Department of
Computer Science and Engineering

Title: Linux/Unix Commands- II

Operating System Lab

CSE 310

Green University of Bangladesh

1 Objective(s)

- To gather knowledge of basic Linux commands.

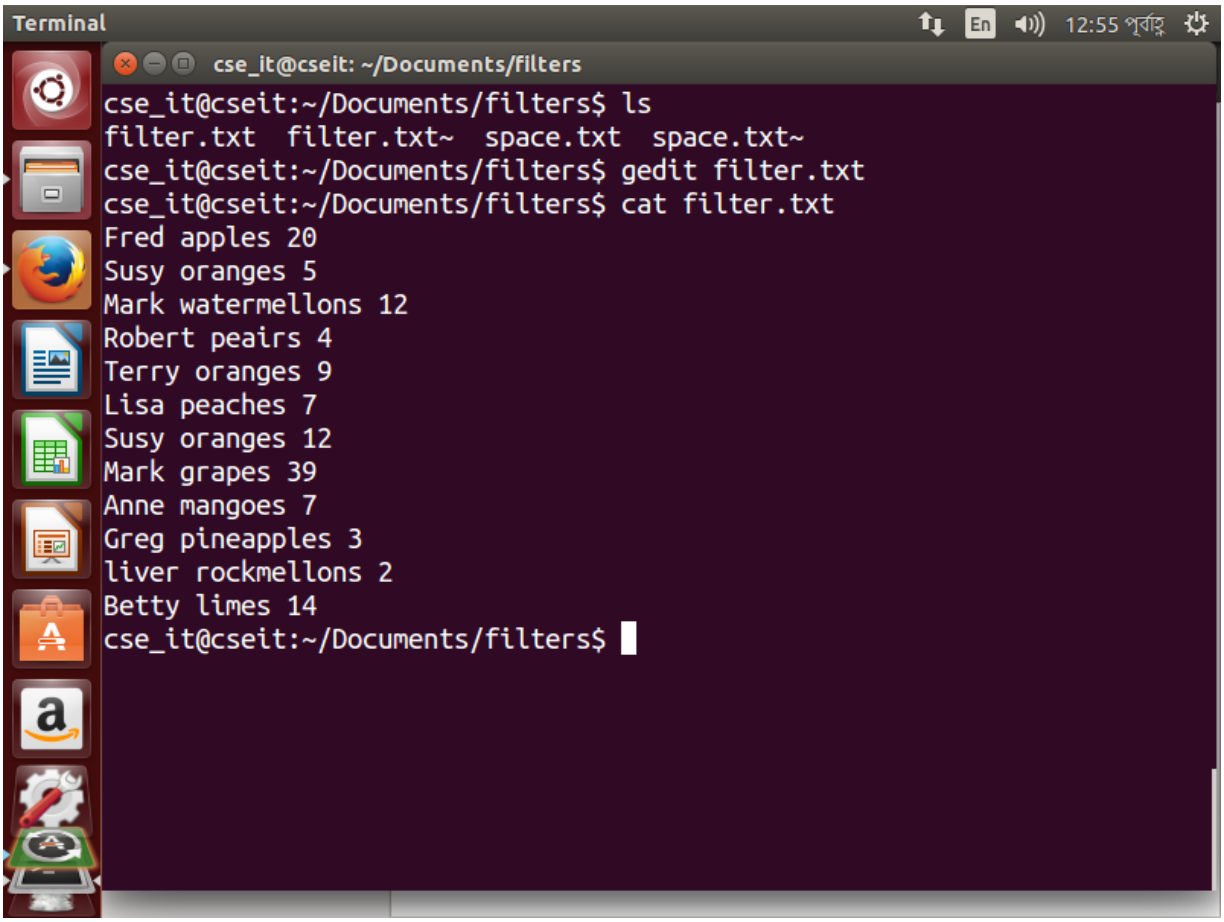
2 Linux

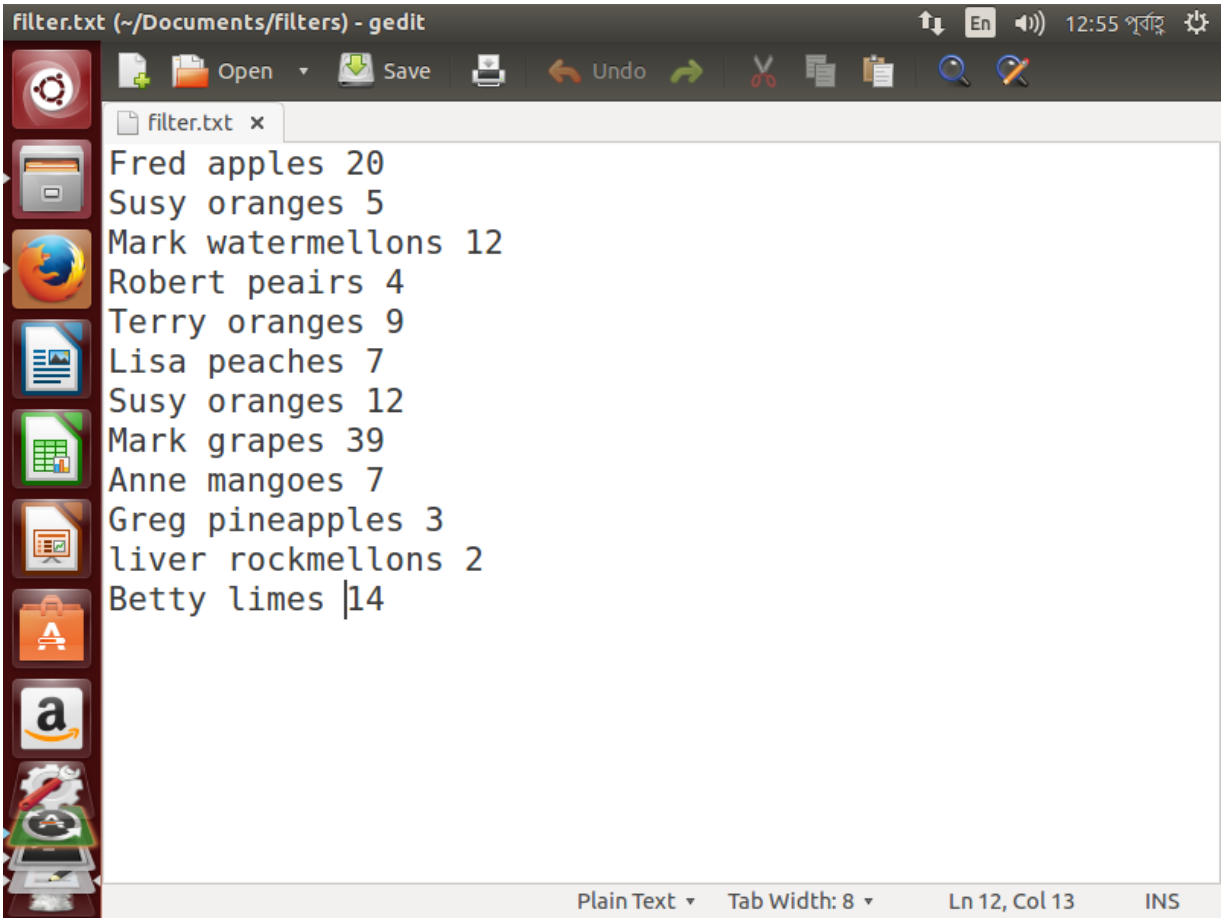
Linux has a graphical user interface and it works pretty much like the GUI's on other systems that you are familiar with such as Windows and OSX. This tutorial won't focus on these as I reckon you can probably figure that part out by yourself. This tutorial will focus instead on the command line (also known as a terminal) running Bash. Some Linux tutorial:

