**Operating System Lab-1**

**Basic Linux Commands**

There are many common Linux commands that will be helpful to you, if you ever even use the command line interface in Linux. Most average users just use the graphical user interface instead which usually has many tools and front-ends to Linux common commands. This Linux tutorial on command commands will help even the average user in case X server crashes, fails, is not properly configured, etc. So continue reading for some of the more common Linux bash commands.

Some of the more common Linux shell commands are listed below for more information on each command you can always run man [command] and this will bring up the manpage for that command, you can also click on the commands listed for some common examples and syntax.

First before I list them any syntax in [] will need some kind of input from you normally, for example:

man [command] you will want to actually replace [command] with the shell command you want to read the man page for: man ls will give you the man page for the Linux shell command ls.

* linux [ls](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/list-files-and-folders-with-linux-ls-command/" \o "Linux ls Command) command – is used to list files on the filesystem.
* [file](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-file-info-command/) – command that will check the filetype, this will output to you what the file type is no matter what the extension is.
* [mkdir](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/mkdir-how-to-create-a-directory-in-linux/) command – used to make directories on the filesystem.
* [cd](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/change-directory-using-cd-linux-command/) – is used for changing into a different directory in the Linux shell
* [cp](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/cp-linux-copy-command/) – is the Linux copy command, this shell command is used to copy files|directories from one location on the filesystem to another.
* [mv](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/how-to-move-files-in-linux-mv-command/) – the Linux terminal command to move files|directories. Like the [cp](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/cp-linux-copy-command/) command, but deletes the original source.
* [rm](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/how-to-delete-in-linux-terminal-rm-command/) – shell command in Linux to remove files|directories.
* Linux [cat](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-cat-command/) command- this command is used to print|view the contents of a file to the screen|terminal.
* [grep](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-grep-command/) – command used to search|find contents of a file and print|view on your terminal|screen.
* Linux [more](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-more-and-less-commands/) and [less](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-more-and-less-commands/) – commands that will allow you to read output of files, unlike [cat](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-cat-command/) that will output the entire file at once, even if it is too large for your terminal [more](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-more-and-less-commands/) and [less](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-more-and-less-commands/) will output only as many lines as the shell you are in can output, and allow you to scroll through the file contents.
* [chown](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-chown-command-change-owner/) – Linux command to change ownership of a file|directory.
* Linux [chmod](http://beginnerlinuxtutorial.com/help-tutorial/basic-linux-commands/linux-chmod-command/" \o "Linux chmod Command) – command that allows you to change mode of user access|permissions, basically set read, write, and execute permissions.
* Linux ps – lists the current running processes on your Linux system
* Linux kill and killall commands – used to kill|terminate running processes
* **Quick Tutorial for Editor *vi***
  + **Invoking *vi***

If you write a command

vi my\_file

you will see the screen with a column of tildes. The editor vi is now in so called *command mode*.

The screen looks like:

~

~

~

~

~

~

~

~

~

The two basic commands are the following:

|  |  |
| --- | --- |
| *i* | Insert text to the left of cursor |
| *a* | Insert text to the right of cursor |

Since you are at the beginning of an empty file it does not matter which of these you type. Write a text:

**…………………………………………..**

**…………………………………….**

* + **Cursor Movements Commands**

You need to be in the command mode. If you do not know what mode is actual, press the buttom esc. This keystroke always turns the editor in the command mode. Then you can move along the screen if you keystrokes the button:

|  |  |
| --- | --- |
| Keystroke of | Doing |
| h | Cursor is moved one space to the left |
| j | Cursor is moved one line down |
| k | Cursor is moved one line up |
| l | Cursor is moved one space to the right |

* + **Deleting Text**

If you are in command mode then

|  |  |
| --- | --- |
| Keystroke of | Doing |
| x | Delete one character at the cursor |
| dd | Delete one line where the cursor is placed |

If you are in command mode and you will write

:set smd nu

(where smd means *Show MoDe* and nu means *NUmber*),  
you will now see the *mode* at the right down corner (usually) and all lines are numbered.

1 ………………………..,

2 …………………….

3 ………………

4 ……………………………

5 ………………………….

6

7

* + **File Saving**

You must be in command mode. You can use then several tricks to save the file:

|  |  |
| --- | --- |
| Keystrokes | Doing |
| :x | Write file to the disk and finish |
| ZZ | Write file to the disk and finish |
| :w | Write file to the disk and continue |

* + **Replace Mode**

Very useful is so called *replace mode* which enables overwrite the existing text.

|  |  |
| --- | --- |
| Keystrokes | Doing |
| :r | Replace one character over the cursor |
| :R | Overwrite text until the next action (e.g. keystroke of *esc*) |

**Compiling a C program:**

The program can be easily *compiled* (i.e., converted to its final, ready-to-run form) using the C [compiler](http://www.linfo.org/compiler.html) in the GCC ([GNU](http://www.linfo.org/gnu.html) Compiler Collection), which is included in most[distributions](http://www.linfo.org/distributions_list.html) (i.e., versions) of Linux. All that is necessary is to copy and save it with a [text editor](http://www.linfo.org/text_editor.html) and, assuming that this file is named *aveg.c* and that the final program is to be named *aveg*, run the following command:

gcc -o aveg aveg.c

The resulting program can be run by merely typing in its name preceded by a [dot](http://www.linfo.org/dot.html) and a [forward slash](http://www.linfo.org/forward_slash.html) and then pressing the ENTER key as follows:

./aveg

**Shell Commands and Shell script Programming**

The term "shell" sometimes confuses beginners. When referred to by Linux users, the term "shell" means using the command line interface. It is important to know that using a shell is similar to use the DOS prompt. Linux also has a GUI (Graphical User Interface - pronounced gew-eee) that is similar to Windows. Using the shell is often frustrating for beginners because you must know the commands, what they do, and the proper way to enter them. This means using the correct syntax. The Linux OS is controlled by the kernel, which is the heart of entire system. However, the kernel can only understand machine code. This is why a shell must be used. The shell interprets commands given by the user and translates them into machine code that the kernel can understand.

