



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.

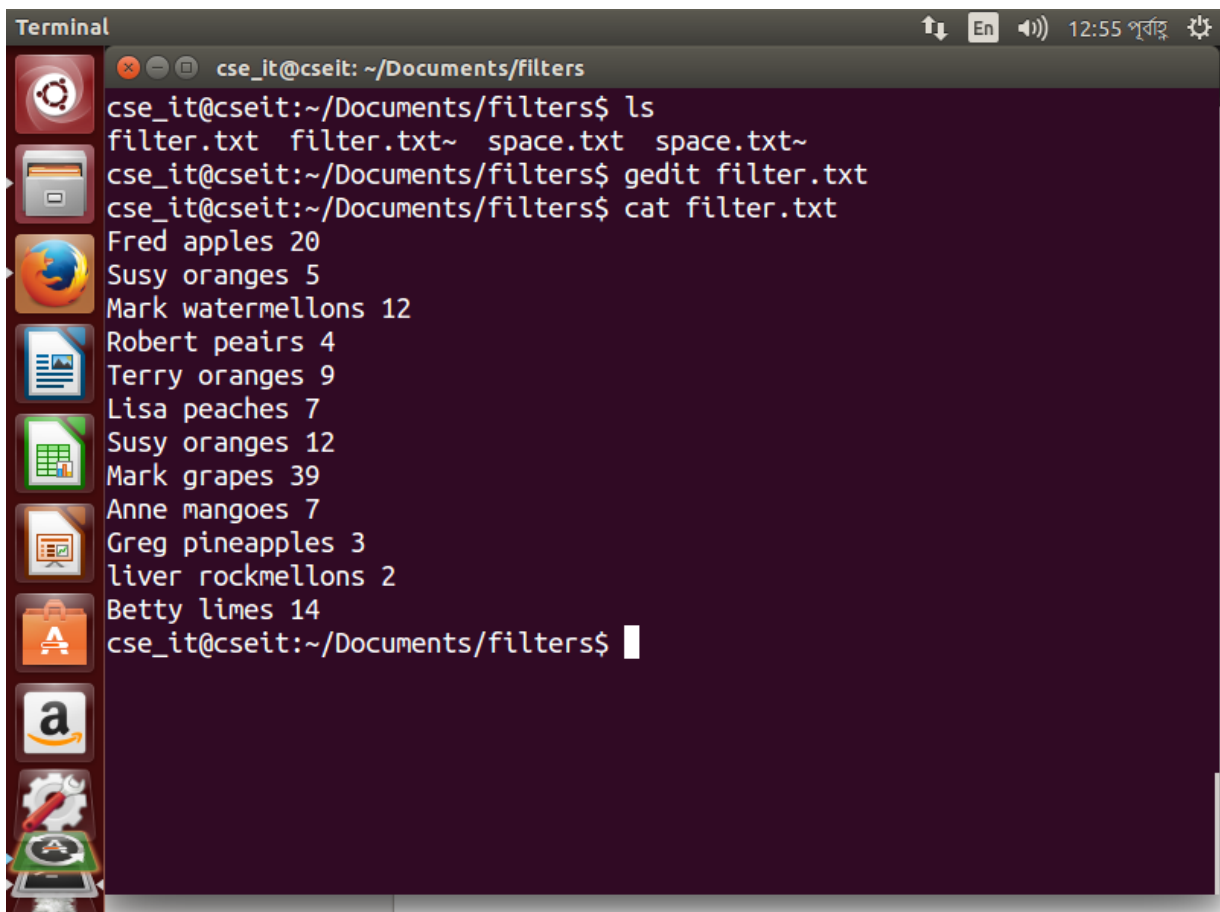
2 Linux

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:

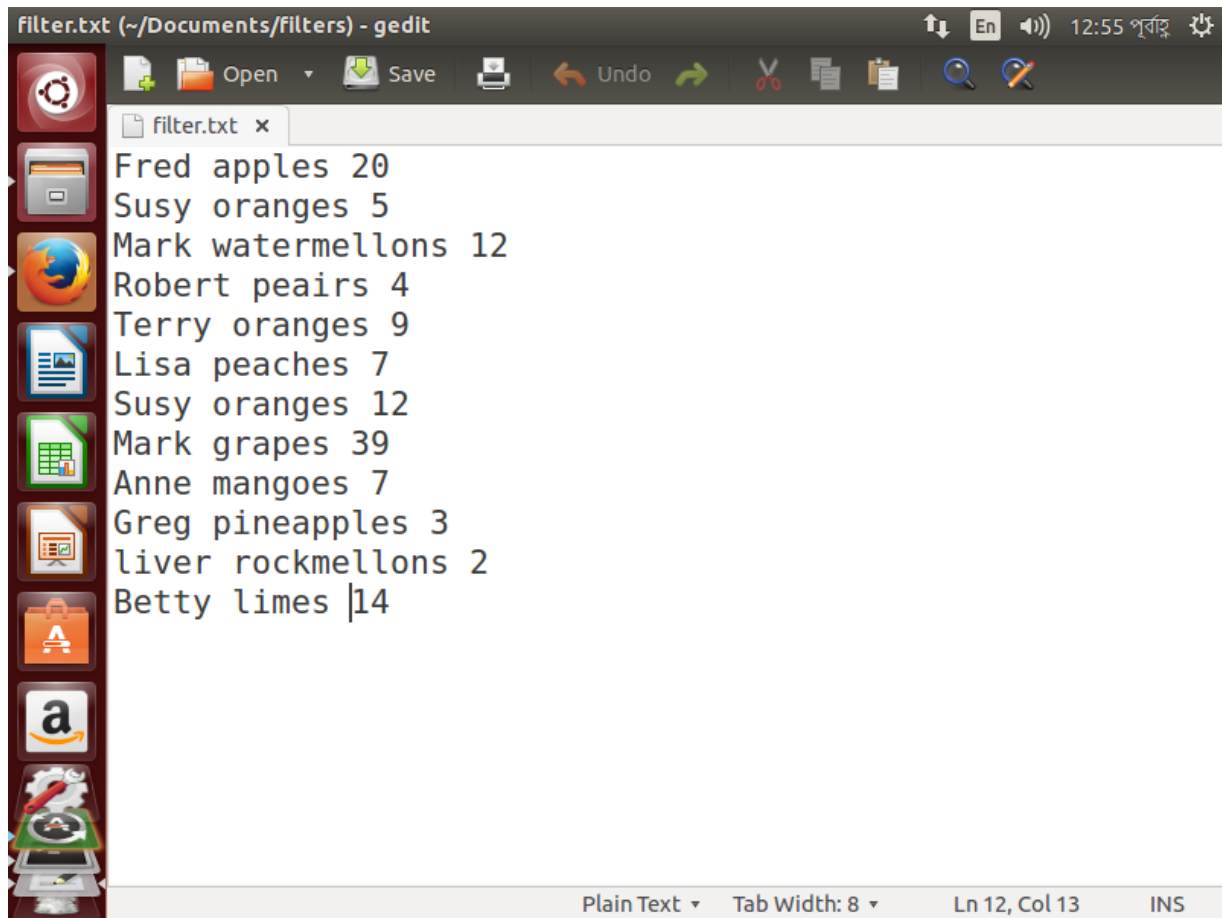
- **Filters** - An introduction to various commands that allow us to mangle data in interesting and useful ways.
- **Grep and Regular Expressions** - Master a powerful pattern matching language that is useful for analysing and processing data.
- **Piping and Redirection** - Join commands together in 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.



```
Terminal
cse_it@cseit: ~/Documents/filters
cse_it@cseit:~/Documents/filters$ ls
filter.txt  filter.txt~  space.txt  space.txt~
cse_it@cseit:~/Documents/filters$ gedit filter.txt
cse_it@cseit:~/Documents/filters$ cat filter.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert peairs 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
liver rockmellons 2
Betty limes 14
cse_it@cseit:~/Documents/filters$
```



