

Chapter 8

231011 Kevin Antonio Andrade López



universidad tecnologica de aguscalientes

# 8.1 Introduction

A large number of the files in a typical file system are text files. Text files contain only text, with no formatting features that you might see in a word processing file.

Since there are many of these files on a typical Linux system, there are a large number of commands to help users manipulate text files. There are commands to view and modify these files in various ways.

In addition, there are features available to the shell to control the output of commands, so instead of having the output in the terminal window, the output can be redirected to another file or another command. These redirection features offer users a more flexible and powerful environment to work in.

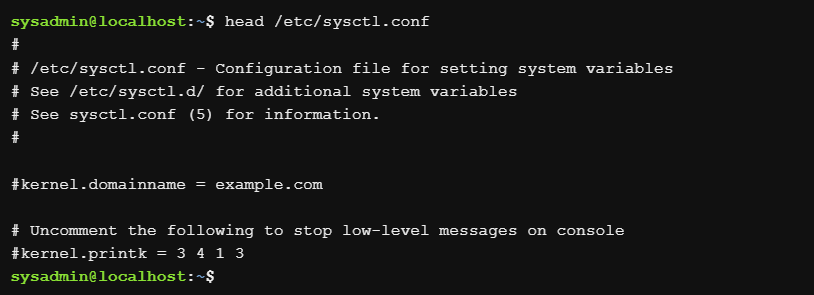
# 8.2 Vertical Bars on the Command Line

Previous chapters describe how to use individual commands to perform actions in the operating system, including how to create/move/delete files and move throughout the system. Usually, when a command provides output or generates an error, the output is displayed on the screen; however, this does not have to be the case.

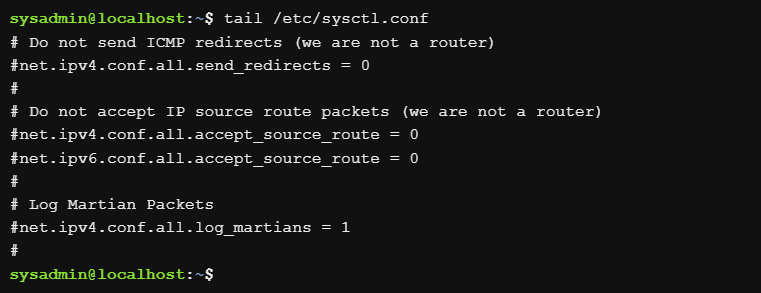
The vertical slash character | (or “pipe”) can be used to send output from one command to another. Instead of being printed to the screen, the output of one command becomes an input for the next command. This can be a powerful tool, especially in searching for specific data; the implementation of piping is often used to refine the results of an initial command.

The commands head and tail will be used in many examples below to illustrate the use of vertical bars. These commands can be used to display only some of the first or last lines of a file (or, when used with a vertical bar, the output of a previous command).

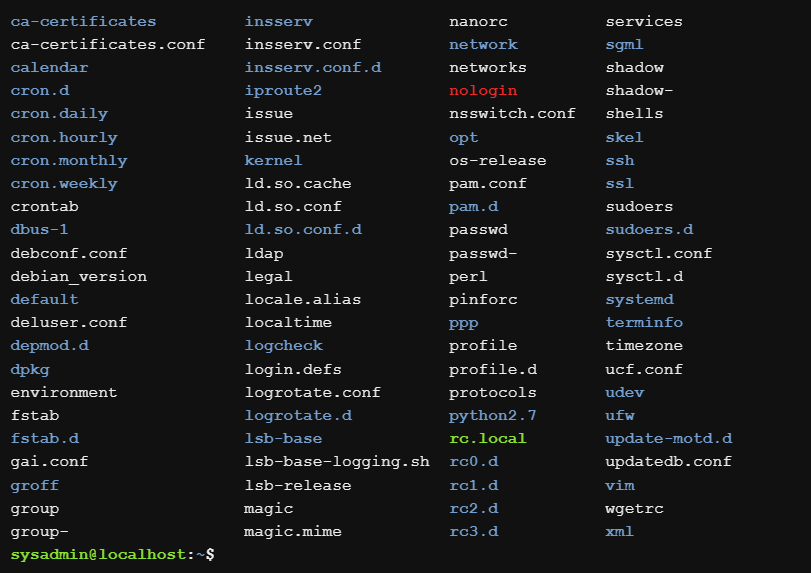
By default, the head and tail commands will display ten lines. For example, the following command displays the first ten lines of the /etc/sysctl.conf file:



In the following example, the last ten lines of the file will be displayed:

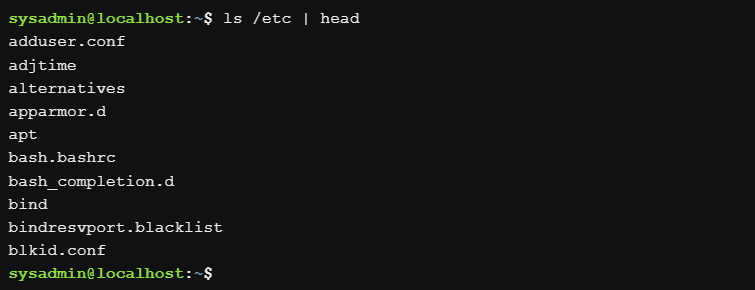


The vertical bar character allows users to use these commands not only in files, but also in the output of other commands. This can be useful when listing a large directory, for example the /etc directory:



If you look at the output of the above command, you will notice that that first file name is ca-certificates. But there are other files listed “above” that can only be seen if the user uses the scroll bar. What if you only want to list some of the first few files in the /etc directory?

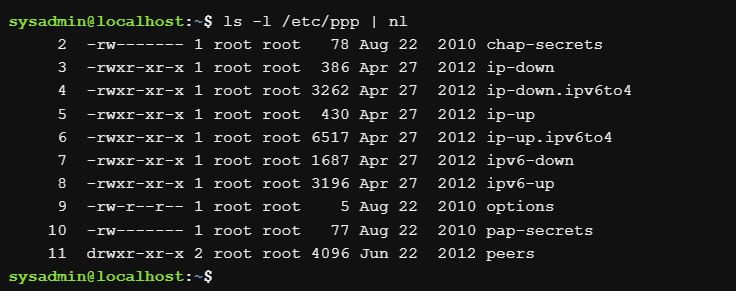
Instead of showing the output of the above command, putting the vertical bar next to the head command shows only the first ten lines:



The output of the ls command is passed to the head command by the shell instead of being printed to the screen. The head command takes this output (from the ls) as “input data” and then the output of the head is printed to the screen.

Multiple vertical bars can be used consecutively to join multiple commands. If three commands are joined with the vertical bar, the output of the first command is passed to the second command. The output of the second command is passed to the third command. The output of the third command is printed on the screen.

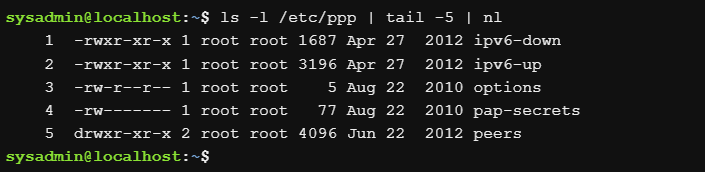
It is important to choose carefully the order in which the commands are joined with the vertical bar, since the third command will only see as input, the output of the second command. The following examples illustrate this situation using the nl command. In the first example, the nl command is used to number the lines of the output of a previous command:



In the example below, notice that the ls command is executed first and its output is sent to the nl command, listing all the lines of the output of the ls command. Next, the tail command is executed, displaying the last five lines of the nl command output:



Compare the above output with the following example:



Notice the different line numbers. Why does this happen?

In the second example, the output of the ls command is first sent to the tail command which “captures” only the last five lines of the output. The tail command sends those five lines to the nl command, which lists them from 1 to 5.

Vertical bars can be powerful, but it is important to consider how commands are linked with them to ensure that the desired output is displayed.

8.3 I/O Redirection

Input/Output (I/O) Redirection allows information to be passed from the command line to different sequences. Before discussing redirection, it is important to understand standard sequences.

