**UNIX Piping**

Imagine a situation where you have to write a large monolith program with tight space constraints. Also, you work in a non-UNIX environment. Imagine the difficulty that you will have. This problem was solved with the introduction of UNIX and UNIX-Pipes. With the introduction of pipes, it became possible to pipe the standard output of one program to the standard input of another program. We can break the larger program in smaller sub-programs and then we can give the output of a program as an input to another program. Also, piping operates in such a manner that the **output never has to be stored** before being used by the next program. Before we dig into the concept of pipe command, it is necessary to know about the basic concepts of standard streams.

**Standard Stream**

First of all, what is a stream? Stream is basically a sequence of data available over time. Every Unix program has three streams opened for it when it starts up, one for input, one for output and one for printing diagnostic or error messages. These are typically attached to the user's terminal. So, What do you mean by it? If any program wants to read input from the user, by default it will read whatever the user writes to the terminal by keyboard and after processing the data it will print it to the user’s terminal by default, even the error message is printed to the user’s terminal by default.

Basically the three types of standard streams are:

1) Standard input(**stdin**)

2) Standard Output(**stdout**)

3) Standard Error(**stderr**)

So when you are writing a complex program, simply writing the output to your console or reading input always from the user’s terminal might not be your goal, here come into play, handling input and output. So, most of the commands in UNIX have the three standard streams flowing, for which the default location is the user's terminal. Sometimes we want our output to be saved into a different file, it is done by redirection using the ">” or “>>” symbol. Sometimes you want it that,the output of one program to be input for the other,it can be done by “|” or “|&” symbol about which we will be mentioning further. So, now basically you have the basic idea of what the Standard streams are? Now let’s dig into the pipe command.

**PIPE COMMAND (“|”):**

So, What the pipe command basically do? What it does is it connects the outgoing stream of one command to the incoming stream of other, this happens when you separate the different command using the “|” symbol. In simpler terms, the output of one command is input for the other and you can create a pipeline of command separated by “|”.

But wait! We have to understand the concept of Redirection before we get into piping. *Redirection can be defined as changing the way from where commands read input to where commands send output.*

The technical definition of pipe is:

*A pipe is a form of redirection that is used in linux and other unix-like programming languages to send the output of one program to another program for further processing.*

So a question might arise. What happens if some command produces a message in the error stream, does it flow to the stdin stream of others, the answer is NO. It prints out in the console. But now you want your error message to also flow through the pipe, then you can use “|&” symbol.

Now the outgoing stream of first command will not be shown in the console, rather the filtered data after processing through the command following the pipe will be displayed at the console unless it is redirected to some other file .

*REMEMBER THE FACT:* Pipes are unidirectional **i.e data flows from left to right through the pipeline.**

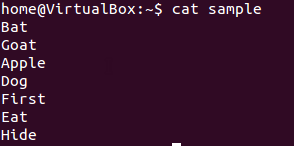
**Syntax :**

command\_1 | command\_2 | command\_3 | .... | command\_N

Pipelines allow commands/programs/processes to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen.

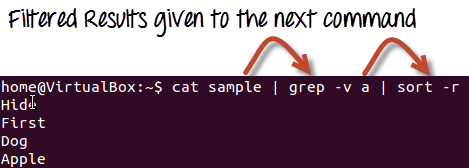
So,that was basically what a pipe is, what it does, so now let's look at some examples.

**Example (1):**



So now the cat command displayed it output directly at the terminal(the default stdout).Now we want to search the line which does not contain the character ‘a’.So we will use the grep command,which needs a stdin stream and we want the output of cat command to be input for this and print the result in a sorted order.So,here we will be using pipe.