***Types of Shells***

There are several different types of shells available. Each shell has its own pro's and con's, but each shell can perform the same basic tasks. The main difference between them is the prompt, and how they interpret commands. The table below shows the most common shells and their attributes.

**Bourne Shell**

The original Bourne shell is named after its developer at Bell Labs, Steve Bourne. It was the first shell used for the UNIX operating system, and it has been largely surpassed in functionalit y by many of the more recent shells. However, all UNIX and many Linux versions allow users to switch to the original Bourne Shell, known simply as "sh”.

**C Shell**

The C shell, as its name might imply, was designed to allow users to write shell script program s using syntax very similar to that of the C programming language. It is known as "csh."

**TC Shell**

TC shell is an expansion upon the C shell. It has all the same features, but adds the ability to use keystrokes from the Emacs word processor program to edit text on the command line. For example, users can press Esc-D to delete the rest of the highlighted word. It is also known as

"tcsh."

**Korn Shell**

Korn Shell was also written by a developer at Bell Labs, David Korn. It attempts to merge the features of the C shell, TC shell and Bourne shell under one package. It also includes the ability for developers to create new shell commands as the need arises. It is known as "ksh."

**Bourne-Again Shell**

The Bourne-Again shell is an updated version of the original Bourne shell that was created by the Free Software Foundation for its open source GNU project. For this reason, it is a widely used shell in the open source community. Its syntax is similar to that used by the Bourne shell, however it incorporates some of the more advanced features found in the C, TC and Korn shells.

Among the added features that Bourne lacked are the ability to complete file names by pressing the TAB key, the ability to remember a history of recent commands and the ability to run multiple programs in the background at once. It is known as "bash."

**Script programming**

A shell script is a plain-text file that contains shell commands. It can be executed by typing its name into a shell, or by placing its name in another shell script. To be executable, a shell script file must meet some conditions:

The file must have a special first line that names an appropriate command processor. For this tutorial, the following will work in most cases:

#!/bin/bash

If this example doesn't work, you will need to find out where your Bash shell executable is located and substitute that location in the above example. Here is one way to find out:

$where is bash

The file must be made executable by changing its permission bits. An example:

$ chmod +x (shell script filename)

A shell script file may optionally have an identifying suffix, like ".sh". This only helps the user remember which files are which. The command processor responsible for executing the file uses the executable bit, plus the file's first line, to decide how to handle a shell script file.

Execution of shell script

$ ./scriptname.sh

This special entry is a way to tell the command processor that the desired script is located in the current directory. Always remember: if you cannot get your shell script to run, remember this trick to provide its location as well as its name.

First Shell Script

 This will get you past the details of writing and launching a simple script.

1. Choose a text editor you want to use. It can be a command-line editor like emacs, pico or vi, or an X Windows editor if you have this option.( We are going to use VI editor)

2. Run your choice of editor and type the following lines:

#!/bin/bash

echo "Hello, world." # This is a comment (comment can be write as)!

**NOTE:** *Be sure to place a linefeed at the end of your script.* Forgetting a terminating linefeed is a common beginner's error.

3. Save the file in the current working directory as "myscript.sh".

4. Move from the text editor to a command shell.

5. From the command shell, type this:

$ chmod +x myscript.sh

Or

$ chmod 755 myscript.sh

6. To execute the script, type this:

$ ./myscript.sh

**Output of the program**

Hello, world.

**How echo works??**

Use echo command to display text or value of variable.

echo [options] [string, variables...] Displays text or variables value on screen. Options

-n Do not output the trailing new line.

**Working with variable**

*Variables* - Symbolic names for a chunk of memory, to which we can assign values, read and manipulate its contents.

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:

1. **System variables** - Created and maintained by Linux itself. This type of variable defined in

CAPITAL LETTERS.

$ echo $USERNAME

$ echo $HOME

2. **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

Example

#!/bin/sh mymessage="Hello World" echo $mymessage

**Variable Naming rules**

1. Variable name must begin with Alphanumeric character or underscore character (\_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows

HOME SYSTEM\_VERSION vech

no

2. Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error

$ no=10

But there will be problem for any of the following variable declaration:

$ no =10

$ no= 10

$ no = 10

3. Variables are case-sensitive, just like filename in Linux. For e.g.

$ no=10

$ No=11

$ NO=20

$ nO=2

Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following

$ echo $no # will print 10 but not 20

$ echo $No# will print 11 but not 20

$ echo $nO# will print 2 but not 20

4. You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

$ vech=

$ vech=""

Try to print it's value by issuing following command

$ echo $vech

Nothing will be shown because variable has no value i.e. NULL variable.

5. Do not use ?,\* etc, to name your variable names.

**Example**

**$ echo "Today is `date`".**

It will print today's date as, Today is Tue Jan ....,Can you see that the `date` statement uses back quote?

**Note: `**Back quote` - To execute command

**Taking data at run time and printing it.**

Example:

#!/bin/sh

echo What is your name?

read my\_name

echo "Hello $my\_name - hope you're well."

**Shell Arithmetic**

Use to perform arithmetic operations. Syntax:

expr op1 math-operator op2

Examples:

$ expr 1 + 3

$ expr 2 - 1

$ expr 10 / 2

$ expr 20 % 3

$ expr 10 \\* 3

$ echo `expr 6 + 3`

**Note:**

expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2. expr 10 \\* 3 - Multiplication use \\* and not \* since its wild card.