CS101
Linux Shell Handout

Introduction

This handout is meant to be used as a quick reference to get a beginner level hands on experience to using Linux based systems.

We prepared this handout assuming the target audience has zero to very limited exposure to linux or a linux command shell.

What is a Linux Command Shell??

- It is just a program that interprets your commands and executes them.
- All your basic tasks like moving files, renaming directories etc. - all can be done using some simple commands we will see in some time.
- To get a live picture , fire up the terminal by pressing Ctrl+Alt+T. You will see a window with some text say - “labuser@slxx:~$”.
- To execute a command just type a valid command - say “ls” or “pwd” and press enter.

Why the command shell??

- A natural question one has in mind is - Why the command shell? Can’t I just use the file browser to do all my work?
- Sure you can but as you will realise in some time, the command line is an extremely powerful tool with which you can perform seemingly complex task using some clever combination of commands.
- For example, say you want to download all the PDFs from the CS101 webpage. How would you normally do it? Well one could click individually on all links and download them. Some Windows users might prefer IDM. Using command line? A simple command “wget -np -nd -r -N -A pdf,txt -e robots=off https://www.cse.iitb.ac.in/~cs101/” does the trick. You can try it later on.

List of commands:

Command	Description
pwd	“Print Working Directory”. Shows the current location in the directory tree.
cd	“Change Directory”. When typed all by itself, it returns you to your home directory.
cd <directory>	Change into the specified directory name. Example: cd /usr/src/linux
cd ~	“~” is an alias for your home directory. It can be used as a shortcut to your “home”, or other directories relative to your home
cd ..	Move up one directory. For example, if you are in ~/vic and you type “cd ..”, you will end up in ~.
cd -	Return to previous directory. An easy way to get back to your previous location
ls	List all files in the current directory, in column format
ls directory	List the files in the specified directory. Example: ls /var/log
ls -l	List files in “long” format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
ls -a	List all files, including “hidden” files. Hidden files are those files that begin with a “.”, e.g. The .bash_history file in your home directory
ls -ld directory	A “long” list of “directory”, but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following two commands: ls -l /usr/bin ls -ld /usr/bin
ls /usr/bin/d*	List all files whose names begin with the letter “d” in the /usr/bin directory
cat	Display the contents of a text file on the screen. For example: cat ~/.bash_history
cp	Copies a file from one location to another Example: cp <source_file> <dest_file>

mv	Moves a file to a new location, or renames it. Example: mv <source_file> <dest_file>
rm	Removes a file Example: rm <file>
rm -r	Remove a directory. Example: rm -r ~/<directory> **CAUTION: Be careful of what you delete. Files don't go to a recycle bin or anything like that**
mkdir	Make Directory. Example: mkdir ~/temp
touch	Make a file if it doesn't exist. If its exists modify its timestamp.
man	See manual page of a command to get more of its information. Example: man touch Hint: Press `q` to quit.
xargs	Build and execute command lines from standard input Example: echo dir1 dir2 dir3 | xargs -n1 cp file Copies `file` to each of dir1, dir2 and dir3
wget	Download a resource from web. For example, link of your CS101 Lecture 18 is https://www.cse.iitb.ac.in/~cs101/slides/Lecture18.pdf. Running "wget <*link*>" downloads the file to you current directory.

Some useful shortcuts:

1. **Up/Down Arrow Keys:** Scroll through the history of commands.
2. **TAB completion:** Type in just the first few letters of your command/file and press TAB. Shell will try to complete the command/file name
3. **CTRL-R:** Press CTRL-R and then type any portion of a recent command. It will search the commands for you, and once you find the command you want, just press ENTER.
4. **CTRL-L:** Somewhat same as `clear`. Clean up the terminal.
5. **CTRL-A**: Go to the beginning of the line you are currently typing on
6. **CTRL-E**: Go to the end of the line you are currently typing on
7. **CTRL-U**: Clear the line from the cursor position to the beginning.
8. **CTRL-C**: Kill the current running process
9. **ALT-F**: Move the cursor forward by one word

10. **ALT-B**: Move the cursor back by one word
11. **!!** : run last command. Example: Previous command: `cd temp` but temp doesn't exist. Just do a `mkdir temp && !!`
12. **!blah** : run the most recent command that starts with 'blah' (e.g. !ls)

