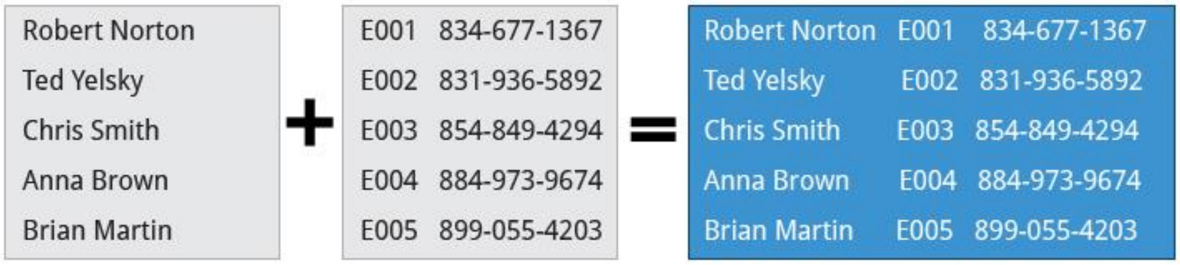
paste

Suppose you have a file that contains the full name of all employees and another file that lists their phone numbers and Employee IDs. You want to create a new file that contains all the data listed in three columns: name, employee ID, and phone number. How can you do this effectively without investing too much time?

paste can be used to create a single file containing all three columns. The different columns are identified based on delimiters (spacing used to separate two fields). For example, delimiters can be a blank space, a tab, or an **Enter**. In the image provided, a single space is used as the delimiter in all files.

paste accepts the following options:

- **-d** delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, **paste** begins again at the first delimiter.
- **-s**, which causes paste to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.



```
Robert Norton       E001   834-677-1367       Robert Norton  E001   834-677-1367
Ted Yelsky          E002   831-936-5892       Ted Yelsky      E002   831-936-5892
Chris Smith    +    E003   854-849-4294   =   Chris Smith     E003   854-849-4294
Anna Brown          E004   884-973-9674       Anna Brown      E004   884-973-9674
Brian Martin        E005   899-055-4203       Brian Martin    E005   899-055-4203
```

Using paste

paste can be used to combine fields (such as name or phone number) from different files, as well as combine lines from multiple files. For example, line one from **file1** can be combined with line one of **file2**, line two from **file1** can be combined with line two of **file2**, and so on.

To paste contents from two files one can do:

```
$ paste file1 file2
```

The syntax to use a different delimiter is as follows:

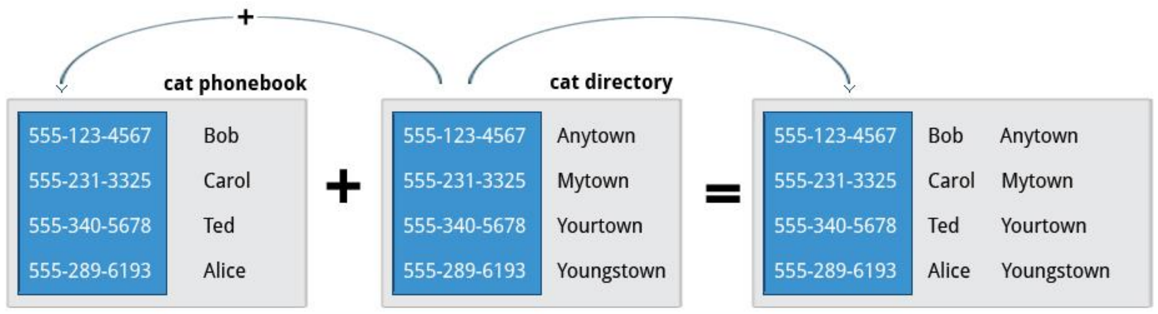
```
$ paste -d, file1 file2
```

Common delimiters are 'space', 'tab', '|', 'comma', etc.

join

Suppose you have two files with some similar columns. You have saved employees' phone numbers in two files, one with their first name and the other with their last name. You want to combine the files without repeating the data of common columns. How do you achieve this?

The above task can be achieved using **join**, which is essentially an enhanced version of **paste**. It first checks whether the files share common fields, such as names or phone numbers, and then joins the lines in two files based on a common field.