- **Filters** - An introduction to various commands that allow us to mangle data in interesting and useful ways.
- **Grep and Regular Expressions** - Master a powerful pattern matching language that is useful for analysing and processing data.
- **Piping and Redirection** - Join commands together in powerful combinations.
- **Process Management** - See what is currently running on your Linux system and what state the system is in, learn how to kill programs that have hung and put jobs in the background.

2.1 Filters!

A filter, in the context of the Linux command line, is a program that accepts textual data and then transforms it in a particular way. Filters are a way to take raw data, either produced by another program, or stored in a file, and manipulate it to be displayed in a way more suited to what we are after.





2.1.1 Head

Head is a program that prints the first so many lines of it's input. By default it will print the first 10 lines but we may modify this with a command line argument.

```
1 head [-number of lines to print] [path]
```

2.1.2 Tail

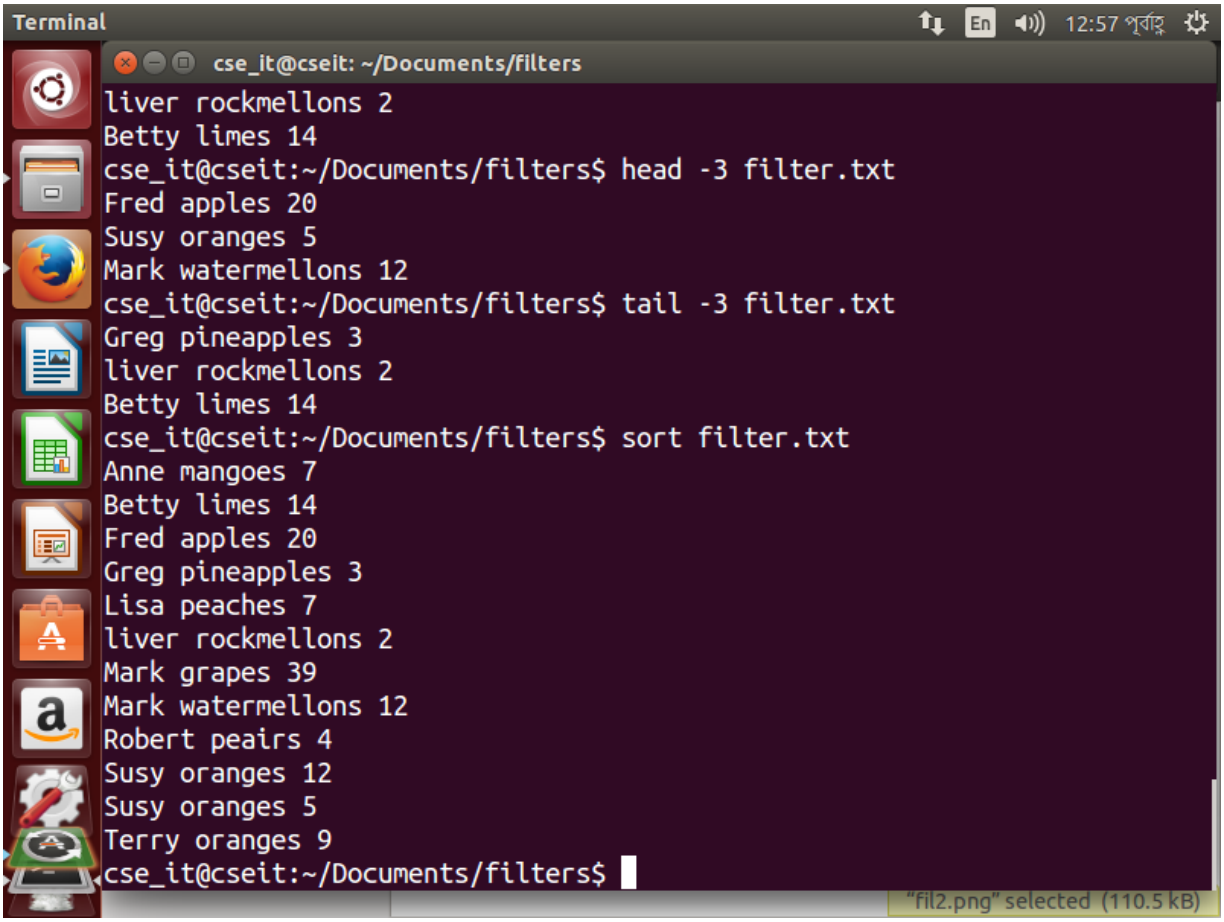
Tail is the opposite of head. Tail is a program that prints the last so many lines of it's input. By default it will print the last 10 lines but we may modify this with a command line argument.

```
1 tail [-number of lines to print] [path]
```

2.1.3 Sort

Sort will sort it's input, nice and simple. By default it will sort alphabetically but there are many options available to modify the sorting mechanism. Be sure to check out the manu page to see everything it may do.

```
1 sort [-options] [path]
```