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

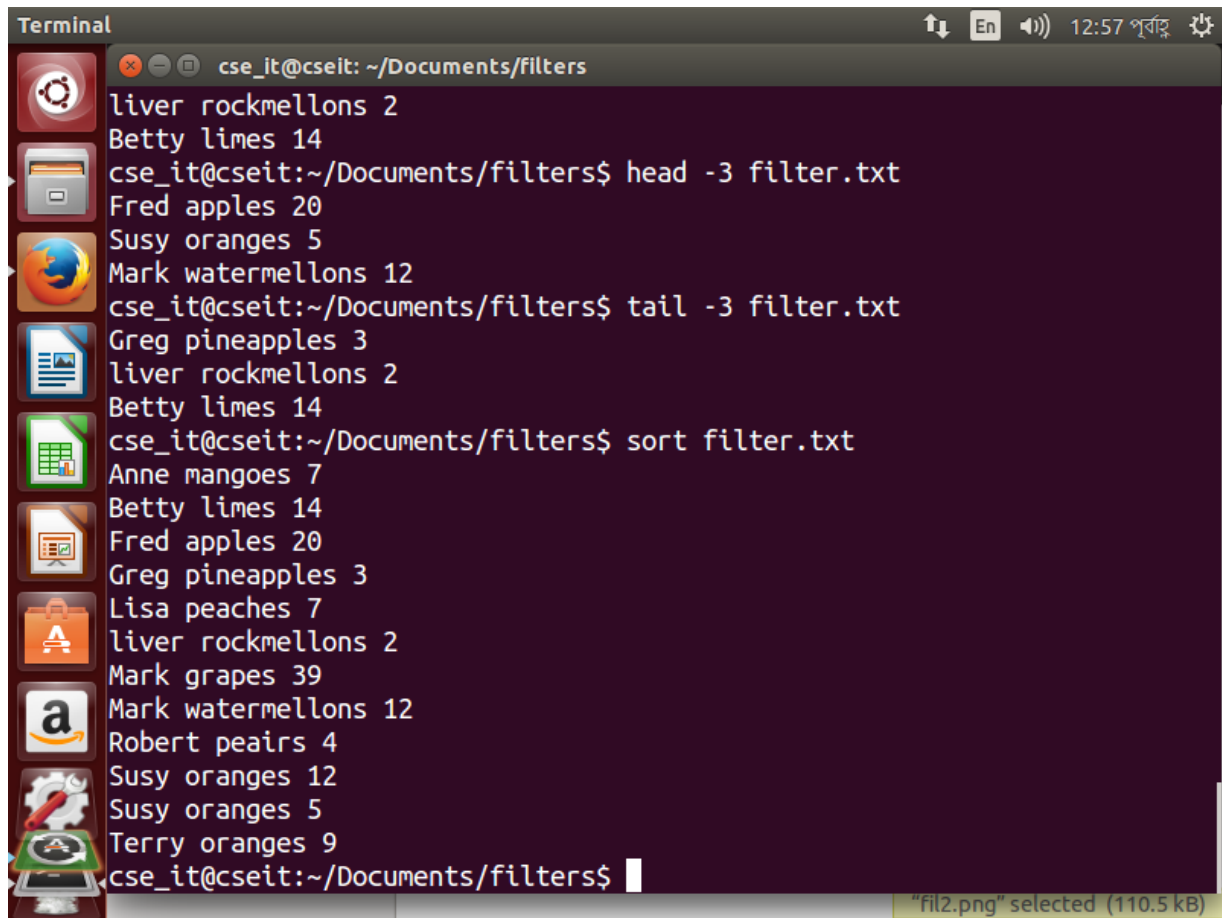
Tail is the opposite of head. Tail is a program that prints the last so many lines of it's input. By default it will print the last 10 lines but we may modify this with a command line argument.

```
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]
```