Using join

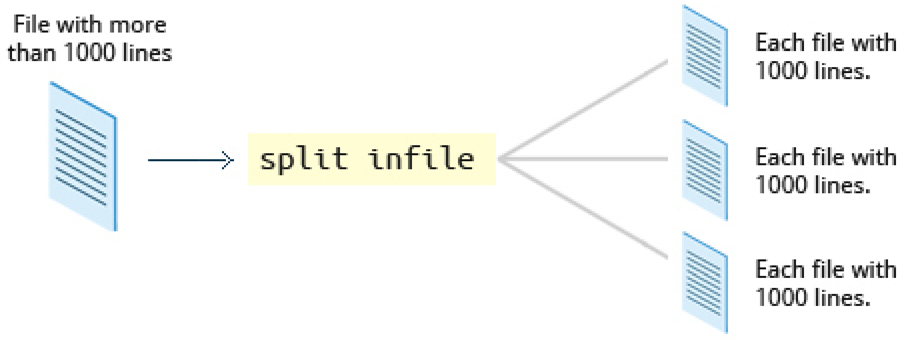
To combine two files on a common field, at the command prompt type **join file1 file2** and press the **Enter** key.

For example, the common field (i.e. it contains the same values) among the **phonebook** and **cities** files is the phone number, and the result of joining these two files is shown in the screen capture.

split

To combine two files on a common field, at the command prompt type **join file1 file2** and press the **Enter** key.

For example, the common field (i.e. it contains the same values) among the **phonebook** and **cities** files is the phone number, and the result of joining these two files is shown in the screen capture.


File with more than 1000 lines
split infile
Each file with 1000 lines.
Each file with 1000 lines.
Each file with 1000 lines.

Using split

We will apply **split** to an American-English dictionary file of over 99,000 lines:

```
$ wc -l american-english
99171 american-english
```

where we have used **wc** (word count, soon to be discussed) to report on the number of lines in the file. Then, typing:

```
$ split american-english dictionary
```

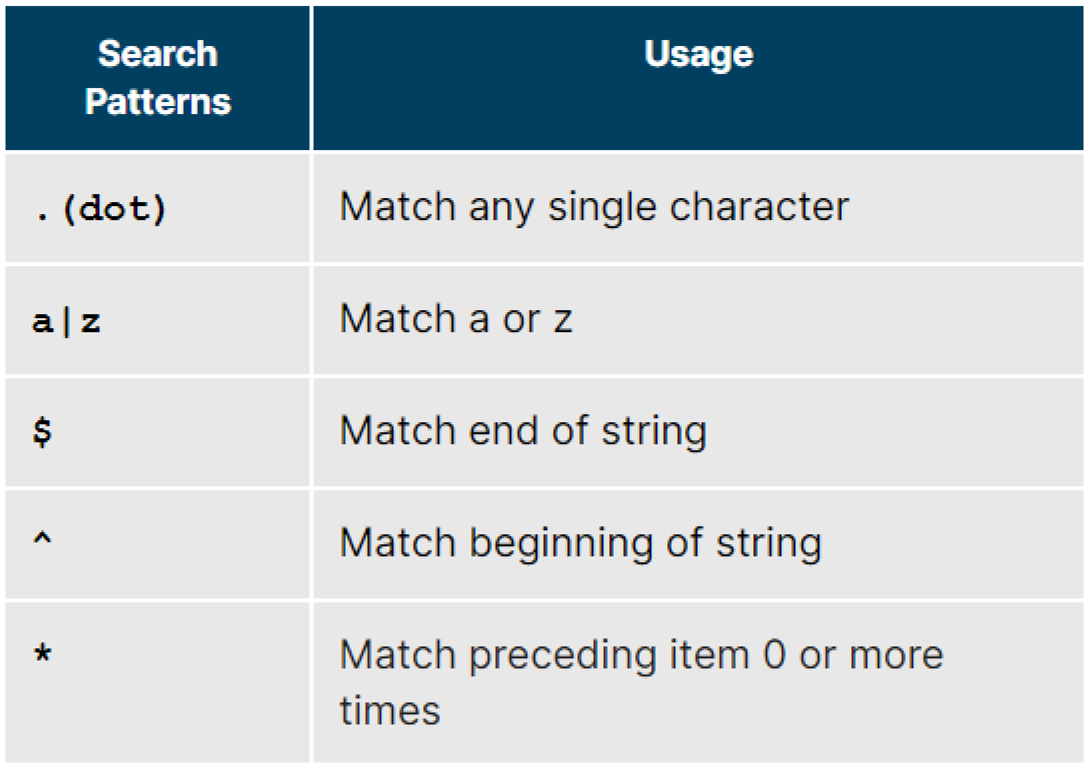
will split the American-English file into 100 equal-sized segments named **dictionary*xx***. The last one will of course be somewhat smaller.