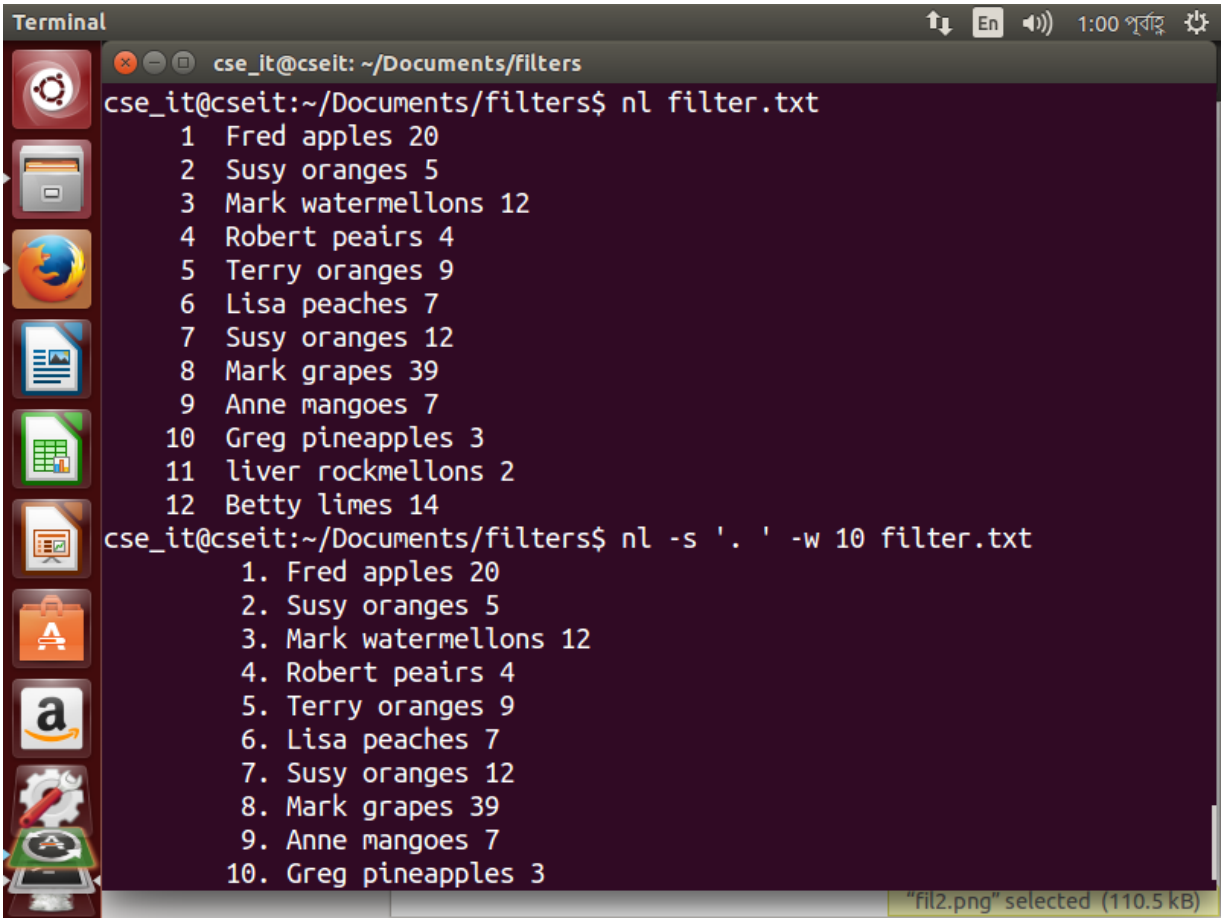
2.1.4 nl

nl stands for number lines and it does just that.

```
1 nl [-options] [path]
```

The basic formatting is ok but sometimes you are after something a little different. With a few command line options, nl is happy to oblige.

In the below example we have used 2 command line options. The first one -sspecifies what should be printed after the number while the second one -wspecifies how much padding to put before the numbers.



2.2 wc and cut

wc stands for word count and it does just that (as well as characters and lines. By default it will give a count of all 3 but using command line options we may limit it to just what we are after.

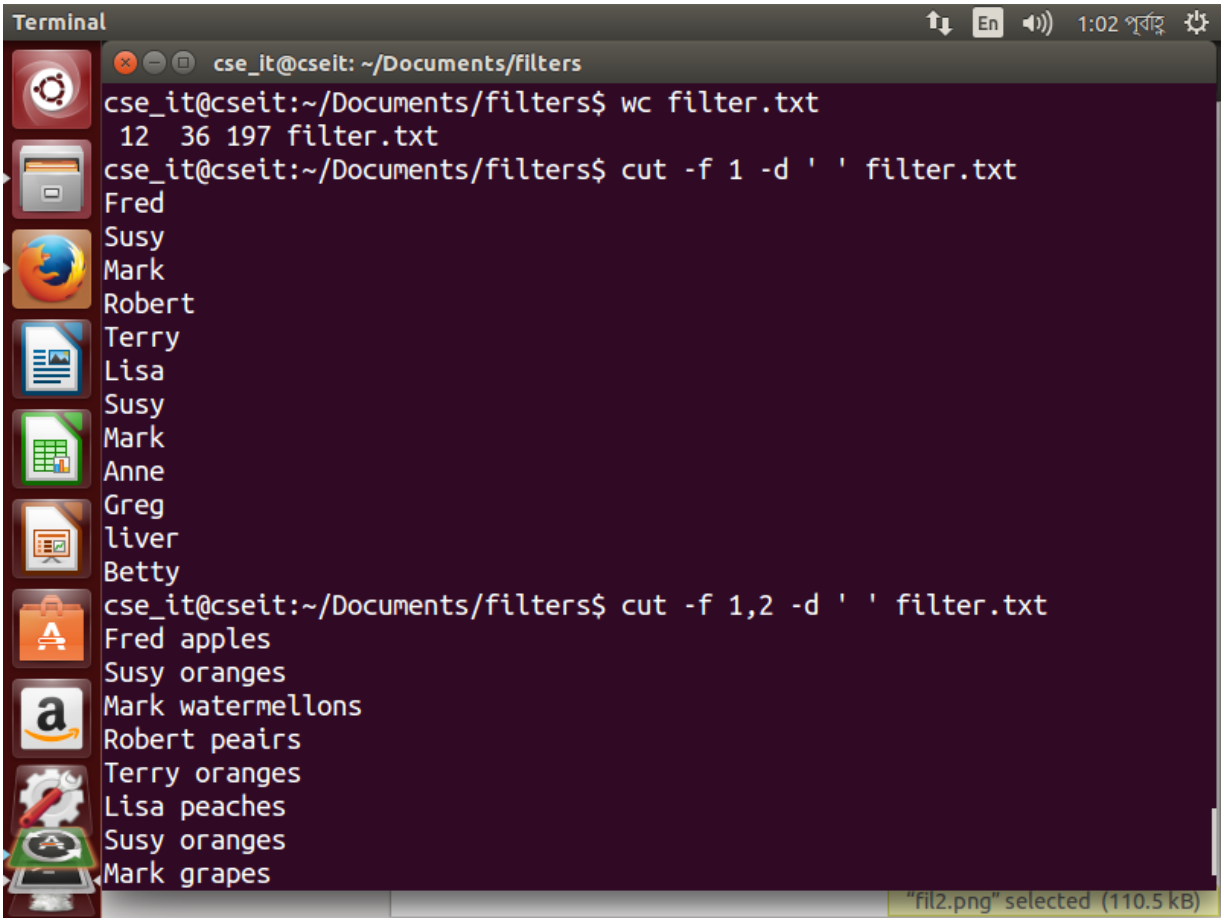
```
1 wc [-options] [path]
```

cut is a nice little program to use if your content is separated into fields (columns) and you only want certain fields.

```
1 cut [-options] [path]
```