8.3.1 STDIN

Standard STDIN input is information normally entered by the user via the keyboard. When a command sends a prompt to the shell expecting data, the shell provides the user with the ability to enter commands, which in turn, are sent to the command as STDIN.

8.3.2 STDOUT

Standard output or STDOUT is the normal output of commands. When a command runs correctly (without errors), the output it produces is called STDOUT. By default, STDOUT is displayed in the terminal window (screen) where the command is executed.

8.3.3 STDERR

Standard error or STDERR are error messages generated by commands. By default, STDERR is displayed in the terminal window (screen) where the command is executed.

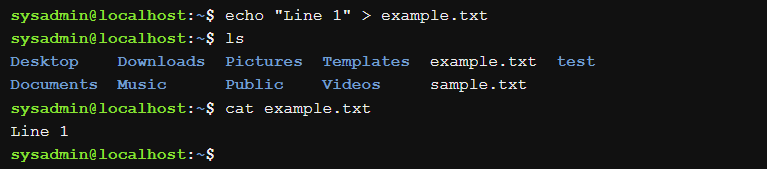
I/O redirection allows the user to redirect STDIN so that data comes from a file and STDOUT/STDERR output goes to a file. The redirection is accomplished by using the arrow characters: < and >.

8.3.4 Redirecting STDOUT

STDOUT can be directed to files. To begin, observe the output of the following command that will be displayed on the screen:

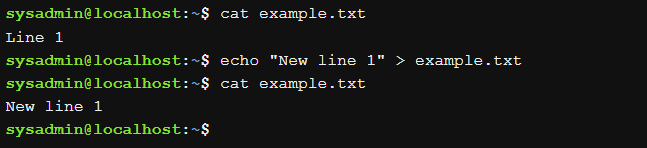


Using the > character, the output can be redirected to a file:



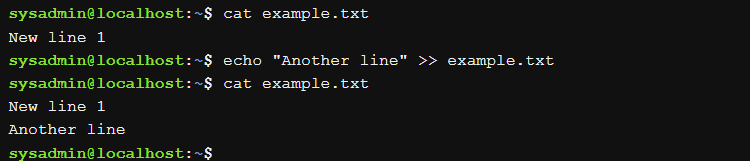
This command does not show any output, because the STDOUT was sent to the example.txt file instead of the screen. You can see the new file with the output of the ls command. The newly created file contains the output of the echo command when you view the contents of the file with the cat command.

It is important to note that the arrow alone overwrites any contents of an existing file:



The original contents of the file are gone and have been replaced by the output of the new echo command.

It is also possible to preserve the contents of an existing file by appending to it. Use the “double arrow” >> to append to a file instead of overwriting it:



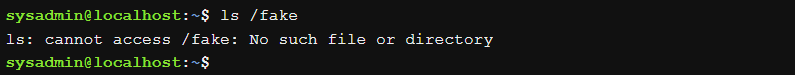
Instead of being overwritten, the command output of the recent echo command is appended to the bottom of the file.

8.3.5 Redirecting the STDERR

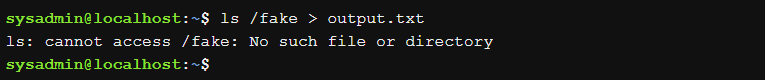
You can redirect the STDERR in a similar way to the STDOUT. STDOUT is also known as stream or channel #1. STDERR is assigned stream #2.

When using the arrows to redirect, stream #1 will be assumed as long as no other stream is specified. Therefore, sequence #2 must be specified when redirecting the STDERR.

To demonstrate the STDERR redirection, first observe the following command which will produce an error because the specified directory does not exist:

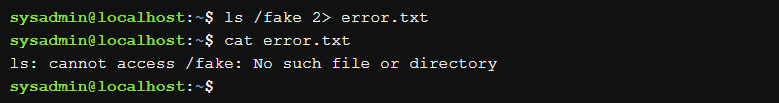


Note that there is nothing in the above example implying that the output is STDERR. The output is clearly an error message, but how would you know that it is sent to STDERR? An easy way to determine this is to redirect to STDOUT:



In the example above, the STDOUT was redirected to the output.txt file. Therefore, the output displayed cannot be STDOUT because it would have been left in the output.txt file. Since all command results go to STDOUT or STDERR, the output displayed must be STDERR.