Process related:

1. **ps**
2. **top/htop**
3. **kill/pkill/killall**
4. **pidof**

Compiling C++ programs and running them

Source: https://community.linuxmint.com/tutorial/view/860

GCC (GNU Compiler Collection) is installed by default, in Ubuntu. To compile the program, open the terminal and go to the target directory (where your cpp file is present) and type the command (g++ is the compiler name, hello.cpp is the filename of the source program while -o option specifies the filename of the output executable hello)

```
g++ hello.cpp -o hello
```

Note that if the -o option is not used (i.e. if no output file is given) the executable generated will be named a.out.

If there is no syntax/semantic error in you program then the compiler will successfully generate an executable file (named hello), otherwise fix the problem in your code.

To execute the program, you must run the following command:

```
./hello
```

Type the input on the terminal. After all inputs have been given, the program will display the output on the terminal.

Taking input from a file and printing output to a file

Consider the program sum.cpp. This program takes an integer N and integers a[0] .. a[N-1]. It prints out the sum of the N integers input. For this program, you may want to put your input in a file instead of typing the input each time the program is run. The compilation step will not change (same as above).

```
g++ sum.cpp -o sum
```

To take input directly from a file, run the following command on the terminal:

```
./sum < input.txt
```

Above command will print the output sum on the terminal itself. If you want to redirect the output also to a file, run the following command:

```
./sum < input.txt > output.txt
```

How to use header file in Program

Both user and system header files are include using the pre-processing directive #include. It has following two forms:\

Syntax

```
#include<file>
```

This form is used for system header files. It searches for a file named file in a standard list of system directives.

Syntax

```
#include"file"
```

This form used for header files of our own program. It searches for a file named file in the directive containing the current file.

Note: The use of angle brackets **<>** informs the compiler to search the compilers include directory for the specified file. The use of the double quotes **""** around the filename inform the compiler to search in the current directory for the specified file.

Example:

The '#include' directive works by directing the C preprocessor to scan the specified file as input before continuing with the rest of the current file. The output from the preprocessor contains the output already generated, followed by the output resulting from the included file, followed by the output that comes from the text after the '#include' directive. For example, if you have a header file header.h as follows,

```
  char *test (void);
```

and a main program called program.c that uses the header file, like this,

```
  int x;
   #include "header.h"

   int
   main (void)
   {
     puts (test ());
   }
```

the compiler will see the same token stream as it would if program.c read

```
  int x;
   char *test (void);

   int
   main (void)
   {
```

```
    puts (test ());
  }
```

Included files are not limited to declarations and macro definitions; those are merely the typical uses. Any fragment of a C program can be included from another file. The include file could even contain the beginning of a statement that is concluded in the containing file, or the end of a statement that was started in the including file. However, an included file must consist of complete tokens. Comments and string literals which have not been closed by the end of an included file are invalid. For error recovery, they are considered to end at the end of the file.

To avoid confusion, it is best if header files contain only complete syntactic units—function declarations or definitions, type declarations, etc.

The line following the '#include' directive is always treated as a separate line by the C preprocessor, even if the included file lacks a final newline.

7

File IO in cpp

A file must be opened before you can read from it or write to it. Either the **ofstream** or **fstream** object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.

Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```

Here, the first argument specifies the name and location of the file to be opened and the second argument of the **open()** member function defines the mode in which the file should be opened.

Mode Flag	Description

ios::app	Append mode. All output to that file to be appended to the end.
ios::ate	Open a file for output and move the read/write control to the end of the file.
ios::in	Open a file for reading.
ios::out	Open a file for writing.
ios::trunc	If the file already exists, its contents will be truncated before opening the file.