In our sample file we have our data in 3 columns, the first is a name, the second is a fruit and the third an amount. Let's say we only wanted the first column.

cut defaults to using the TAB character as a separator to identify fields. In our file we have used a single space instead so we need to tell cut to use that instead. The separator character may be anything you like, for instance in a CSV file the separator is typically a comma (,). This is what the -d option does (we include the space within single quotes so it knows this is part of the argument). The -f option allows us to specify which field or fields we would like. If we wanted 2 or more fields then we separate them with a comma as below.



2.2.1 sed

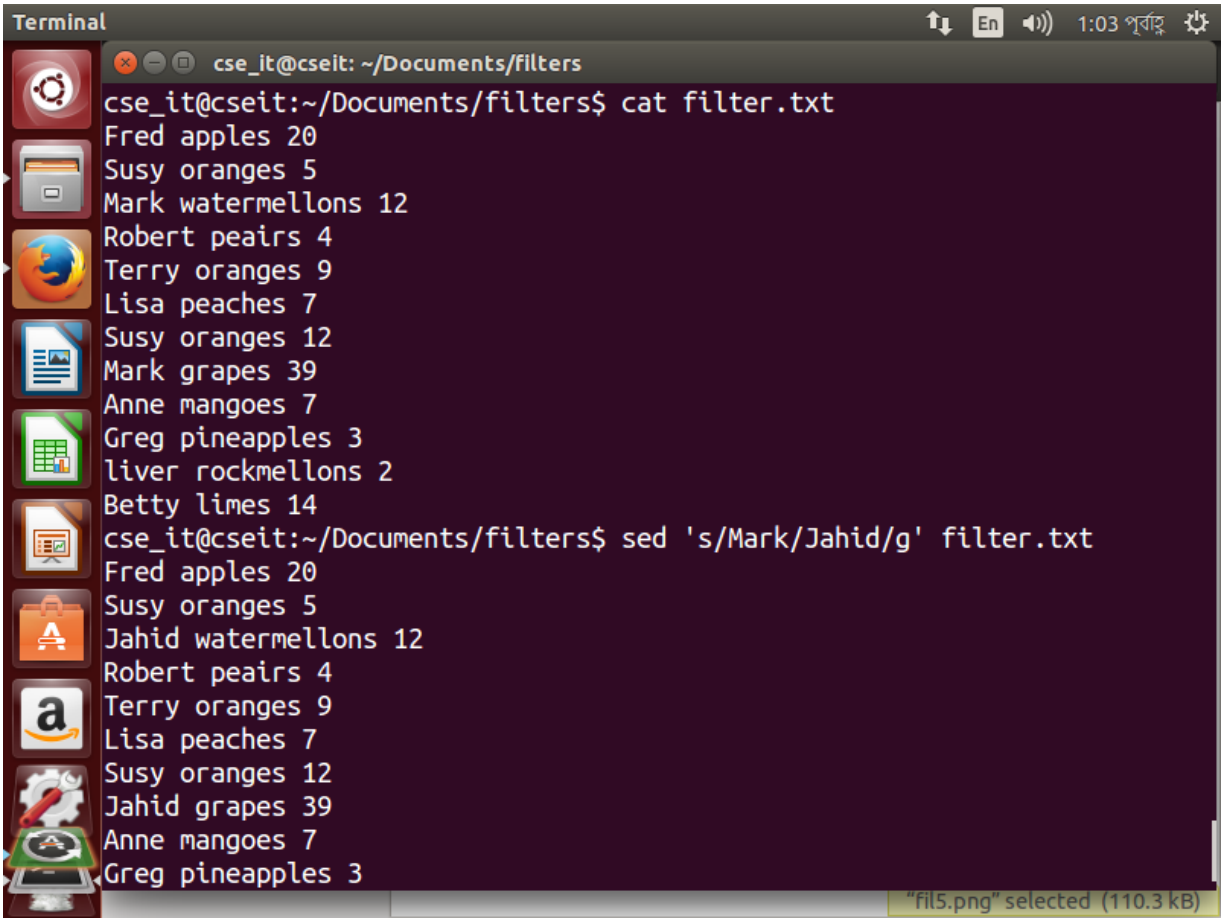
sed stands for Stream Editor and it effectively allows us to do a search and replace on our data. It is quite a powerful command but we will use it here in it's basic format.

```
1 sed <expression> [path]
```

A basic expression is of the following format:

- s/search/replace/g

The initial s stands for substitute and specifies the action to perform (there are others but for now we'll keep it simple). Then between the first and second slashes (/) we place what it is we are searching for. Then between the second and third slashes, what it is we wish to replace it with. The g at the end stands for global and is optional. If we omit it then it will only replace the first instance of search on each line. With the g option we will replace every instance of search that is on each line. Let's see an example. Say we ran out of oranges and wanted to instead give those people bananas.



2.3 Grep and Regular Expressions!

Regular expressions are similar to the wildcards. They allow us to create a pattern. They are a bit more powerful however. Re's are typically used to identify and manipulate specific pieces of data. eg. we may wish to identify every line which contains an email address or a url in a set of data.

The characters used in regular expressions are the same as those used in wildcards. Their behaviour is slightly different however. A common mistake is to forget this and get their functions mixed up.