A terminal window titled "Terminal" with a dark background. The prompt is "cse_it@cseit: ~/Documents/filters". The user has entered several commands: "head -3 filter.txt" which outputs "liver rockmellons 2", "Betty limes 14", and "Fred apples 20"; "tail -3 filter.txt" which outputs "Susy oranges 5", "Mark watermellons 12", and "Greg pineapples 3"; and "sort filter.txt" which outputs a sorted list of items including "Anne mangoes 7", "Betty limes 14", "Fred apples 20", "Greg pineapples 3", "Lisa peaches 7", "liver rockmellons 2", "Mark grapes 39", "Mark watermellons 12", "Robert peairs 4", "Susy oranges 12", "Susy oranges 5", and "Terry oranges 9". The prompt is now "cse_it@cseit:~/Documents/filters\$". On the left side of the terminal, there is a vertical dock with various application icons. At the bottom right, a status bar shows "“fil2.png” selected (110.5 kB)".

```
Terminal
cse_it@cseit: ~/Documents/filters
liver rockmellons 2
Betty limes 14
cse_it@cseit:~/Documents/filters$ head -3 filter.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
cse_it@cseit:~/Documents/filters$ tail -3 filter.txt
Greg pineapples 3
liver rockmellons 2
Betty limes 14
cse_it@cseit:~/Documents/filters$ sort filter.txt
Anne mangoes 7
Betty limes 14
Fred apples 20
Greg pineapples 3
Lisa peaches 7
liver rockmellons 2
Mark grapes 39
Mark watermellons 12
Robert peairs 4
Susy oranges 12
Susy oranges 5
Terry oranges 9
cse_it@cseit:~/Documents/filters$
```

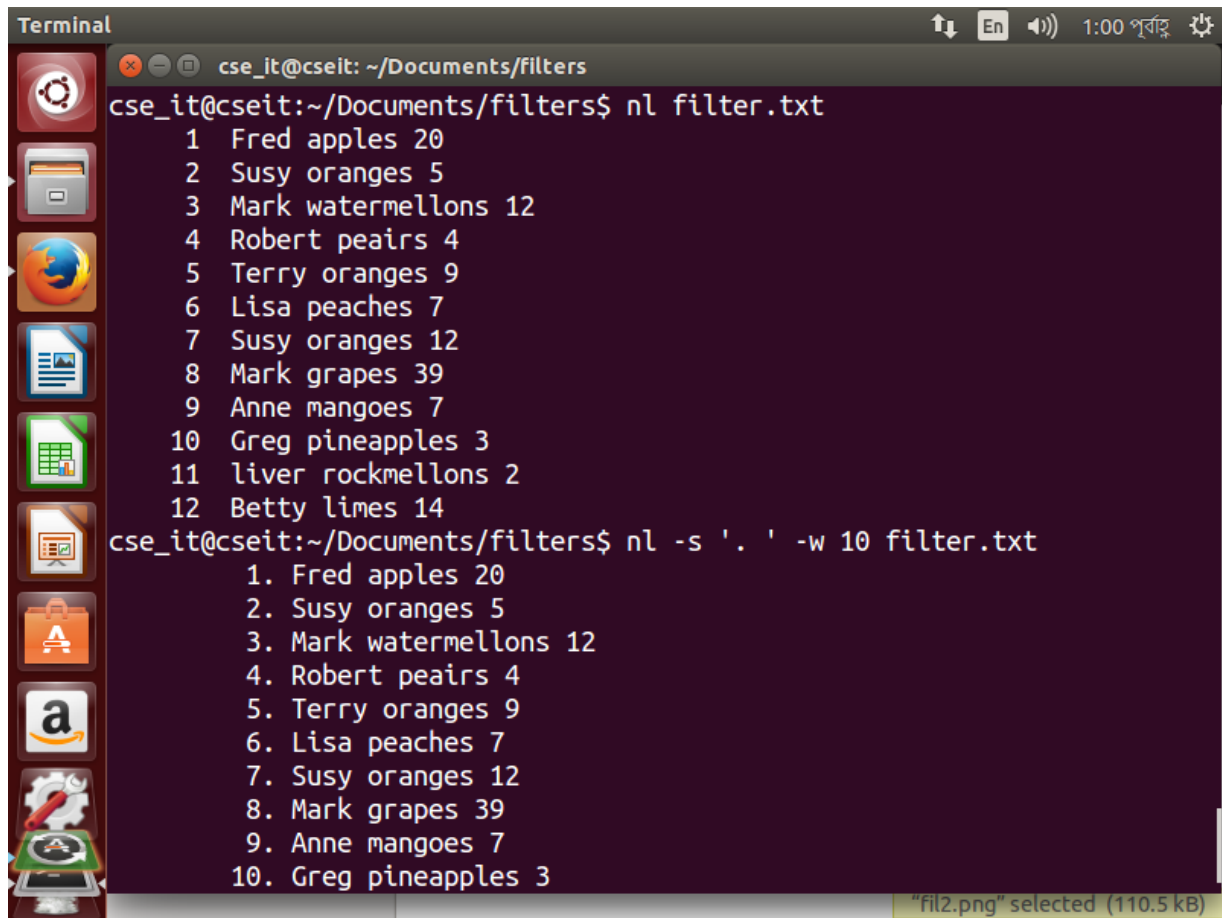
2.1.4 nl

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.



```
Terminal
cse_it@cseit: ~/Documents/filters
cse_it@cseit:~/Documents/filters$ nl filter.txt
 1 Fred apples 20
 2 Susy oranges 5
 3 Mark watermellons 12
 4 Robert peairs 4
 5 Terry oranges 9
 6 Lisa peaches 7
 7 Susy oranges 12
 8 Mark grapes 39
 9 Anne mangoes 7
10 Greg pineapples 3
11 liver rockmellons 2
12 Betty limes 14
cse_it@cseit:~/Documents/filters$ nl -s ' ' -w 10 filter.txt
 1. Fred apples 20
 2. Susy oranges 5
 3. Mark watermellons 12
 4. Robert peairs 4
 5. Terry oranges 9
 6. Lisa peaches 7
 7. Susy oranges 12
 8. Mark grapes 39
 9. Anne mangoes 7
10. Greg pineapples 3
```

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.