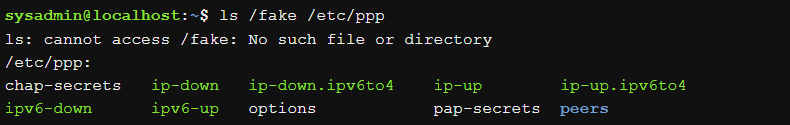
The STDERR of a command can be sent to a file:



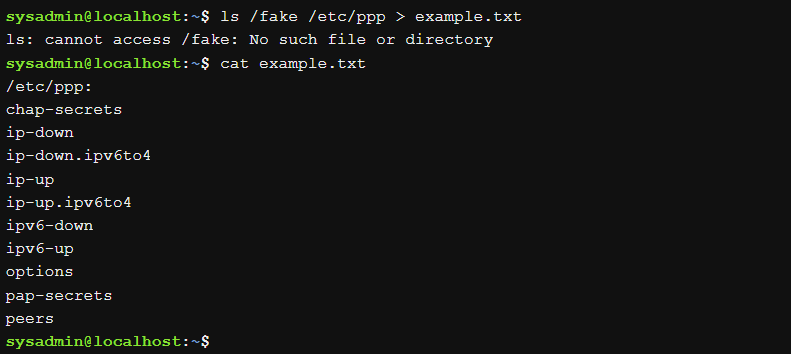
In the above command, 2> indicates that all error messages should be sent to the error.txt file.

8.3.6 Redirecting Multiple Sequences

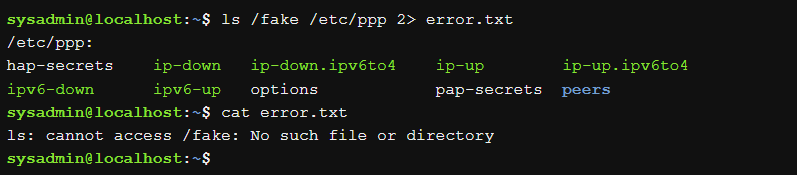
It is possible to address the STDOUT and STDERR output of one command at a time. The following command produces both STDOUT and STDERR output because one of the specified directories exists and the other does not:



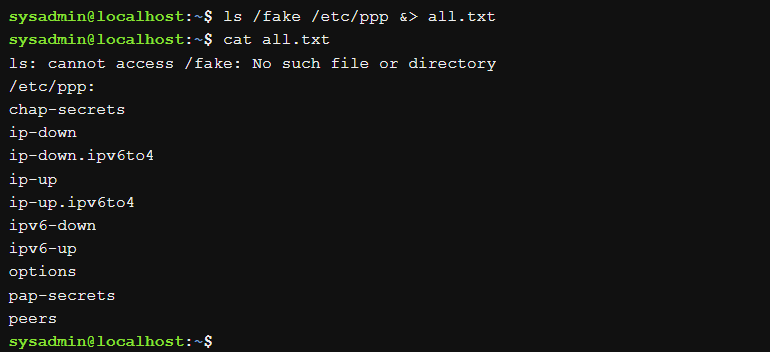
If only the STDOUT output is sent to a file, the STDERR will still be printed to the screen:



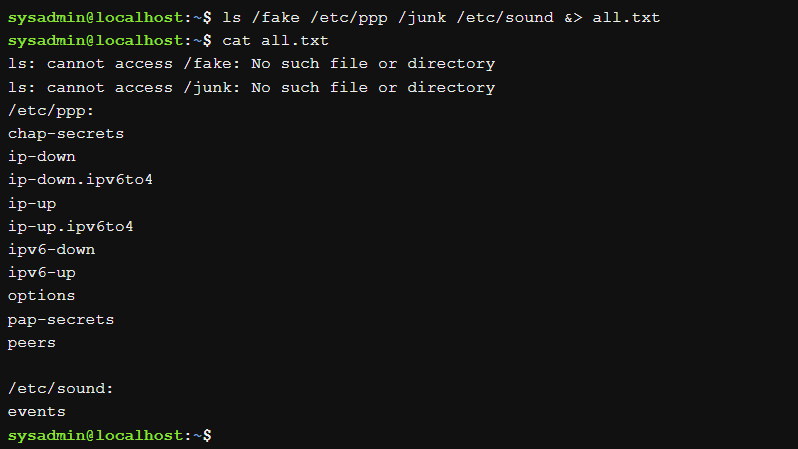
If only the STDERR output is sent to a file, the STDOUT will still be printed to the screen:



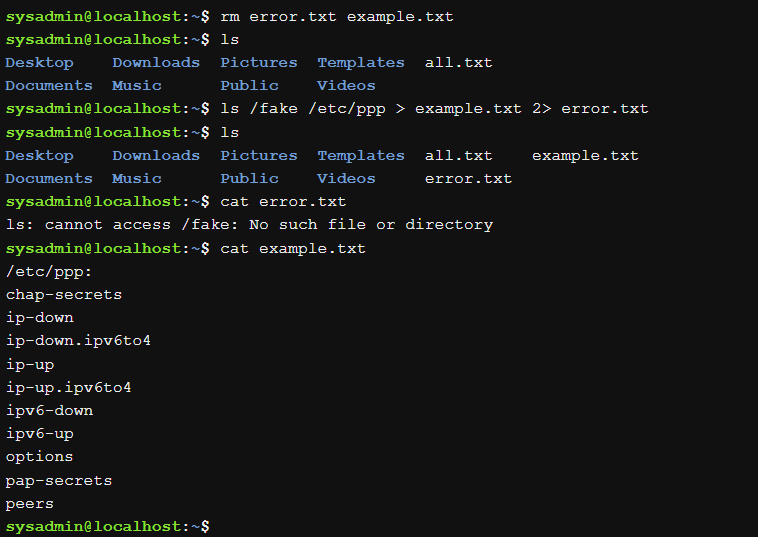
The STDOUT and STDERR outputs can be sent to a file by using &> a set of characters meaning “both 1> and 2>”:



Note that when using &>, the output appears in the file with all STDERR messages on top and all STDOUT messages below all STDERR messages:



If you do not want the STDERR and STDOUT outputs to go to the same file, you can redirect them to different files using > and 2>. For example:



The order in which the specified sequences come does not matter.

8.3.7 Redirect STDIN entry

The concept of redirecting the STDIN is difficult, as it is harder to understand why you would want to redirect the STDIN. With the STDOUT and STDERR outputs, the answer as to why is fairly easy: because sometimes you want to store the result in a file for future use.

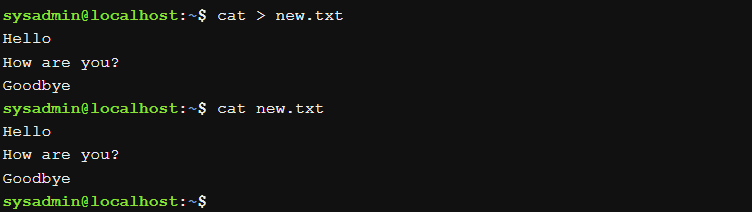
Most Linux users end up routinely redirecting STDOUT, occasionally STDERR, and STDIN.... well, very rarely. There are very few commands that require you to redirect STDIN because in the case of most commands, if you want to pass data from a file to a command, you can simply specify the file name as an argument to the command. Then, the command will search the file.

For some commands, if you do not specify a file name as an argument, they will fall back to using the STDIN output to get the data. For example, consider the following cat command:



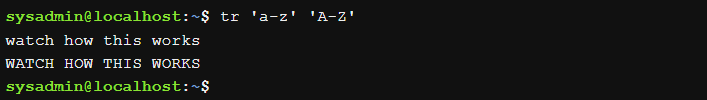
In the above example, the cat command did not receive the file name as an argument. Therefore, it requested the data to be displayed on the screen from the STDIN input. The user enters hello and then the cat command displays hello on the screen. Maybe this is useful for lonely people, but not really a good use of the cat command.