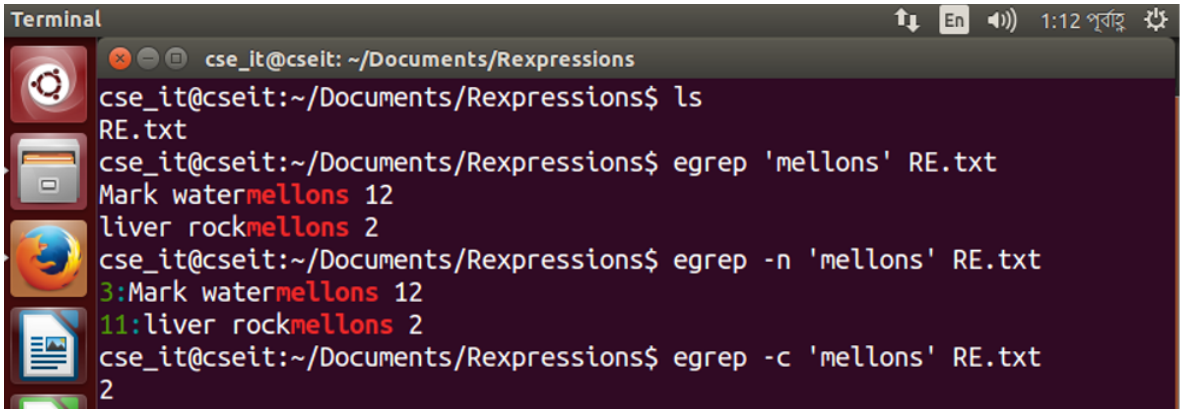
2.3.1 eGrep

egrep is a program which will search a given set of data and print every line which contains a given pattern. It is an extension of a program called grep. It's name is odd but based upon a command which did a similar function, in a text editor called ed. It has many command line options which modify it's behaviour so it's worth checking out it's man page. ie the -v option tells grep to instead print every line which does not match the pattern.

```
1 egrep [command line options] <pattern> [path]
```

In the examples below we will use a similar sample file as in the last section. It is included below as a reference.

- Let's say we wished to identify every line which contained the string mellons
- Sometimes we want to know not only which lines matched but their line number as well.
- Or maybe we are not interested in seeing the matched lines but wish to know how many lines did match.



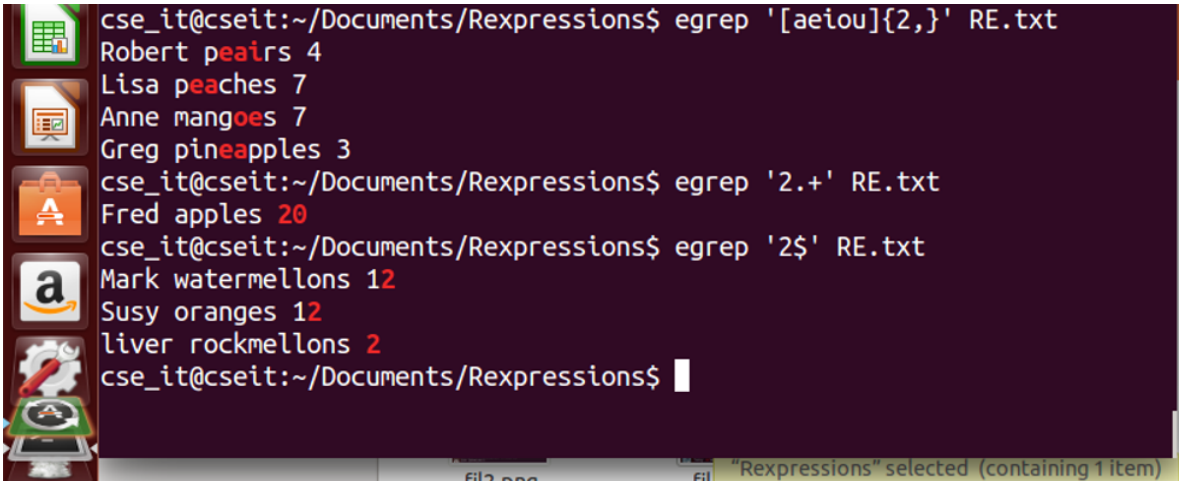
2.3.2 Regular Expression Overview

I will outline the basic building blocks of re's below then follow on with a set of examples to demonstrate their usage.

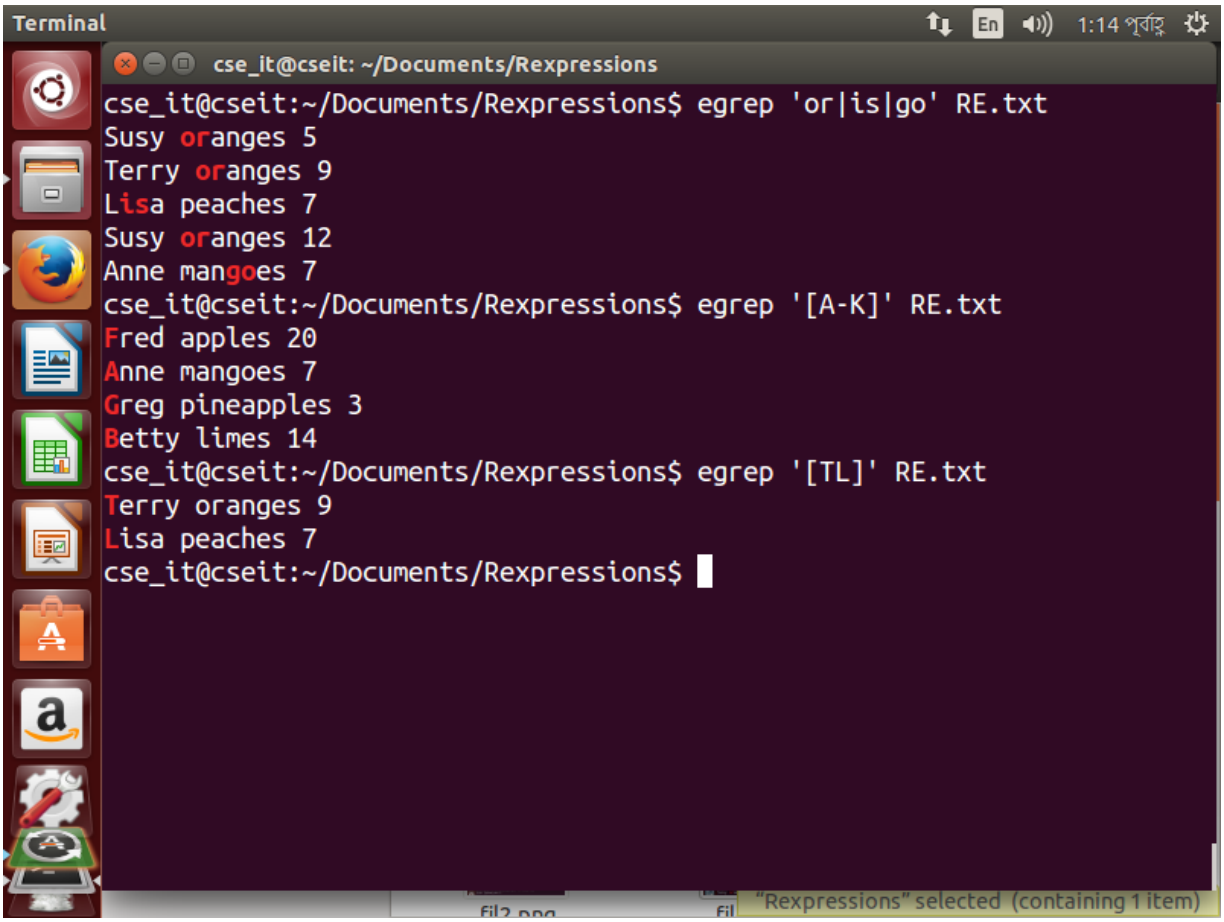
- . (dot) - a single character.
- ? - the preceding character matches 0 or 1 times only.
- * - the preceding character matches 0 or more times.
- + - the preceding character matches 1 or more times.
- n - the preceding character matches exactly n times.
- n,m - the preceding character matches at least n times and not more than m times.
- $[agd]$ - the character is one of those included within the square brackets.
- $[gd]$ - the character is not one of those included within the square brackets.
- $[c - f]$ - the dash within the square brackets operates as a range. In this case it means either the letters c, d, e or f.
- () - allows us to group several characters to behave as one.
- | (pipe symbol) - the logical OR operation.
- ^- matches the beginning of the line.
- $ - matches the end of the line.