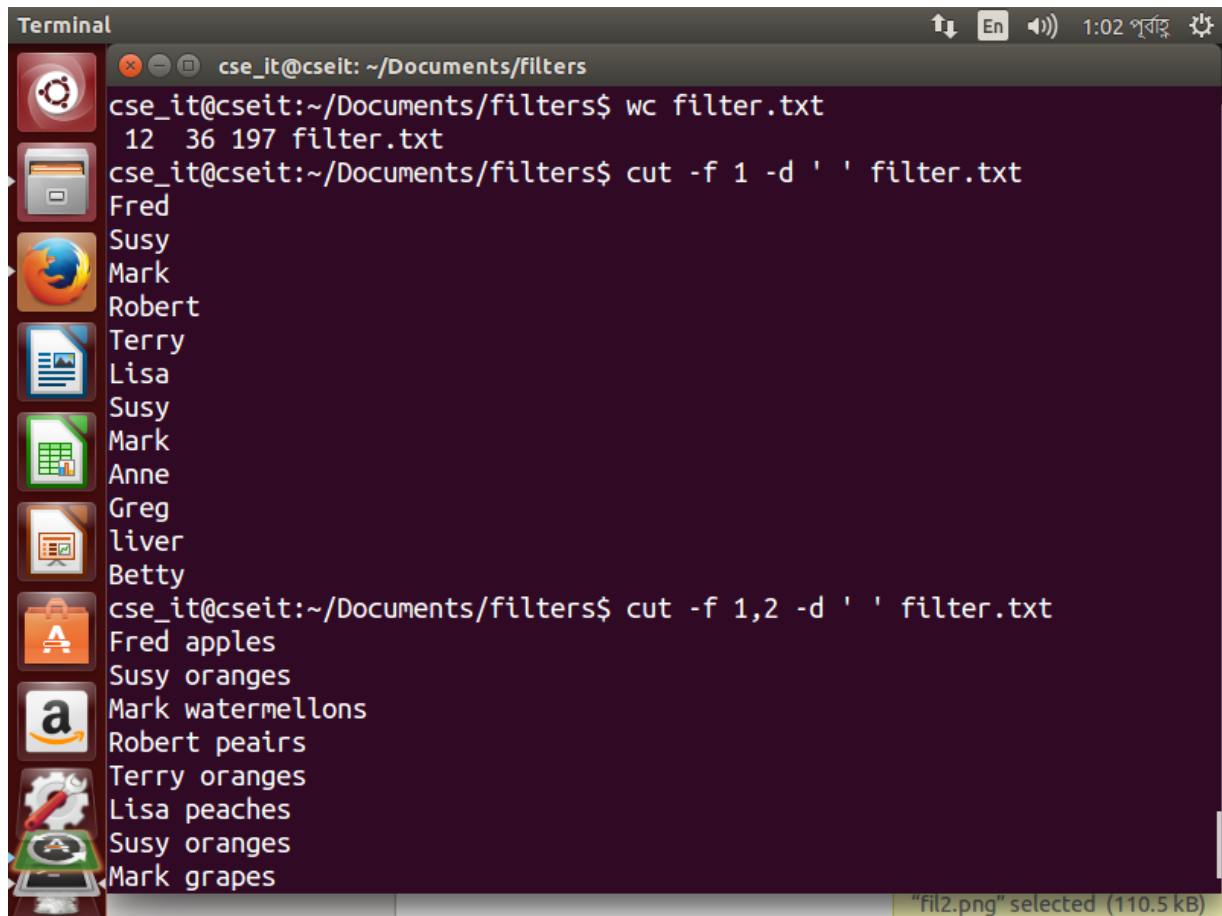
```
1 wc [-options] [path]
```

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.

A terminal window titled 'Terminal' with a dark background. The prompt is 'cse_it@cseit: ~/Documents/filters'. The user runs 'wc filter.txt' and gets '12 36 197 filter.txt'. Then they run 'cut -f 1 -d ' ' filter.txt' and see a list of names: Fred, Susy, Mark, Robert, Terry, Lisa, Susy, Mark, Anne, Greg, liver, Betty. Finally, they run 'cut -f 1,2 -d ' ' filter.txt' and see pairs of names and fruits: Fred apples, Susy oranges, Mark watermelons, Robert peairs, Terry oranges, Lisa peaches, Susy oranges, Mark grapes. A status bar at the bottom right says '"fil2.png" selected (110.5 kB)'.

```
Terminal
cse_it@cseit: ~/Documents/filters
cse_it@cseit:~/Documents/filters$ wc filter.txt
12 36 197 filter.txt
cse_it@cseit:~/Documents/filters$ cut -f 1 -d ' ' filter.txt
Fred
Susy
Mark
Robert
Terry
Lisa
Susy
Mark
Anne
Greg
liver
Betty
cse_it@cseit:~/Documents/filters$ cut -f 1,2 -d ' ' filter.txt
Fred apples
Susy oranges
Mark watermelons
Robert peairs
Terry oranges
Lisa peaches
Susy oranges
Mark grapes
```

