# An introduction to Linux

Jayanti Prasad
*National Centre for Radio Astronomy Pune (India)*

November 6, 2009

In this tutorial I will discuss some of the most commonly used Linux commands which are in general used for the following tasks:

- To manage various resources like CPU, RAM, Hard disk etc.
- To manage files and directories i.e., create, delete, move etc.
- To manipulate ASCII data files using editors like **vi, pico, xemacs** and scripting languages like **sed** and **awk**.
- To login a computer remotely.

In the end of the tutorial I will also include a test which have different levels of difficulties i.e., Level I and Level II.

## 1 Linux Commands

### 1.1 Linux shells

In general, on a Linux system tasks are accomplished by typing **commands** on a **shell** and the shell translates those commands into the machine language. On a Linux system there are more than one type of shells. Some of the common shells are the **bash**, **csh** and **tcsh**. Apart from interpretation commands, shells have their own languages also (called shell scripts) which can be used to accomplish various type of complex tasks related to system management etc. In order to see which shell we are using, we can type

```
> echo $SHELL
```

A shell remembers all the commands it interprets and allows short-cuts also. For example, if we type **acro** on a bash shell and press the TAB key then the shell automatically completes the command which starts from **acro**. If there is just one command which starts from **acro** otherwise it gives a list of all commands which start from **acro**. Note that we can also configure the look and feel of a shell according to our taste. If our default shell is **bash** and we want to use **csh** shell, we use the following command

```
> csh
```

When we type a command, the shell looks for that command (executable) in some familiar places which are shown by

```
>  echo $PATH
```

If the command is not found in any of these places than the error message is returned. If we want to add any other command (executable) in the list of the commands which a shell can interpret, we can add the path of that command in the existing path

```
 > PATH=$PATH:PATHOFCOMMAND
```

where **PATHOFCOMMAND** is the path of that command.

Note that the executable for all the commands which are available to a user on a Linux system "stay" in the directory **/bin** which can be seen by the command

```
>  ls /bin
```

For every command there is a man page and a help page available which can be seen by the following commands

```
> man command
```

and

```
> command --help
```

## 1.2 Login, password and IP address

In general, a **username** and **password** is given to every user on a Linux system. If the user knows the **IP address** also (which acts as an "identity") of the computer, the user can login into that computer remotely also and can exchange files with the remote computer if they are connected over the network. In order to remotely login a machine the following command is used

```
>ssh -X  username@hostname
```

where "hostname" is the IP address or "name" of the remote computer.

In order to transfer files, the following command is used

```
> scp  filename  username@hostname:dir
```

here "dir" (see the next section) is the directory in which the file will be transferred on the remote computer. In order to copy a directory the following command is used

```
> scp -r  direname  username@hostname:dir
```

**scp** command can be used to transfer files or directories from a remote computer to the local computer also using

```
> scp username@hostname:filename .
```

Here dot (.) says that the remote file will be copied in the current directory. In order to check if the remote computer is up and is connected to the network, the following command is used

```
> ping hostname
```

## 1.3 Managing processes and resources

At any point of time many processes are running on a Linux system simultaneously. Some of these processes are necessary for the system to work and the rest are optional. Some of the optional processes are generally started by users. Every process contributes some amount of CPU and RAM use which can be checked by using the following command

```
> top
```

This command gives the information about the user who started the process (owner), the CPU and RAM use , the time for which the process is running etc. In place of **top** the following command also can be used

```
>  ps -ef
```

The second column of the output of the above command gives the PID (process ID) of the process which can be used to terminate the process using

```
>  kill -9  PID
```

If we wish to terminate every process running on the system we can use

```
> kill -9 1
```

In order to see the use of the hard disk (HDD) the following command is used

```
> df -h
```

This gives the free space on various partition of a hard disk. In order to see the disk space taken by a file or directory in mega bytes (MB) the following command is used

```
> du -m filename
```

The total use of the hard disk by a user is given by

```
> du -ms filename
```

In order to see which Linux the computer is running the following command is used

```
> uname --a
```

Some of the important parameters of the Linux setting are shown by

```
> env
```

## 1.4 Making your own command AKA alias

Any executable can be considered as a "command" if it is in in the "path". We can also make an executable a "command" my liking that in a file called ".bashrc" which stays in the home directory. If it does not exit, we can create it. Not only that we can make short form of many commands using ".bashrc". For example, I have an executable named "devnag" in the directory "/data/software/velthuis/bin/linux/i386" and I want to make it as a command then I can write in ".bashrc"

```
alias devnag='/home/jayanti/software/velthuis/bin/linux/i386/devnag'
```