Some Examples:

- Let's say we wish to identify any line with two or more vowels in a row. In the example below the multiplier 2, applies to the preceding item which is the range.
- How about any line with a 2 on it which is not the end of the line. In this example the multiplier + applies to the. which is any character.
- The number 2 as the last character on the line.



- And now each line which contains either 'is' or 'go' or 'or'.
- Maybe we wish to see orders for everyone who's name begins with A - K.
- Moreover, we wish to see orders for everyone who's name begins with T and L.

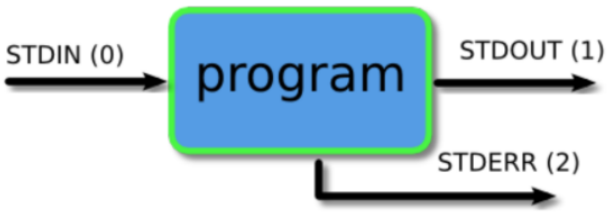


2.4 Piping and Redirection!

Every program we run on the command line automatically has three data streams connected to it.

- **STDIN (0)** - Standard input (data fed into the program)
- **STDOUT (1)** - Standard output (data printed by the program, defaults to the terminal)

- **STDERR (2)** - Standard error (for error messages, also defaults to the terminal)

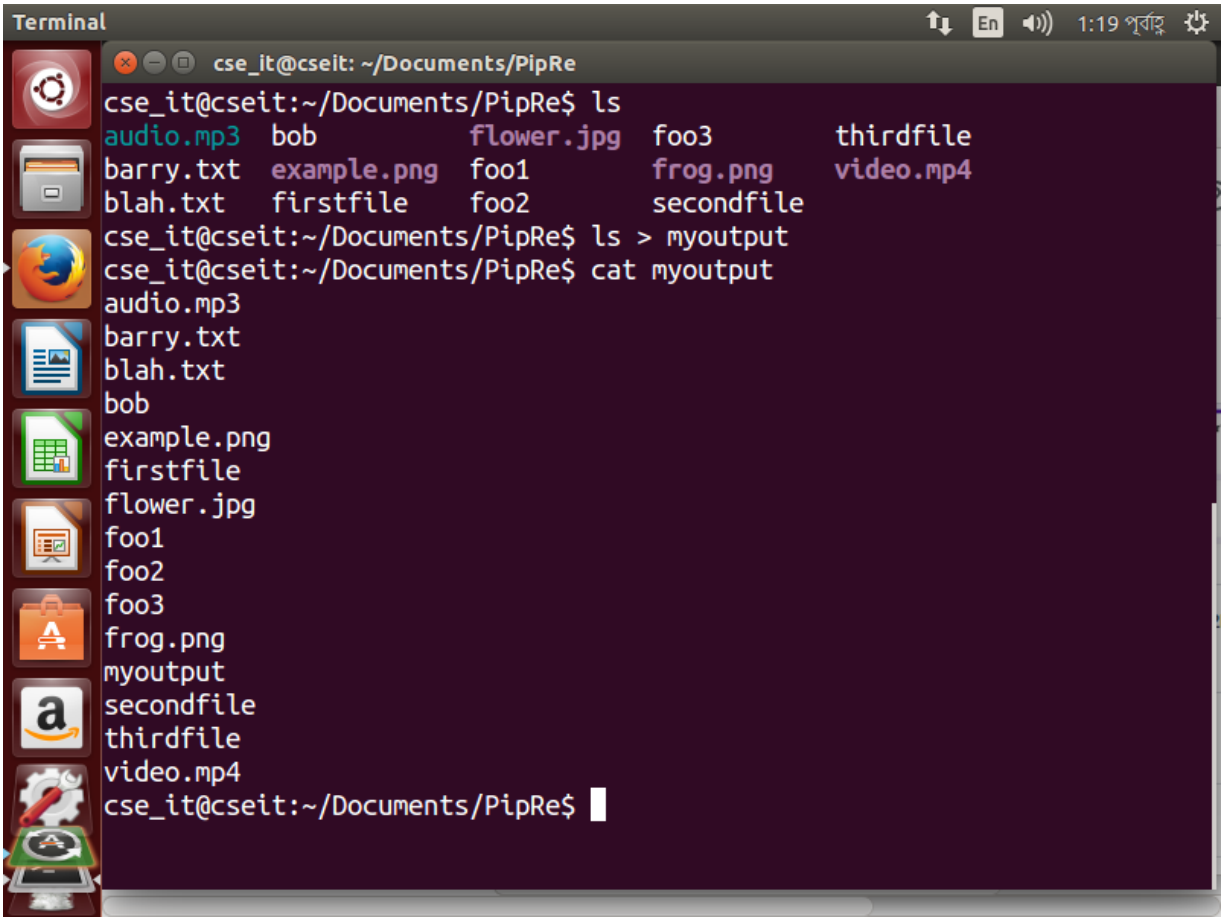


Piping and redirection is the means by which we may connect these streams between programs and files to direct data in interesting and useful ways.

We'll demonstrate piping and redirection below with several examples but these mechanisms will work with every programm on the command line, not just the ones we have used in the examples.

2.4.1 Redirecting to a File

Normally, we will get our output on the screen, which is convenient most of the time, but sometimes we may wish to save it into a file to keep as a record, feed into another system, or send to someone else. The greater than operator (>) indicates to the command line that we wish the programs output (or whatever it sends to STDOUT) to be saved in a file instead of printed to the screen. Let's see an example.