2.2.1 sed

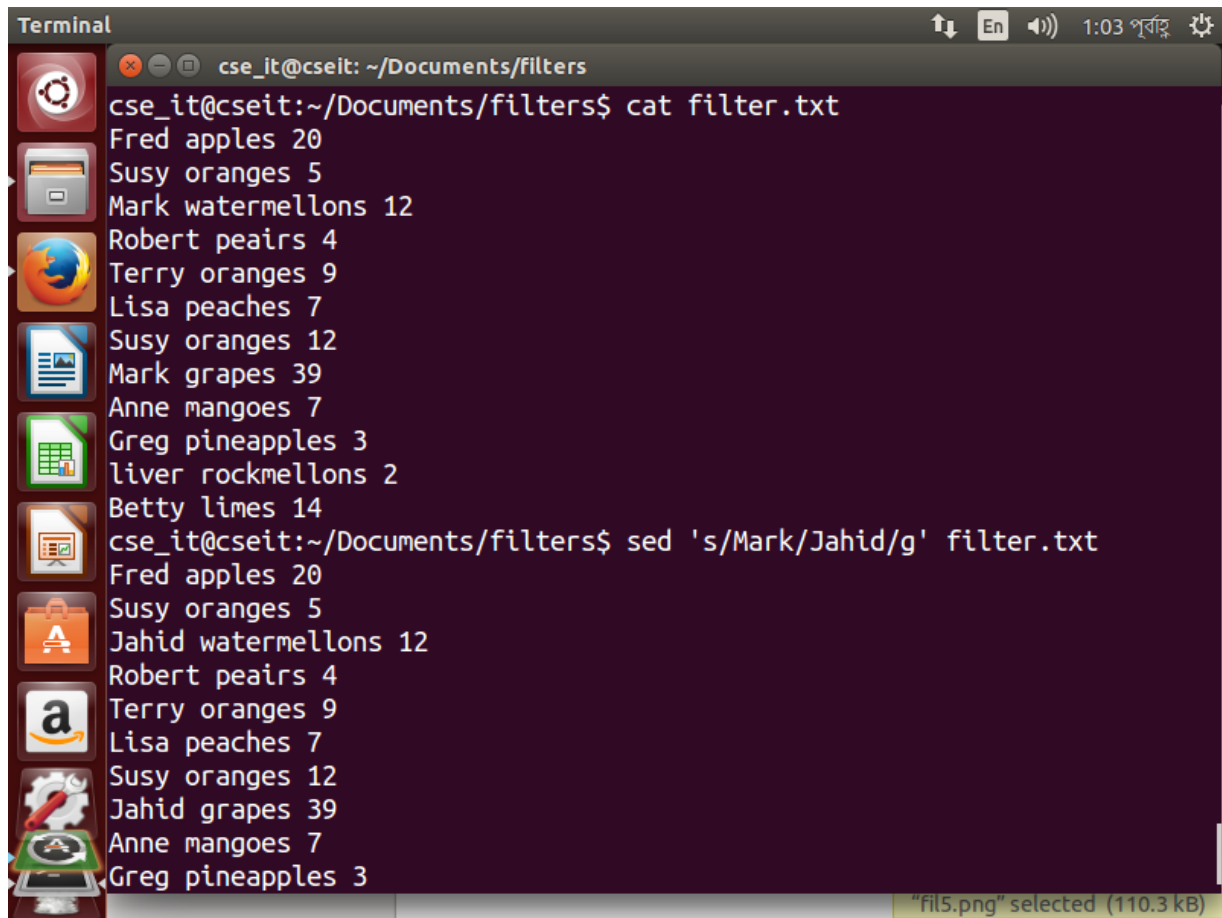
sed stands for Stream Editor and it effectively allows us to do a search and replace on our data. It is quite a powerful command but we will use it here in it's basic format.

```
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.

A terminal window titled 'Terminal' with a dark background. The prompt is 'cse_it@cseit: ~/Documents/filters'. The first command is 'cat filter.txt', which outputs a list of names, fruits, and counts: Fred apples 20, Susy oranges 5, Mark watermelons 12, Robert peairs 4, Terry oranges 9, Lisa peaches 7, Susy oranges 12, Mark grapes 39, Anne mangoes 7, Greg pineapples 3, and liver rockmellons 2. The second command is 'sed 's/Mark/Jahid/g' filter.txt', which outputs the same list but with 'Mark' replaced by 'Jahid' in two lines: 'Jahid watermelons 12' and 'Jahid grapes 39'. The terminal has a sidebar on the left with various application icons. At the bottom right, a status bar indicates '"fil5.png" selected (110.3 kB)'.

```
Terminal
cse_it@cseit: ~/Documents/filters
cse_it@cseit:~/Documents/filters$ cat filter.txt
Fred apples 20
Susy oranges 5
Mark watermelons 12
Robert peairs 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
liver rockmellons 2
cse_it@cseit:~/Documents/filters$ sed 's/Mark/Jahid/g' filter.txt
Fred apples 20
Susy oranges 5
Jahid watermelons 12
Robert peairs 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Jahid grapes 39
Anne mangoes 7
Greg pineapples 3
"fil5.png" selected (110.3 kB)
```

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.

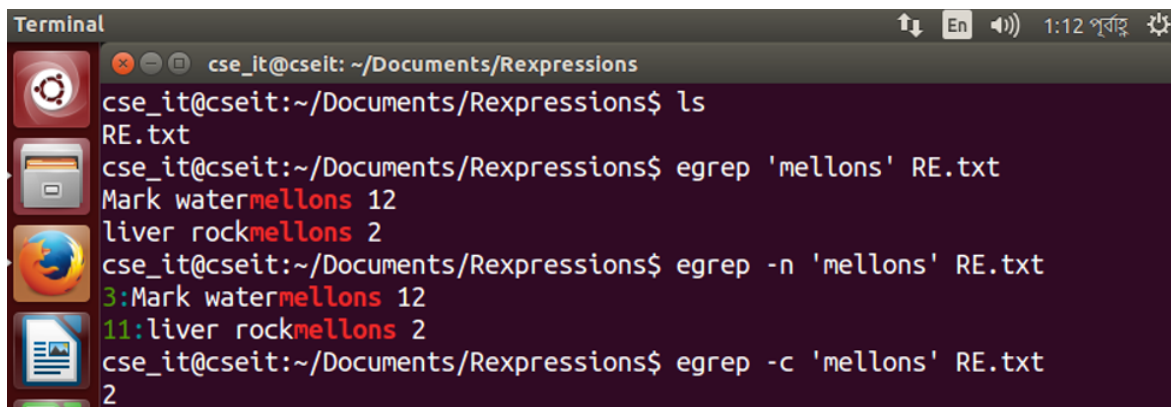
2.3.1 eGrep

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.

A terminal window titled 'Terminal' with a dark background. The prompt is 'cse_it@cseit: ~/Documents/Rexpressions'. The user enters 'ls', showing 'RE.txt'. Then 'egrep 'mellons' RE.txt' is entered, showing 'Mark watermellons 12' and 'liver rockmellons 2'. Next, 'egrep -n 'mellons' RE.txt' is entered, showing '3:Mark watermellons 12' and '11:liver rockmellons 2'. Finally, 'egrep -c 'mellons' RE.txt' is entered, showing '2'.