Note that after every change in ".bashrc" we must run the following command, in order for changes to take place in the current session otherwise the changes will be effective only when we reboot the computer.

```
> source .bashrc
```

Apart from the ".bashrc", there is another file called ".bash_profile" which also can be used to put permanently the executable and binaries in the path. All the directories in the path can be known by the following command

```
> echo $PATH
```

Now if we want to add "/data/software/velthuis/bin/linux/i386" in the path we write in ".bash_profile"

```
> PATH=$PATH:/data/software/velthuis/bin/linux/i386
> export LD_LIBRARY_PATH
```

note that after every change ".bash_profile" also has to be "sourced". Note that if we want to link the path of a library only for the current session then we type that on terminal also. For example, we can link the above directory by typing

```
> PATH=$PATH:/data/software/velthuis/bin/linux/i386
> export LD_LIBRARY_PATH
```

on the terminal.

## 1.5 Printing

The command line for printing in Linux system is **lpr**. In order to print a file named "paper.pdf" on a printer names "nplr" we use the following command:

```
> lpr -Pnplr paper.pdf
```

on the terminal.

In order to see if there are other jobs in Que on the printer "nplr" we use

```
> lpq -Pnplr
```

In order to cancel our job on the printer "nplr" we use

```
> lprm  -Pnplr
```

| Directory | Content |
|---|---|
| /bin | Linux commands |
| /usr | various header files, libraries etc |
| /proc | information about the hardware etc |
| /etc | setting and configuration files |

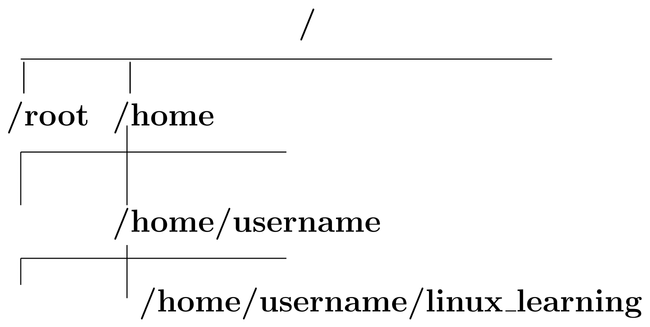

Figure 1: The tree structure of directories in a Linux system

## 1.6 Directories

In a Linux system files are stored in directories which form a tree structure (see Figure 1). On the root (top) of the tree is the **root** directory which is represented by /. Inside the / directory there are other directories. Some of the important directories inside the / directory (like **/use/bin, /usr/local/ /root, /etc, /proc, /sbin** are used by Linux itself for its own files and some can be used by the users. In general every user gets a directory inside the **/home** directory which generally have the name **/home/username**, where "username" is the name of the user. Some of the important directory and the type of content they have are as follows: In order to see the content inside a directory (list) the following command (which I guess is the most commonly used Linux command !) is used

```
> ls
```

This command can be used with many options like **ls -l, ls -a, ls -aR** etc. The name of the present working directory is shown by the following command

```
> pwd
```

Some of the important tasks which are generally done on directories are as follows:

1. Change:

   ```
   > cd  nameofdir
   ```

   This command is used to "go" (called changing) in a directory

2. Create:-

   ```
   > mkdir nameofdir
   ```

3. Delete:-

   ```
   > rmdir nameofdir
   ```

   when the directory is not empty we use "force delete"

   ```
   > rm -rf nameofdir
   ```

4. Move:-

   ```
   > mv oldname newname
   ```

   Note that this command can move the directory "oldname" inside the directory "newname" if that exit, otherwise it will rename directory "oldirname' as "newname".

5. Copy :-

   ```
   > cp -r oldirname newdirname
   ```

## 1.7 Permissions

All the files and directories in a Linux system have a "owner" and a set of permissions. The permissions are of the three type: read (r), write (r) and execute (x). In general, the owner of the files of directories has all the three permissions and has authority to transfer the ownership to any other user or give or take any type of permission from other users. The giving and taking operations are represented by "+" and "-" respectively and the read, write and execute permissions are represented by digits 4, 2 and 1 respectively. The owner can change the permission for himself (u), the group (g) or for all (a) or any combination of these.