2.4.2 Saving to an Existing File

If we redirect to a file which does not exist, it will be created automatically for us. If we save into a file which already exists, however, then it's contents will be cleared, then the new output saved to it.

```
  cat myoutput
  barry.txt
  bob
  example.png
  firstfile
  foo1
  myoutput
  video.mpeg
  wc -l barry.txt > myoutput
  cat myoutput
  7 barry.txt
```

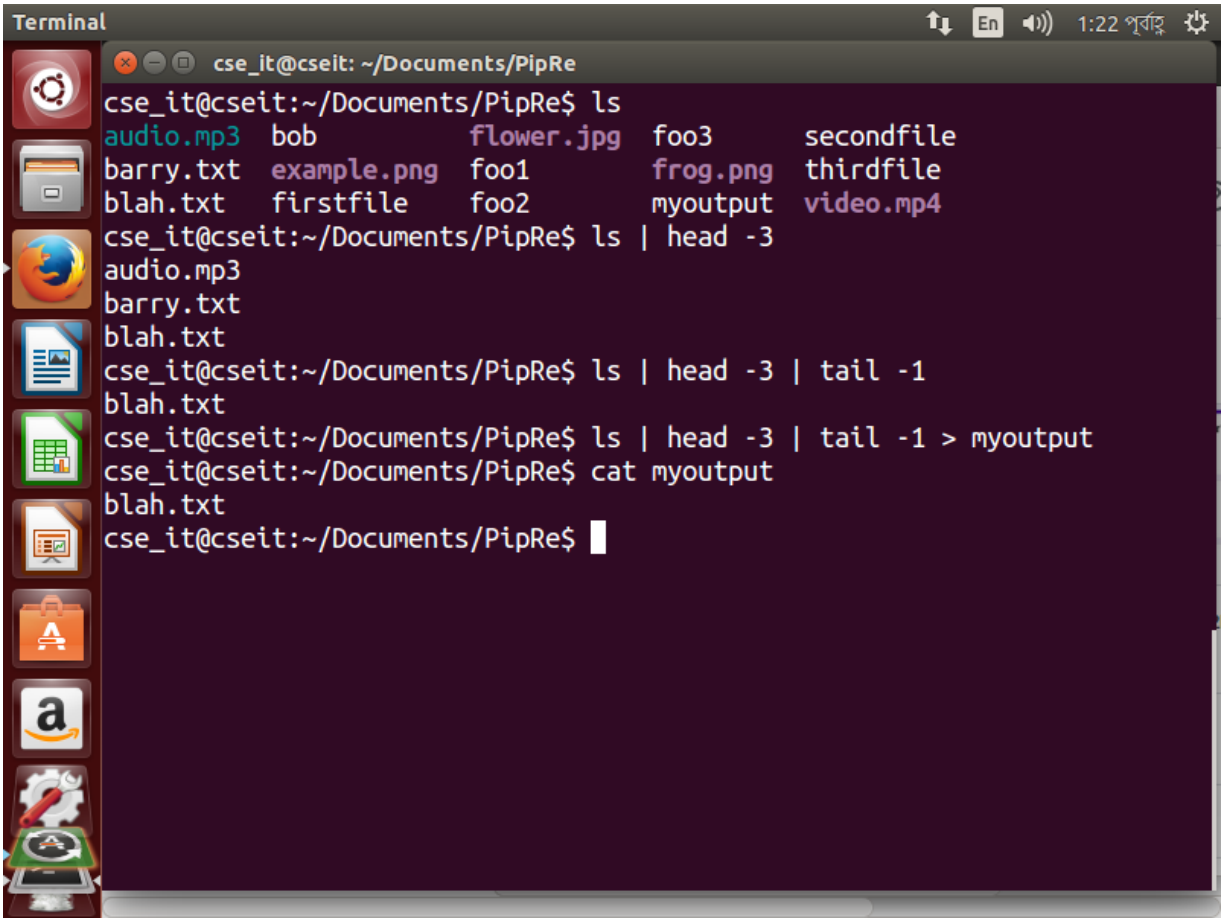
We can instead get the new data to be appended to the file by using the double greater than **operator (»)**.

```
cat myoutput
7 barry.txt
ls >> myoutput
cat myoutput
7 barry.txt
barry.txt
bob
example.png
firstfile
foo1
myoutput
video.mpeg
```

2.4.3 Piping

Now we'll take a look at a mechanism for sending data from one program to another. It's called piping and the operator we use is (|) (found above the backslash (\ ) key on most keyboards). What this operator does is feed the output from the program on the left as input to the program on the right. In the example below we will list only the first 3 files in the directory.

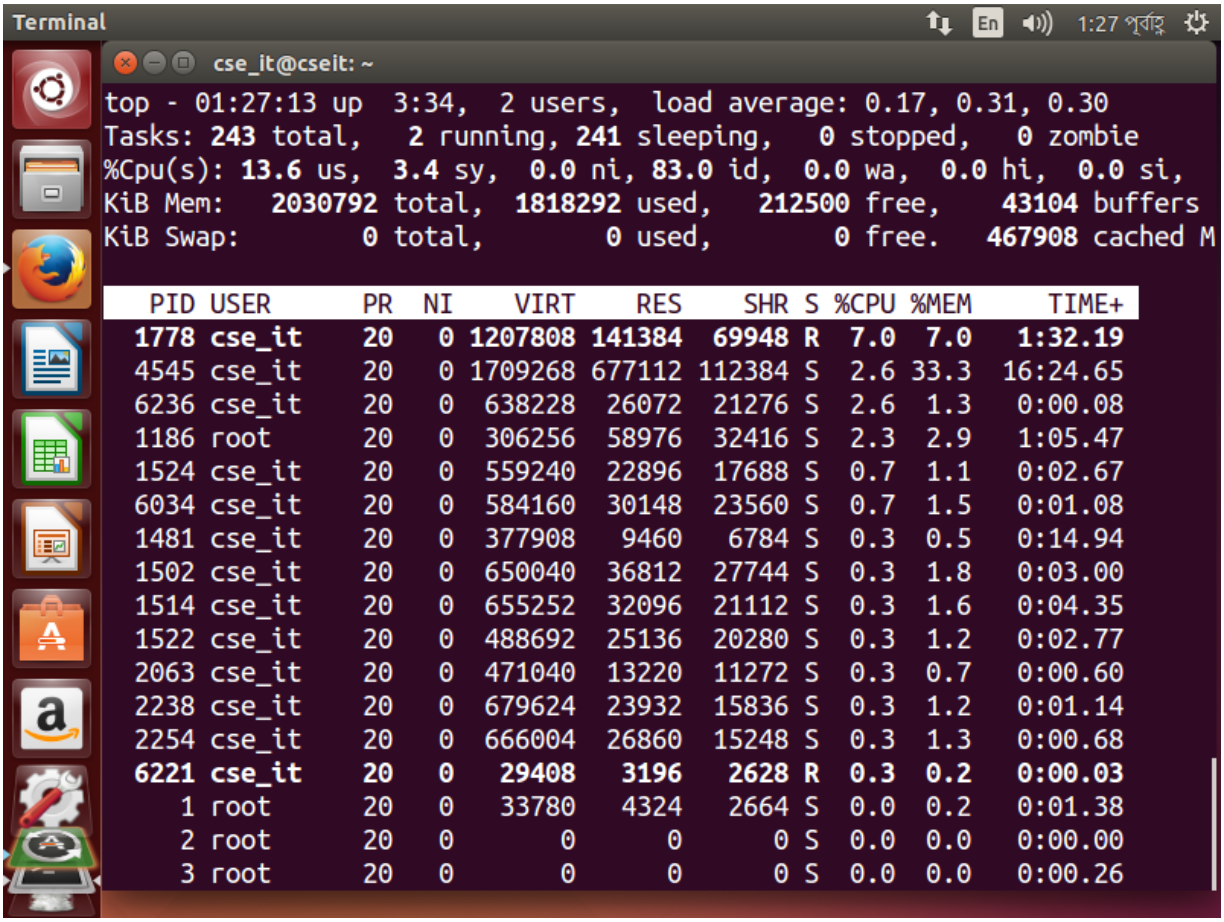
We may pipe as many programs together as we like. In the below example we have then piped the output to tail so as to get only the third file.