```
Terminal
cse_it@cseit: ~/Documents/Rexpressions
cse_it@cseit:~/Documents/Rexpressions$ ls
RE.txt
cse_it@cseit:~/Documents/Rexpressions$ egrep 'mellons' RE.txt
Mark watermellons 12
liver rockmellons 2
cse_it@cseit:~/Documents/Rexpressions$ egrep -n 'mellons' RE.txt
3:Mark watermellons 12
11:liver rockmellons 2
cse_it@cseit:~/Documents/Rexpressions$ egrep -c 'mellons' RE.txt
2
```

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.

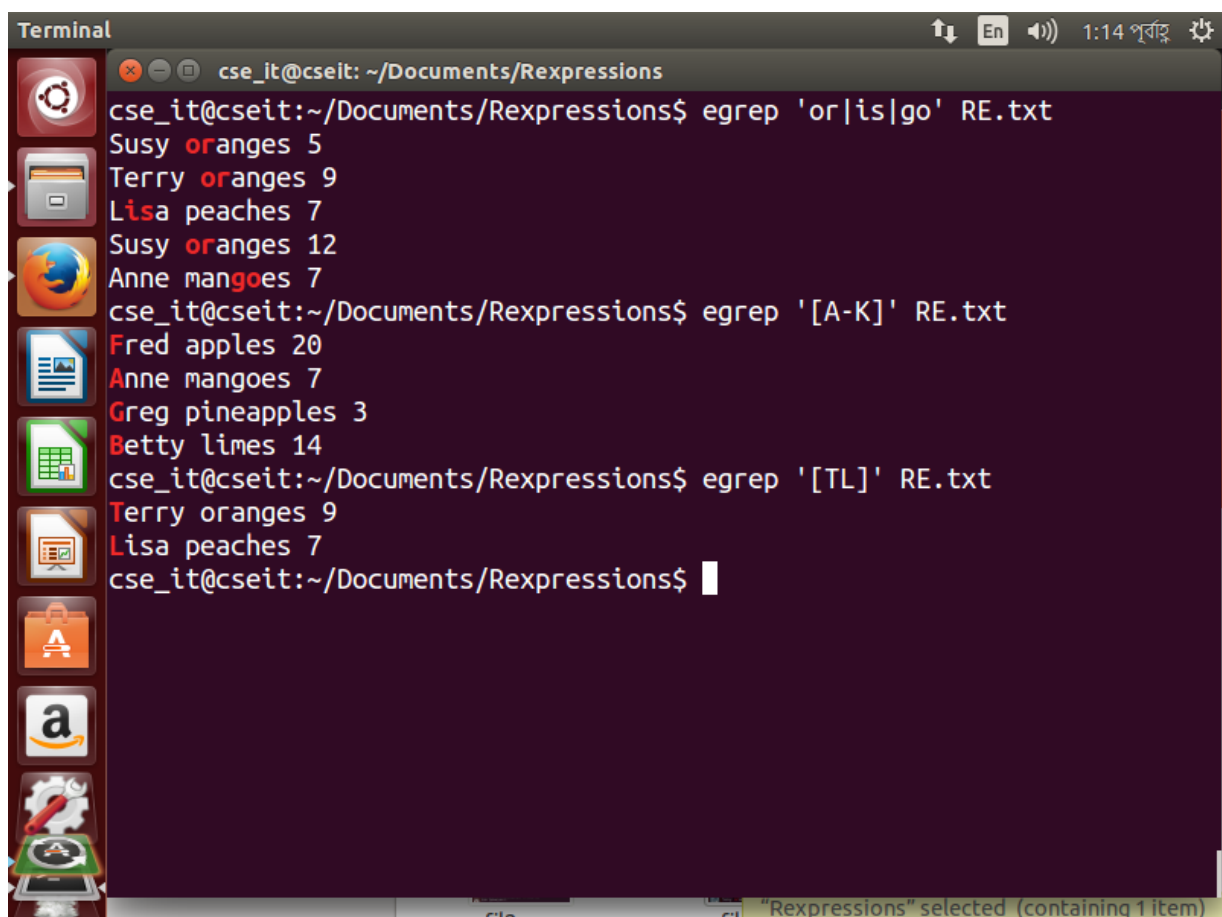
Some Examples:

- 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.



```
cse_it@cseit:~/Documents/Rexpressions$ egrep '[aeiou]{2,}' RE.txt
Robert peairs 4
Lisa peaches 7
Anne mangoes 7
Greg pineapples 3
cse_it@cseit:~/Documents/Rexpressions$ egrep '2.+' RE.txt
Fred apples 20
cse_it@cseit:~/Documents/Rexpressions$ egrep '2$' RE.txt
Mark watermellons 12
Susy oranges 12
liver rockmellons 2
cse_it@cseit:~/Documents/Rexpressions$
```

- 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.



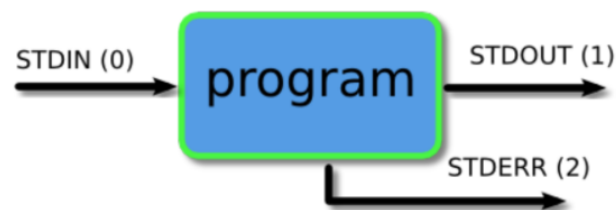
```
Terminal
cse_it@cseit: ~/Documents/Rexpressions
cse_it@cseit:~/Documents/Rexpressions$ egrep 'or|is|go' RE.txt
Susy oranges 5
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Anne mangoes 7
cse_it@cseit:~/Documents/Rexpressions$ egrep '[A-K]' RE.txt
Fred apples 20
Anne mangoes 7
Greg pineapples 3
Betty limes 14
cse_it@cseit:~/Documents/Rexpressions$ egrep '[TL]' RE.txt
Terry oranges 9
Lisa peaches 7
cse_it@cseit:~/Documents/Rexpressions$
```

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 program 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.