You can combine two or more of these values by **OR**ing them together. For example if you want to open a file in write mode and want to truncate it in case it already exists, following will be the syntax:

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );
```

Similar way, you can open a file for reading and writing purpose as follows:

```
fstream  afile;
afile.open("file.dat", ios::out | ios::in );
```

Closing a File

When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

```
void close();
```

Writing to a File:

While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen. The only difference is that you use an **ofstream** or **fstream** object instead of the **cout** object.

Reading from a File:

You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard. The only difference is that you use an **ifstream** or **fstream** object instead of the **cin** object.

Read & Write Example:

Following is the C++ program which opens a file in reading and writing mode. After writing information inputted by the user to a file named afile.dat, the program reads information from the file and outputs it onto the screen:

```
#include <fstream>
#include <iostream>
using namespace std;

int main ()
{

  char data[100];

  // open a file in write mode.
  ofstream outfile;
  outfile.open("afile.dat");

  cout << "Writing to the file" << endl;
  cout << "Enter your name: ";
  cin.getline(data, 100);

  // write inputted data into the file.
  outfile << data << endl;
```

```
   cout << "Enter your age: ";
   cin >> data;
   cin.ignore();

   // again write inputted data into the file.
   outfile << data << endl;

   // close the opened file.
   outfile.close();

   // open a file in read mode.
   ifstream infile;
   infile.open("afile.dat");

   cout << "Reading from the file" << endl;
   infile >> data;

   // write the data at the screen.
   cout << data << endl;

   // again read the data from the file and display it.
   infile >> data;
   cout << data << endl;

   // close the opened file.
   infile.close();

   return 0;
}
```

When the above code is compiled and executed, it produces the following sample input and output:

```
$./a.out
Writing to the file
Enter your name: Zara
Enter your age: 9
Reading from the file
Zara
9
```

Above examples make use of additional functions from cin object, like getline() function to read the line from outside and ignore() function to ignore the extra characters left by previous read statement.

Using the Vim editor

The **vi** editor is a very powerful tool and has a very extensive built-in manual, which you can activate using the **:help** command when the program is started (instead of using **man** or **info**, which don't contain nearly as much information). We will only discuss the very basics here to get you started. You can refer http://www.openvim.com/ for another interactive tutorial.

vim has a very steep learning curve but nonetheless it is an extremely powerful tool, in fact more powerful than most of the modern GUI based editors. In addition one can add a large array of plugins like NERDTREE, gitgutter, YouCompleteMe etc. which boost productivity.

What makes **vi** confusing to the beginner is that it can operate in two modes: command mode and insert mode. The editor always starts in command mode. Commands move you through the text, search, replace, mark blocks and perform other editing tasks, and some of them switch the editor to insert mode.

This means that each key has not one, but likely two meanings: it can either represent a command for the editor when in command mode, or a character that you want in a text when in insert mode.

Moving through the text is usually possible with the arrow keys. Other navigation keys:

- **h** to move the cursor to the left
- **l** to move it to the right
- **k** to move up
- **j** to move down
- **0** moves cursor to beginning of line
- **$** moves cursor to end of line

- **w** move to beginning of next word
- **n-gg** takes you to line number n

SHIFT-G will put the prompt at the end of the document.

Basic operations

These are some popular **vi** commands:

- **n dd** will delete n lines starting from the current cursor position.
- **n dw** will delete n words at the right side of the cursor.
- **x** will delete the character on which the cursor is positioned
- **:n** moves to line n of the file.
- **:w** will save (write) the file
- **:q** will exit the editor.
- **:q!** forces the exit when you want to quit a file containing unsaved changes.
- **:wq** will save and exit
- **:w newfile** will save the text to newfile.
- **:wq!** overrides read-only permission (if you have the permission to override permissions, for instance when you are using the *root* account.
- **/astring** will search the string in the file and position the cursor on the first match below its position. Press **n** to get to the next occurrence of the word, **N** for the previous one.
- **/** will perform the same search again, moving the cursor to the next match.
- **:1, $s/word/anotherword/g** will replace all occurrences of *word* with *anotherword* throughout the file.
- **yy** will copy a block of text.
- **n p** will paste it n times.
- **:recover** will recover a file after an unexpected interruption.

Commands that switch the editor to insert mode

- **a** will append: it moves the cursor one position to the right before switching to insert mode
- **i** will insert

- **o** will insert a blank line under the current cursor position and move the cursor to that line.

Pressing the **Esc** key switches back to command mode. If you're not sure what mode you're in because you use a really old version of **vi** that doesn't display an "INSERT" message, type **Esc** and you'll be sure to return to command mode. It is possible that the system gives a little alert when you are already in command mode when hitting **Esc**, by beeping or giving a visual bell (a flash on the screen). This is normal behavior.