However, perhaps if the output of the cat command redirects to a file, then this method could be used to append data to an existing file or place text in a new file:



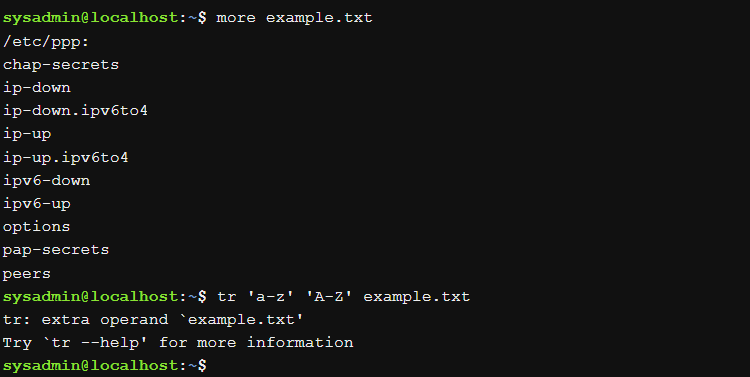
While the above example shows another advantage of redirecting STDOUT, it does not address why or how you can direct STDIN. To understand this, let's first consider a new command called tr. This command will take one set of characters and translate them into another set of characters.

For example, suppose you want to capitalize a command line. You can use the tr command as follows:

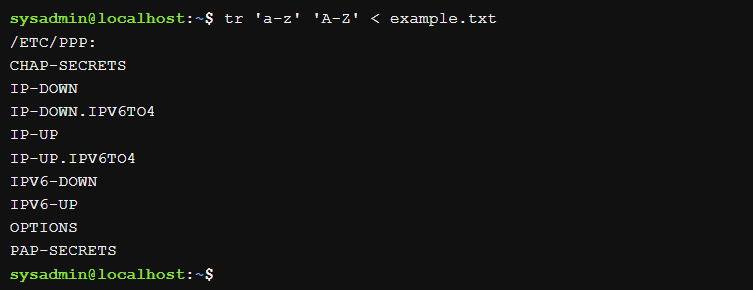


The tr command took the STDIN input from the keyboard (watch how this works) and converts all letters to lower case before sending the STDOUT output to the screen (WATCH HOW THIS WORKS).

It would seem that the tr command would serve more to perform the translation in a file, not the keyboard input. However, the tr command does not take filename arguments:

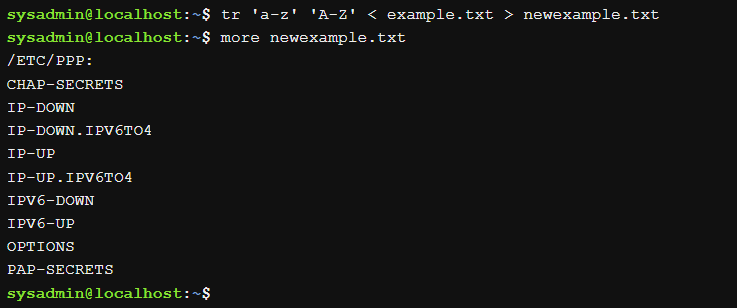


However, you can tell the shell to get the STDIN from a file instead of from the keyboard by using the < character:



This is quite rare because most commands accept filenames as arguments. However, for those that do not, this method could be used to have the shell read from the file instead of relying on the command having this capability.

One final note: In most cases you probably want to take the resulting output and place it in

another file: 

echo: This command displays on the terminal the text or variables you provide as arguments. For example, echo “Hello, world!” will print the phrase “Hello, world!” on terminal1.

cat: The cat command is used to display the contents of a file. In your example, cat mymessage will display the contents of the file named “mymessage”.

find: This command searches for files or directories in a specific location. For example, find /etc -name hosts will search for files named “hosts” inside the “/etc” directory. If you add 2> err.txt, it will redirect errors to a file named “err.txt”.

tr: The tr command is used to transform or replace characters in a text stream. For example, tr a-z A-Z will change all lowercase letters to uppercase. If you do tr A-Z a-z > myfile, a file named “myfile” will be created with the transformed text.

\*\*`tr a-z A-Z < myfile`\*\*: This command transforms the contents of the file “myfile” by converting all lowercase letters to uppercase.

\*\*`ls -l /etc | more`\*\*: The `ls` command lists the files and directories in the “/etc” directory. The `-l` option displays the details in long format, and the `|` (pipe) operator sends the output to the `more` command, which allows you to view the list in paginated form.