```

Terminal
cse_it@cseit: ~/Documents/PipRe
cse_it@cseit:~/Documents/PipRe$ ls
audio.mp3  bob          flower.jpg  foo3        thirdfile
barry.txt  example.png  foo1        frog.png    video.mp4
blah.txt   firstfile    foo2        secondfile
cse_it@cseit:~/Documents/PipRe$ ls > myoutput
cse_it@cseit:~/Documents/PipRe$ cat myoutput
audio.mp3
barry.txt
blah.txt
bob
example.png
firstfile
flower.jpg
foo1
foo2
foo3
frog.png
myoutput
secondfile
thirdfile
video.mp4
cse_it@cseit:~/Documents/PipRe$
  
```

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.

```
1  cat myoutput
2  barry.txt
3  bob
4  example.png
5  firstfile
6  fool
7  myoutput
8  video.mpeg
9  wc -l barry.txt > myoutput
10 cat myoutput
11 7 barry.txt
```

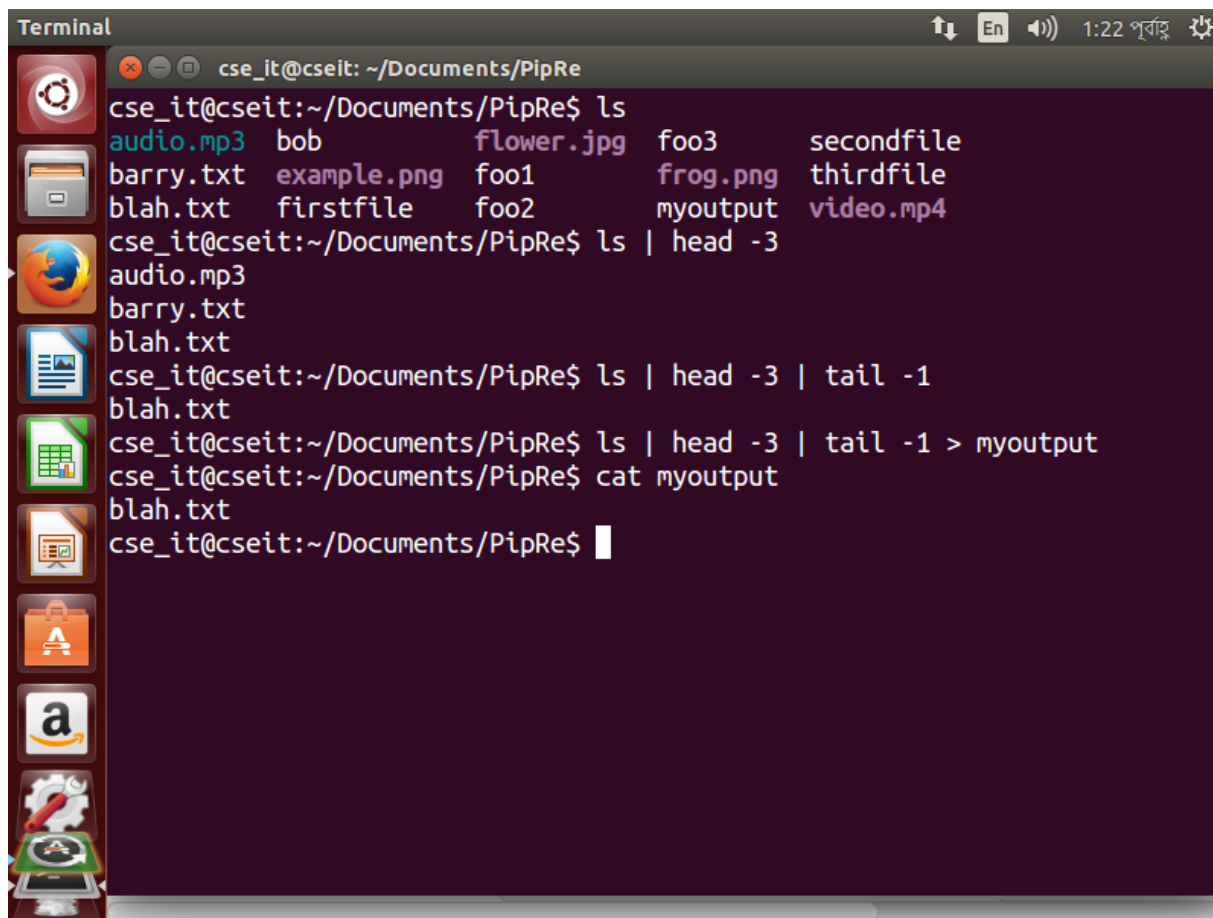
We can instead get the new data to be appended to the file by using the double greater than **operator** (»).