Regular expressions and search patterns

Regular expressions are text strings used for matching a specific pattern, or to search for a specific location, such as the start or end of a line or a word. Regular expressions can contain both normal characters or so-called meta-characters, such as ***** and **$**.

Many text editors and utilities such as **vi**, **sed**, **awk**, **find** and **grep** work extensively with regular expressions. Some of the popular computer languages that use regular expressions include Perl, Python and Ruby. It can get rather complicated and there are whole books written about regular expressions; thus, we will do no more than skim the surface here.

These regular expressions are different from the wildcards (or meta-characters) used in filename matching in command shells such as bash (which were covered in *Chapter 7: Command-Line Operations*)**.** The table lists search patterns and their usage.

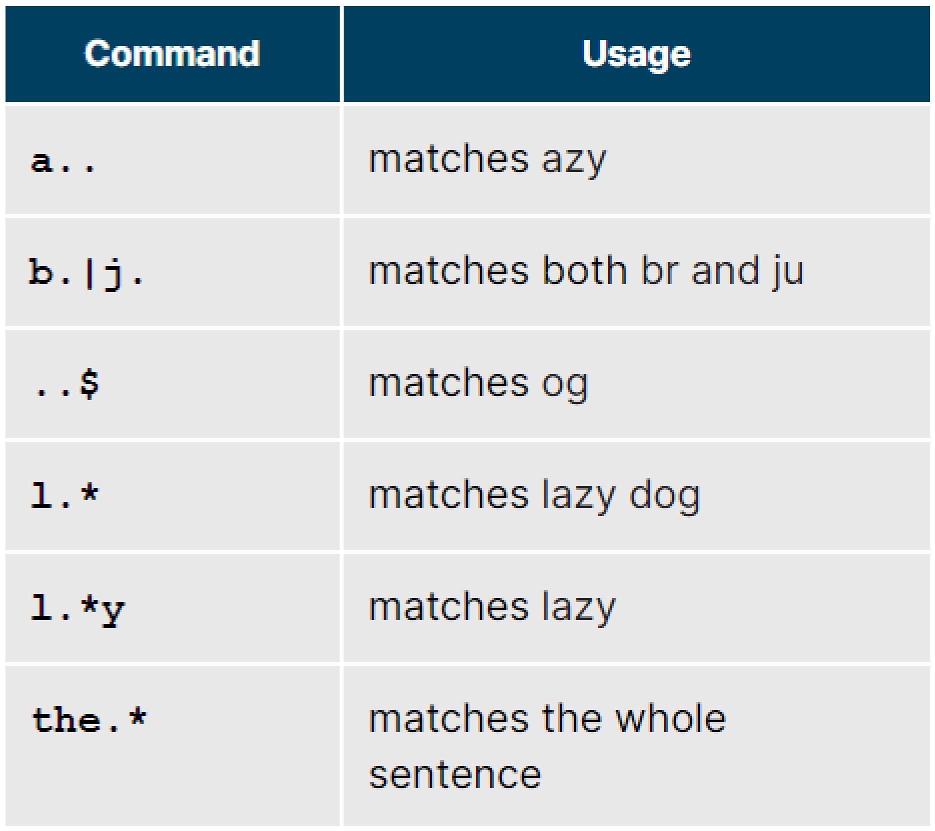


Search Patterns	Usage
`. (dot)`	Match any single character
`a|z`	Match a or z
`$`	Match end of string
`^`	Match beginning of string
`*`	Match preceding item 0 or more times

Using regular expressions and search patterns

For example, consider the following sentence: **the quick brown fox jumped over the lazy dog**.