2.5 Process Management!

Linux, like most modern OS's is a multitasking operating system. This means that many processes can be running at the same time. As well as the processes we are running, there may be other users on the system also running stuff and the OS itself will usually also be running various processes which it uses to manage everything in general. If we would like to get a snapshot of what is currently happening on the system we may use a program called **top**.

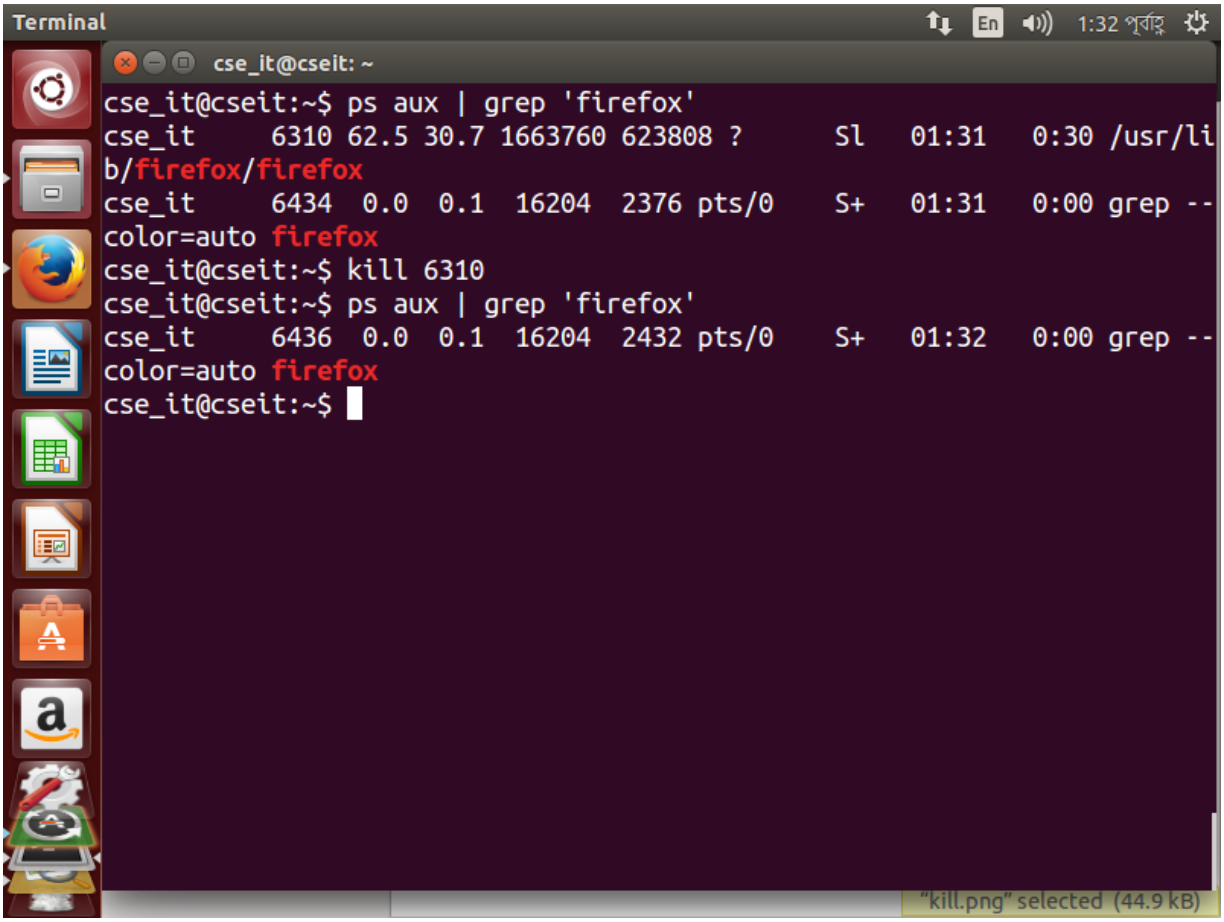
Below is a simplified version of what you should see when you run this program.



Top will give you a realtime view of the system and only show the number of processes which will fit on the screen. Another program to look at processes is called ps which stands for processes. In it's normal usage it will show you just the processes running in your current terminal (which is usually not very much). If we add the argument aux then it will show a complete system view which is a bit more helpful.

```
1
2 ps [aux]
```

It does give quite a bit of output so people usually pipe the output to grep to filter out just the data they are after. We will see in the next bit an example of this.



2.5.1 Lab Task (Please implement yourself and show the output to the instructor)

1. Practice the Linux commands:
    - Filters
    - Regular Expressions
    - Piping and Redirection
    - Process Management

3 Discussion & Conclusion

Based on the focused objective(s) to understand about the basic linux commands, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

4 Lab Exercise (Submit as a report)

- Implement the Linux commands:
    1. Filters
    2. Regular Expressions
    3. Piping and Redirection
    4. Process Management

5 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.