Searching For Files

Standard File Streams:

1. When commands are executed, by default there are three standard file streams (or descriptors) always open for use: standard input (standard in or **stdin**), standard output (standard out or **stdout**) and standard error (or **stderr**).

Name	Symbolic Name	Value	Example
standard input	**stdin**	0	keyboard
standard output	**stdout**	1	terminal
standard error	**stderr**	2	log file

2. Usually, **stdin** is your keyboard, and **stdout** and **stderr** are printed on your terminal.

3. **stderr** is often redirected to an error logging file, while **stdin** is supplied by directing input to come from a file or from the output of a previous command through a pipe.

4. **stdout** is also often redirected into a file.

5. Through the command shell, we can redirect the three standard file streams so that we can get input from either a file or another command, instead of from our keyboard, and we can write output and errors to files or use them to provide input for subsequent commands.

6. For example, if we have a program called **do_something** that reads from **stdin** and writes to **stdout** and **stderr**, we can change its input source by using the less-than sign (<) followed by the name of the file to be consumed for input data:

do_something < input-file

7. If you want to send the output to a file, use the greater-than sign (>) as in:

do_something > output-file

8. Because **stderr** is not the same as **stdout**, error messages will still be seen on the terminal windows in the above example.

9. If you want to redirect **stderr** to a separate file, you use **stderr's** file descriptor number (2), the greater-than sign (>), followed by the name of the file you want to hold everything the running command writes to stderr:

do_something 2> error-file

NOTE: By the same logic, **do_something 1> output-file** is the same as **do_something > output-file**.

10. A special shorthand notation can send anything written to file descriptor 2 (**stderr**) to the same place as file descriptor 1 (**stdout**): **2>&1**.

do_something > all-output-file 2>&1

11. bash permits an easier syntax for the above:

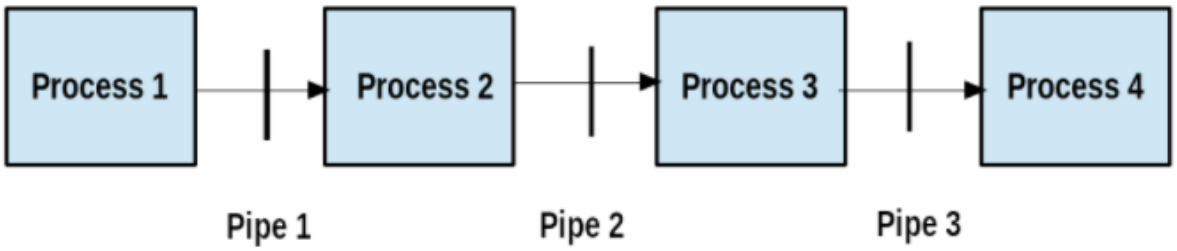
do_something >& all-output-file

Pipes:

1.You can pipe the output of one command or program into another as its input.

2. In order to do this, we use the vertical-bar, pipe symbol (|), between commands as in:

command1 | command2 | command3



Pipeline

Locate:

1.The locate utility program performs a search taking advantage of a previously constructed database of files and directories on your system, matching all entries that contain a specified character string.

2. To get a shorter (and possibly more relevant) list, we can use the grep program as a filter. grep will print only the lines that contain one or more specified strings, as in:

locate zip | grep bin

3. In the above command we are listing all the files and directories with both zip and bin in their name.

4. locate utilizes a database created by a related utility, **updatedb**

5. Most Linux systems run this automatically once a day.

6. However, you can update it at any time by just running **updatedb** from the command line as the root user (i.e **sudo updatedb**).

Wildcards and Matching File Names:

1.To search for files using the **?** wildcard, replace each unknown character with **?**. For example, if you know only the first two letters are **'ba'** of a three-letter filename with an extension of **.out,** type **ls ba?.out.**

2. To search for files using the * wildcard, replace the unknown string with *. For example, if you remember only that the extension was .out, type ls ***.out**.

Wildcard	Result
?	Matches any single character
*	Matches any string of characters
[set]	Matches any character in the set of characters, for example [adf] will match any occurrence of a, d, or f
[!set]	Matches any character not in the set of characters

The find Program:

1.find is an extremely useful and often-used utility program in the daily life of a Linux system administrator.

2. It recurses down the filesystem tree from any particular directory (or set of directories) and locates files that match specified conditions.

3. The default pathname is always the present working directory.

4. When no arguments are given, find lists all files in the current directory and all of its subdirectories.

5. Commonly used options to shorten the list include -name (only list files with a certain pattern in their name), **-iname** (also ignore the case of file names), and **-type** (which will restrict the results to files of a certain specified type, such as d for directory, **l** for symbolic link, or **f** for a regular file, etc.).

6. Searching for files and directories named gcc:

find /usr -name gcc

7. Searching only for directories named gcc:

find /usr -type d -name gcc

8. Searching only for regular files named gcc:

find /usr -type f -name gcc

9. Another good use of find is being able to run commands on the files that match your search criteria. The **-exec** option is used for this purpose.

10. To find and remove all files that end with **.swp**:

find -name "*.swp" -exec rm {} ';'

11. The **{}** (squiggly brackets) is a placeholder that will be filled with all the file names that result from the find expression, and the preceding command will be run on each one individually.

12. Please note that you have to end the command with either **';'** (including the single-quotes) or **"\;".** Both forms are fine.

13. One can also use the **-ok** option, which behaves the same as **-exec**, except that find will prompt you for permission before executing the command. This makes it a good way to test your results before blindly executing any potentially dangerous commands.

14. It is sometimes the case that you wish to find files according to attributes, such as when they were created, last used, etc., or based on their size. It is easy to perform such searches.

15. To find files based on time:

find / -ctime 3

16. Here, **-ctime** is when the **inode** metadata (i.e. file ownership, permissions, etc.) last changed; it is often, but not necessarily, when the file was first created.

17. You can also search for accessed/last read (**-atime**) or modified/last written (**-mtime**) times.

18. The number is the number of days and can be expressed as either a number (**n**) that means exactly that value, **+n**, which means greater than that number, or **-n**, which means less than that number.

19. There are similar options for times in minutes (as in **-cmin, -amin**, and **-mmin**).

20. To find files based on sizes:

find / -size 0

21. Note the size here is in 512-byte blocks, by default; you can also specify **bytes (c), kilobytes (k), megabytes (M), gigabytes (G),** etc. As with the time numbers above, file sizes can also be exact **numbers (n), +n or -n.**

22. For example, to find files greater than 10 MB in size and running a command on those files:

find / -size +10M -exec command {} ';'