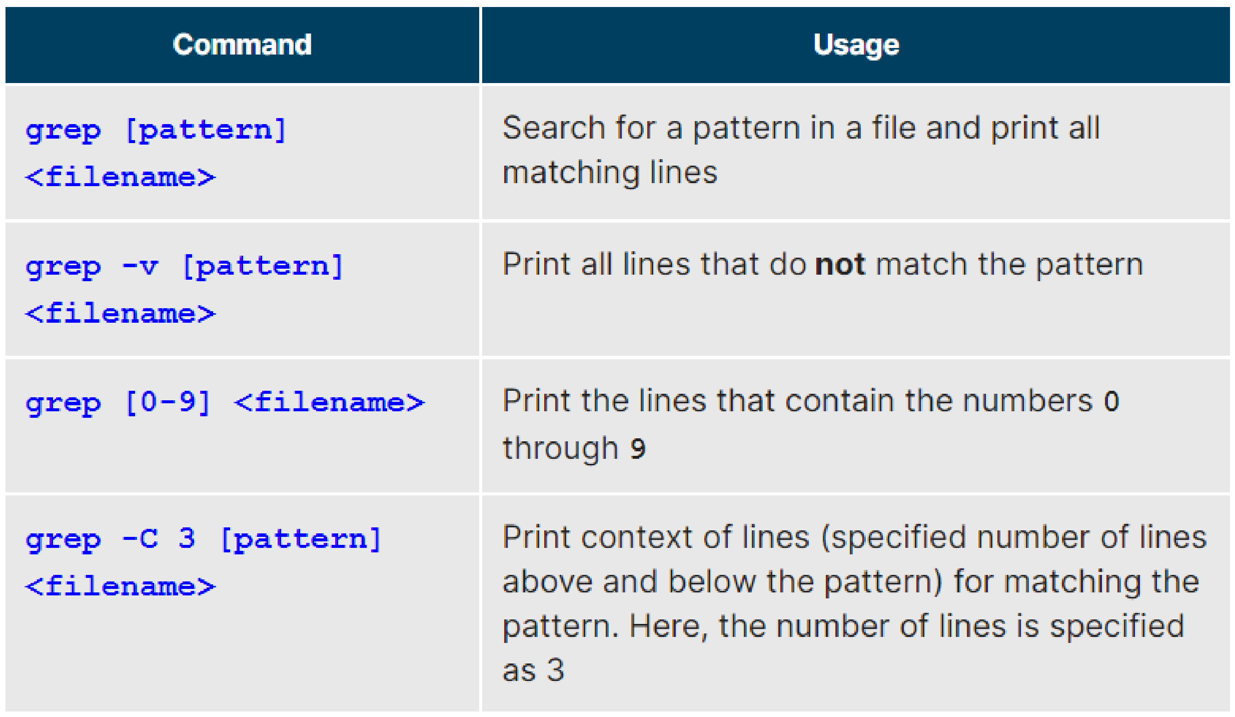
Some of the patterns that can be applied to this sentence are as follows:



Command	Usage
`a..`	matches azy
`b.|j.`	matches both br and ju
`..$`	matches og
`l.*`	matches lazy dog
`l.*y`	matches lazy
`the.*`	matches the whole sentence

grep

grep is extensively used as a primary text searching tool. It scans files for specified patterns and can be used with regular expressions, as well as simple strings, as shown in the table:



Command	Usage
`grep [pattern] <filename>`	Search for a pattern in a file and print all matching lines
`grep -v [pattern] <filename>`	Print all lines that do **not** match the pattern
`grep [0-9] <filename>`	Print the lines that contain the numbers `0` through `9`
`grep -C 3 [pattern] <filename>`	Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here, the number of lines is specified as 3

strings

strings is used to extract all printable character strings found in the file or files given as arguments. It is useful in locating human-readable content embedded in binary files; for text files one can just use **grep**.

For example, to search for the string **my_string** in a spreadsheet:

```
$ strings book1.xls | grep my_string
```

The screenshot shows a search of a number of programs to see which ones have GPL licenses of various versions.

Command: tr

In this section, you will learn about some additional text utilities that you can use for performing various actions on your Linux files, such as changing the case of letters or determining the count of words, lines, and characters in a file.

The **tr** utility is used to translate specified characters into other characters or to delete them. The general syntax is as follows:

```
$ tr [options] set1 [set2]
```

The items in the square brackets are optional. **tr** requires at least one argument and accepts a maximum of two. The first, designated **set1** in the example, lists the characters in the text to be replaced or removed. The second, **set2**, lists the characters that are to be substituted for the characters listed in the first argument. Sometimes these sets need to be surrounded by apostrophes (or single-quotes (')) in order to have the shell ignore that they mean something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets as you will see in the examples below.

For example, suppose you have a file named **city** containing several lines of text in mixed case. To translate all lower case characters to upper case, at the command prompt type **cat city | tr a-z A-Z** and press the **Enter** key.