```
1 cat myoutput
2 7 barry.txt
3 ls >> myoutput
4 cat myoutput
5 7 barry.txt
6 barry.txt
7 bob
8 example.png
9 firstfile
10 fool
11 myoutput
12 video.mpeg
```

2.4.3 Piping

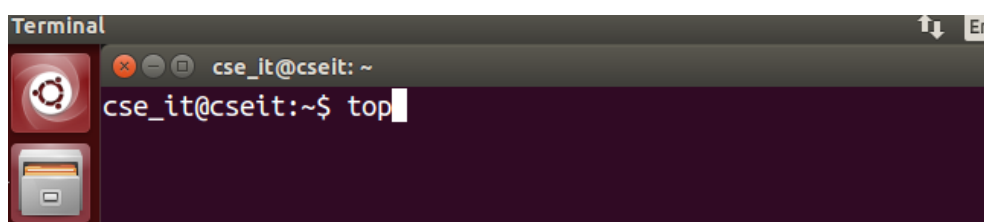
Now we'll take a look at a mechanism for sending data from one program to another. It's called piping and the operator we use is (|) (found above the backslash (\) key on most keyboards). What this operator does is feed the output from the program on the left as input to the program on the right. In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

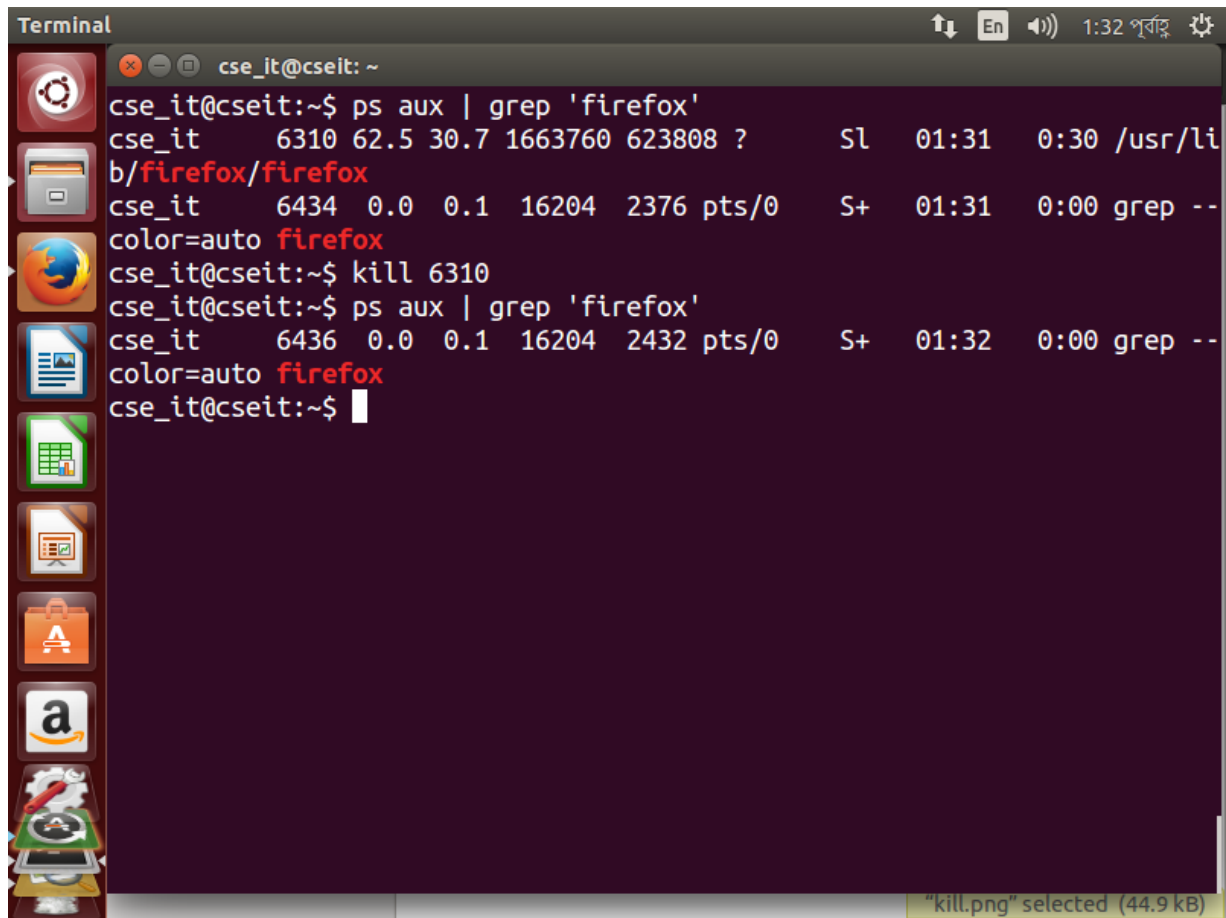
```
Terminal
cse_it@cseit: ~
top - 01:27:13 up 3:34, 2 users, load average: 0.17, 0.31, 0.30
Tasks: 243 total, 2 running, 241 sleeping, 0 stopped, 0 zombie
%Cpu(s): 13.6 us, 3.4 sy, 0.0 ni, 83.0 id, 0.0 wa, 0.0 hi, 0.0 si,
KiB Mem: 2030792 total, 1818292 used, 212500 free, 43104 buffers
KiB Swap: 0 total, 0 used, 0 free. 467908 cached M

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+
1778 cse_it    20   0 1207808 141384 69948 R   7.0   7.0   1:32.19
4545 cse_it    20   0 1709268 677112 112384 S   2.6  33.3  16:24.65
6236 cse_it    20   0 638228  26072 21276 S   2.6   1.3   0:00.08
1186 root      20   0 306256  58976 32416 S   2.3   2.9   1:05.47
1524 cse_it    20   0 559240  22896 17688 S   0.7   1.1   0:02.67
6034 cse_it    20   0 584160  30148 23560 S   0.7   1.5   0:01.08
1481 cse_it    20   0 377908   9460  6784 S   0.3   0.5   0:14.94
1502 cse_it    20   0 650040  36812 27744 S   0.3   1.8   0:03.00
1514 cse_it    20   0 655252  32096 21112 S   0.3   1.6   0:04.35
1522 cse_it    20   0 488692  25136 20280 S   0.3   1.2   0:02.77
2063 cse_it    20   0 471040  13220 11272 S   0.3   0.7   0:00.60
2238 cse_it    20   0 679624  23932 15836 S   0.3   1.2   0:01.14
2254 cse_it    20   0 666004  26860 15248 S   0.3   1.3   0:00.68
6221 cse_it    20   0  29408   3196  2628 R   0.3   0.2   0:00.03
   1 root      20   0   33780   4324  2664 S   0.0   0.2   0:01.38
   2 root      20   0        0        0        0 S   0.0   0.0   0:00.00
   3 root      20   0        0        0        0 S   0.0   0.0   0:00.26
```

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.



```
Terminal
cse_it@cseit: ~
cse_it@cseit:~$ ps aux | grep 'firefox'
cse_it      6310  62.5  30.7 1663760 623808 ?        Sl      01:31   0:30 /usr/li
b/firefox/firefox
cse_it      6434   0.0   0.1  16204   2376 pts/0    S+      01:31   0:00 grep --
color=auto firefox
cse_it@cseit:~$ kill 6310
cse_it@cseit:~$ ps aux | grep 'firefox'
cse_it      6436   0.0   0.1  16204   2432 pts/0    S+      01:32   0:00 grep --
color=auto firefox
cse_it@cseit:~$
```

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.