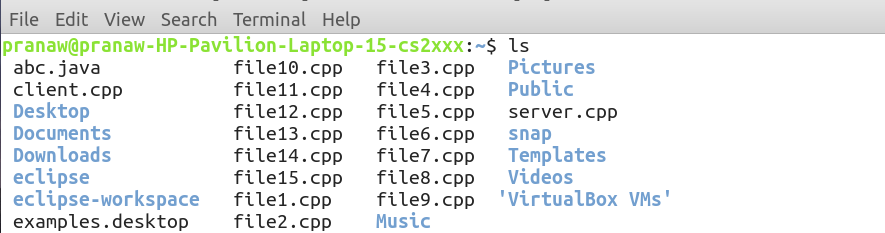
$ cat sample| grep -v a | sort -r



**Example (2):**

In this example we will see a very basic use of piping and will observe how powerful the tool piping is. In our home directory there are many files which we can observe using the following command

$ ls



Now, suppose we have to see only cpp files. Then one way we can use is as follows:

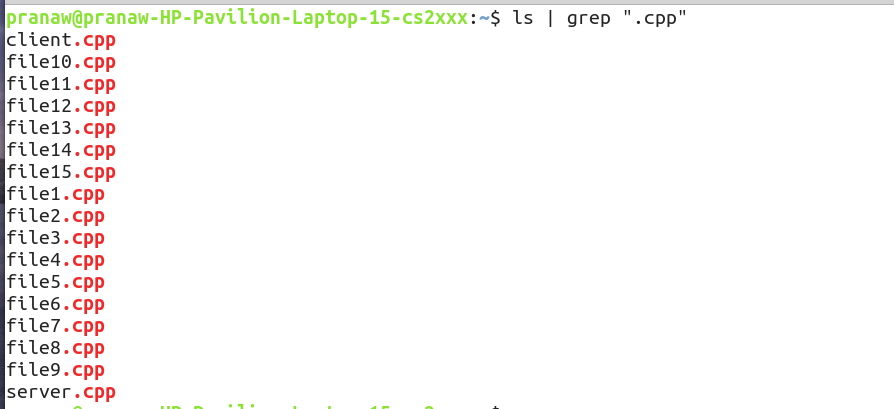
$ ls > temp

$ grep “.cpp” temp

$ rm temp

Which we can also achieve just by using following command:

$ ls | grep “.cpp”



All that thing we can do just in one step. A more amazing fact is that our system does not create any temporary file(s) while performing the task.

Above was implementation of piping between two commands. Let’s dive into piping with more commands.

Suppose we need to find the total number of files having “.cpp” extension. Below is the command required to do that.

$ wc -l

But how we will use this command using pipelines. Simple pipe the output of the “grep” command with “wc” command. We will run the following command.

$ ls -l | grep “.cpp” | wc -l



See the output! Works well :)

**Example (3) :**

Now let's show you how powerful pipes can be and can make your life way easier.

Let's say we have output from a command (say grep) as such:

Part 1: 1f68

Part 2: 0cb3

Part 3: 26bc

Part 4: 656f

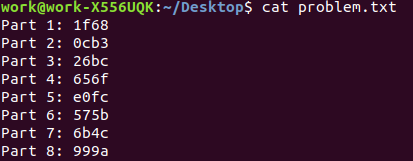
Part 5: e0fc

Part 6: 575b

Part 7: 6b4c

Part 8: 999a

Our goal is to join all the parts. It's do-able, yep simply we can copy and paste. But let's see how we can do it with the help of pipes. As of why? Well, if it would have been 800 instead of 8 as well as the goal is to look geeky.



So, we are just going to club together a few basic linux commands and do it.

cat problem.txt | rev | cut -d " " -f1 | rev | tr -d "\n"



To see the magic I suggest you try the same, and add the pipes one by one as to what each command does. For greater details you can always check their ‘man’ pages.

Btw, for wanderers, in the above example the hexadecimal just says ‘acmiitr’.

**Works of different Commands used here :**

ls -> It is a command in UNIX to list files and directories. -l is an option given to ls command so that it gives output in long format.

grep -> It is a command-line utility for searching plain-text data sets for lines that match a regular expression.

rev -> It reverses the order of the input. Like (123 becomes 321)

cat -> cat (concatenate) reads data from the file and gives their content as output. It helps us to create, view, concatenate files.