Command	Usage
`tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ`	Convert lower case to upper case
`tr '{}' '()' < inputfile > outputfile`	Translate braces into parenthesis
`echo "This is for testing" | tr [:space:] '\t'`	Translate white-space to tabs
`echo "This   is   for    testing" | tr -s [:space:]`	Squeeze repetition of characters using `-s`
`echo "the geek stuff" | tr -d 't'`	Delete specified characters using `-d` option
`echo "my username is 432234" | tr -cd [:digit:]`	Complement the sets using `-c` option
`tr -cd [:print:] < file.txt`	Remove all non-printable character from a file
`tr -s '\n' ' ' < file.txt`	Join all the lines in a file into a single line

Command: tee

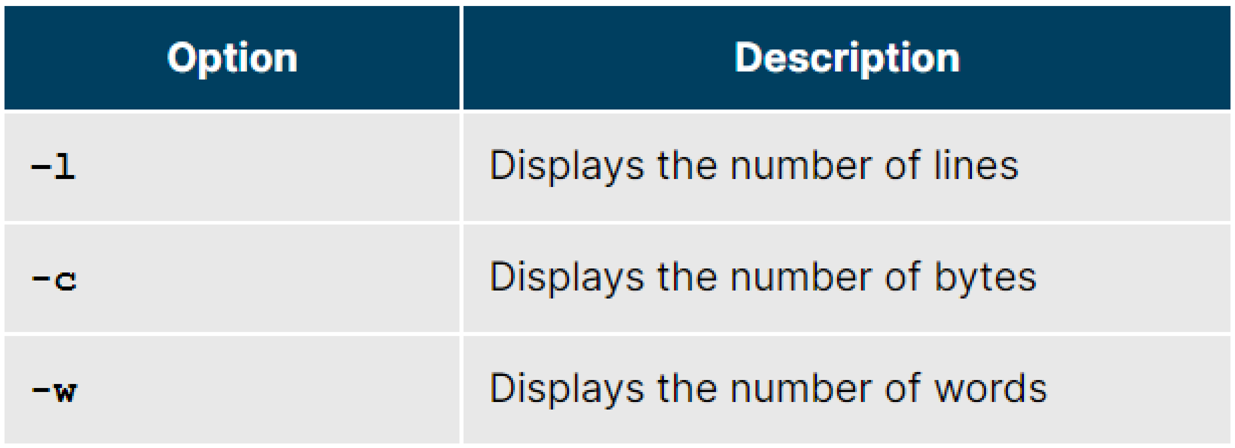
tee takes the output from any command, and, while sending it to standard output, it also saves it to a file. In other words, it ***tees*** the output stream from the command: one stream is displayed on the standard output and the other is saved to a file.

For example, to list the contents of a directory on the screen and save the output to a file, at the command prompt type **ls -l | tee newfile** and press the **Enter** key.

Typing **cat newfile** will then display the output of **ls –l**.

Command: wc

wc (**w**ord **c**ount) counts the number of lines, words, and characters in a file or list of files. Options are given in the table below.



Option	Description
`-l`	Displays the number of lines
`-c`	Displays the number of bytes
`-w`	Displays the number of words

By default, all three of these options are active.

For example, to print only the number of lines contained in a file, type **wc -l filename** and press the **Enter** key.

Command: cut

cut is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **tab** character. A different delimiter can be given as a command option.

For example, to display the third column delimited by a blank space, at the command prompt type **ls -l | cut -d" " -f3** and press the **Enter** key.

Summary

- The command line often allows the users to perform tasks more efficiently than the GUI.
- **cat**, short for concatenate, is used to read, print, and combine files.
- **echo** displays a line of text either on standard output or to place in a file.
- **sed** is a popular stream editor often used to filter and perform substitutions on files and text data streams.
- **awk** is an interpreted programming language, typically used as a data extraction and reporting tool.
- **sort** is used to sort text files and output streams in either ascending or descending order.

- **uniq** eliminates duplicate entries in a text file.
- **paste** combines fields from different files. It can also extract and combine lines from multiple sources.
- **join** combines lines from two files based on a common field. It works only if files share a common field.
- **split** breaks up a large file into equal-sized segments.
- Regular expressions are text strings used for pattern matching. The pattern can be used to search for a specific location, such as the start or end of a line or a word.
- **grep** searches text files and data streams for patterns and can be used with regular expressions.
- **tr** translates characters, copies standard input to standard output, and handles special characters.
- **tee** saves a copy of standard output to a file while still displaying at the terminal.
- **wc** (word count) displays the number of lines, words, and characters in a file or group of files.
- **cut** extracts columns from a file.
- **less** views files a page at a time and allows scrolling in both directions.
- **head** displays the first few lines of a file or data stream on standard output. By default, it displays 10 lines.
- **tail** displays the last few lines of a file or data stream on standard output. By default, it displays 10 lines.
- **strings** extracts printable character strings from binary files.
- The **z** command family is used to read and work with compressed files.