\*\*`cut -d: -f1 /etc/passwd`\*\*: The `cut` command extracts fields from a text file. In this case, the fields in the file “/etc/passwd” are separated using “:” as delimiter, and the first column (user names) is displayed.

\*\*`find ~ -name “\*bash\*”`\*\*: Search for files or directories in your home directory (~) that contain the

\*\*`find ~/Music -mmin -5`\*\*: Find files in the “~/Music” directory modified in the last 5 minutes.

\*\*`touch ~/Music/mysong`\*\*: Creates a file named “mysong” in the “~/Music” directory.

\*\*`find ~/Music -mmin -5`\*\*: Searches again for files in the “~/Music” directory modified in the last 5 minutes (including the newly created file).

\*\*`find /usr -size +2M`\*\*: Find files in the “/usr” directory that are larger than 2 megabytes.

\*\*`find /usr/share/bug -type d`\*\*: Find directories in “/usr/share/bug”.

\*\*`find /usr/share/bug -type d -ls`\*\*: Show details of directories found in “/usr/share/bug”.

\*\*`cat /etc/passwd`\*\*: Displays the contents of the file “/etc/passwd”.

\*\*`more /etc/passwd`\*\*: Allows you to view the contents of the “/etc/passwd” file in paginated form.word “bash” in their name.

\*\*`h`\*\*: Not a valid command. It may be a typo or a reference to something specific on your system.

\*\*`more h`\*\*: Attempts to display the contents of the “h” file in paginated form.

\*\*`<SPACE>`\*\*: This is not a valid command. It may be a blank space or a typo.

\*\*`less /etc/passwd`\*\*: Similar to `more`, displays the contents of the “/etc/passwd” file in paginated form, but with more navigation options.

\*\*`head /etc/passwd`\*\*: Displays the first lines of the “/etc/passwd” file.

\*\*`tail /etc/passwd`\*\*: Shows the last lines of the file “/etc/passwd”.

\*\*`head -2 /etc/passwd`\*\*: This command displays the first two lines of the “/etc/passwd” file. In this case, you will probably see information about system users.

\*\*`ls /etc | tail -5`\*\*: The `ls` command lists the files and directories in the “/etc” directory. Then, the `|` (pipe) operator sends the output to the `tail -5` command, which displays the last 5 lines of that list. This will give you a quick view of the last files and directories in “/etc”.

\*\*`head -n -20 /etc/passwd`\*\*: Displays all the lines of the file “/etc/passwd” except the last 20 lines.

\*\*`cd /etc`\*\*: Change the current directory to the “/etc” directory. You will now be working inside that directory.

\*\*`grep sshd passwd`\*\*: Search for the word “sshd” in the “/etc/passwd” file. This might be useful to find information related to the SSH service.

\*\*`frep root passwd`\*\*: There seems to be a typo here. It should be `grep` instead of `frep`. If you correct this, the command will look for the word `root` in the `/etc/passwd` file.

\*\*`grep root passwd`\*\*: Similar to the previous command, it looks for the word “root” in the “/etc/passwd” file.

\*\*`grep '^root' passwd`\*\*: Searches for lines starting with the word “root” in the file “/etc/passwd”. The `^` symbol indicates the start of the line.

\*\*`grep 'sync' passwd`\*\*: Search for the word “sync” in the file “/etc/passwd”. This could be related to some user or system process.

\*\*`grep 'sycn$' passwd`\*\*: There also seems to be a typo here. It should be ``grep 'sync$' passwd`. This command searches for lines ending with the word “sync” in the file “/etc/passwd”.

\*\*`grep '.y' passwd`\*\*: Searches for lines containing any character followed by the letter “y” in the file “/etc/passwd”.

\*\*`grep 'sshd|root|operator' passwd`\*\*: Searches for lines containing any of the words “sshd”, “root” or “operator” in the file “/etc/passwd”.

\*\*`grep -E '[0-9](3)' passwd`\*\*: Uses extended regular expressions (-E) to search for patterns. In this case, it looks for lines containing a digit followed by the digit “3” in the file “/etc/passwd”.