In order to see what type of permission a file or directory has and who is the owner of the file or directory, the following command is used

```
> ls -l name
```

where "name" is the name of the file or directory for which we want to know. This command gives a table in the form of 8 columns. The first column which is ten characters long, gives information about the type of permissions the item has for various users. If the first character is "d" then the item is a directory. The second, third and fourth characters show the read (r), write (w) and execute (x) permissions for the owner of the file. The rest of six in the groups of three represent the permissions for the members of the group which the user belongs and for all users respectively. The second and third columns show the name of the owner of the file and the name of the group which the user belongs. The fourth column shows the size of items in Bytes which is important to know in order to keep an eye on the disk use by various items.

If we are not happy with the permissions which an item has for various users we can change them also (if we are owner of that item) by using the command **chmod**. If we want to give (+) read and execute permissions of an item to all (a) the users we use the following command:

```
> chmod a+rx name
```

In order to change the ownership of a file or directory **chown** command is used.

| File Extension | Application | Content |
|---|---|---|
| txt,c, cpp, C, f, f90, f99 | vi | vi, pico, nano, xemacs, emacs |
| jpg, png, gif | xv, display, gimp | picture, graphic |
| pdf | acroread, xpdf | document file |
| djvu | djview | scanned document |
| ps | gv | document, draft, figure |
| avi, mpg, mp4, divx, | gmaplayer, realplayer | video |
| mp3, rm | realplayer, xmms | audio file |

Table 1: Editors and viewers for various type of files

## 1.8 Editing and viewing

Files which contain data in ASCII format (plain text or numerical data) can be edited (read, modify etc.) using various editors like **vi**, **pico**, **nano**, **emacs**, **Xemacs** etc. However, the content of these files can be seen using Linux commands also. Some of the common Linux commands to see the content are as follows:

1. **cat :-**

   ```
   > cat filename
   ```

   shows the full content of the file named "filename" without "opening" the file.

2. **head :-** To see just few lines at the top of a file we use

   ```
   > head filename
   ```

3. **tail :-** To see just few lines at the top of a file we use

   ```
   > tail filename
   ```

4. **more, less:-** These command are used to see a small operation of the file at a time.

Apart from ASCII files, a Linux system supports many other type of files including different type of document files (postscript, pdf, djview etc.) graphic files (ps,png,gif,jpeg etc.) and multimedia files (mp3,rm,avi,wmv,mpg etc). In order to "read" different type of files there are dedicated software packages which can be invoked by linux commands or clicking on icons. For some of these files the commands are given in Table 1

## 1.9 Compressing and decompressing files

Sometime it is useful to store files and directories and files in a compressed form which is done by various compressing packages. Some of the common compression packages and the commands for compression and compression are given in Table 2. If we want to protect a directory/file using password (the file can be used if one know the password) then we use

```
> zip -e -r dirname  dirname
```

this will produce a file named "dirname.zip" which can uncompressed only when we the password is known.

| Package | Compression | Decompressing |
|---|---|---|
| zip | zip *itemname* | unzip *itemname.zip* |
| gzip | gzip *itemname* | gunzip *itemname.gz* |
| tar | tar -cvf *tarpackgname.tar* *item1 item2 ...* | tar -xvf *tarpackgname.tar* |
| rar | | unrar -e *itemname.rar* |

Table 2: Commonly used compression packages

### 1.10 TAB, CTRL-A and CTRL-E

These three keys are very useful in working with Linux command line. The TAB key can complete a linux command. In order to see how many commands start with a given set of characters we can type that set of characters and press the TAB key. CRTL-E takes us to the end of command line and TAB-A brings us at the beginning.

## 2 Scripting languages

Sometime we are interested in finding a pattern (word or string of characters) in a text or data file, or replace an old pattern by a new one, or delete all lines in a file containing a given pattern. For accomplishing such tasks there exist scripting languages. In this section I will discuss two of such languages called **sed** and **awk**. These languages are very useful to pull out the interesting chunks of data or replace it by new one from a data file. Not only that, these scripting languages (in particularly awk) can be used to perform mathematical operations also like sorting (in alphabetical or mathematical order), addition, multiplication etc. Since these languages work on "patterns" which can be quite complicated. Moreover it may be more useful to specify the "rules" for the pattern in place of pattern itself. In order to represent patterns there is a language called the "regular expression" (I will use regexp now onwards) which is accepted by many other scripting languages also apart from **sed** and **awk**. Some of the rule for regexp are as follows:

- A regexp is a string of all type of characters.
- A regexp must start and end with a delimiter. The most common delimiter is /.
- If we want to use more than one options for a part of regexp then we can put these options inside a []. For example, if we are searching for a fruit name in a file then we can put the names of all fruits in a []. Not only that, we can also specify the range using []. For example, we are looking for any single digit number between 0 and 9 we can use [0-9]. The same can be used for alpha numerical characters also i.e., [a-z].
- If we accept any character at some position of a part of regexp then we use a single dot (.) at that position. For example, /dog./ matches with dogs, doga, dog1, etc.
- If we want the pattern to be in the beginning or end of a line then we use $\wedge$ before the pattern and $ at the end of the pattern respectively. Note that the regexp /$\wedge$$/ match with every line.
- All the special characters in regexp should start with a \.

| **Command** | **Example** | **Comment** |
|---|---|---|
| `ls` | `ls` | shows list of items in a directory |
| `ls -a` | `ls -a` | list hidden items (.) also |
| `ls -l` | `ls -l` | list more information about items |
| `ls -aR` | `ls -aR` | list all items in a directory tree |
| `mkdir` | `mkdir` *newdir* | make a new directory |
| `cd` | `cd` *newdir* | change directory |
| `rmdir` | `rmdir` *newdir* | remove an empty directory |
| `rm` | `rm` *filename* | remove (delete) a file |
| `rm -rf` | `rm -rf` *filename* | forces remove a file/directory |
| `cp` | `cp` *file1 file2* | copy file1 into file2 |
| `mv` | `mv file1 file2` | rename/move a file/directory |
| `top` | `top` | show the CPU load |
| `ps -ef` | `ps -ef` | list of processes |
| `du -m` | `du -m` *name* | disk use in MB by *name* |
| `clear` | `clear` | clears the terminal/xterm |
| `chmod` | `chmod 700` *filename* | change the permissions |
| `ssh` | `ssh -X` *user@computer* | connect to a remote computer |
| `uname` | `uname --all` | information about architecture & OS |
| `ping` | `ping` *hostname* | status of network connectivity |
| `pwd` | `pwd` | present working directory |
| `find -iname` | `find -iname` *filename* | finds the file filename is |
| `wc` | `wc` *filename* | gives how many lines are in the file *filename* |
| `man` | `man` *command* | shows information about the *command* |
| `help` | `help` *command* | shows help about the *command* |
| `diff` | `diff` *file1 file2* | compares the content of *file 1*& *file2* |
| `cat` | `cat` *file 1 file2* > *file3* | put the content of first two files in third |
| `spell` | `spell` *filename* | shows spelling mistakes in *filename* |
| `sort` | `sort` *filename* | put the entries of file *filename* alphabetically |
| > | *command* > *filename* | redirect the output of *command* in *filename* |
| `g77` | `g77` *file.f* | compiles a fortran 77 file |
| `[f90,ifort,ifc]` | `[f90,ifort,ifc]` *file.f90* | compiles a fortran 90 file |
| `gcc` | `gcc` *file.c* | compiles a c file |
| `c++` | `c++` *file.cpp* | compiles a c++ file |
| `latex` | `latex` *file.tex* | compiles a tex file |
| `exit` | `exit` | close a xterm |

Table 3: Summary of Linux Commands

| Regular Expression | Comments |
|---|---|
| . | matches any one character |
| ⋆ | means times, for example .* will match anything any times |
| + | something followed by + will match any number of repetitions of that thing |
| ? | means that the preceding item is optional |
| \d+ | is equivalent [0-9] |
| \s | white space |
| \s* | any amount white space |
| ∧ | at the start of a line |
| $ | at the end of a line |
| \s+ | any amount of white space |

Table 4: Regular expressions

Some of the rules for a regular expression are given in the Table 4

Before discussing **sed** and **awk** tools it is useful to introduce the concept of **piping** and **directing**.

## 2.1 Pipe

We can write the output of any Linux command or executable in a file using > :

```
> linuxcommand   [parameters] >  output.txt
```

For example

```
> ls -l >   output.txt
```

will write the output of the command **ls -l** in a file named "output.txt".

Pipe (|) is used to combine more than one Linux commands i.e., to type more than one command on the terminal which will be executed one after another (piping). This command is used in the following way:

```
> linuxcommand1  | linuxcommand2
```

For example

```
> ls -l   |  more
```

By the way I have not told that in Linux "*" is used as a wild card. For example if we want to do **ls -l** on all files starting with "big" we use

```
> ls -l big*
```

Before starting **sed** and **awk** let me introduce **grep** which also can be used for pattern matching.

## 2.2 grep

**grep** is a useful to find a given pattern of characters in a file or to find files which have a given pattern. For example the command

```
> grep    "pattern"  filename
```

prints all the lines of a file named *filename* which contains the pattern ”pattern”. The command

```
> grep  "pattern"  filename1 filename2 filename3
```

will print the lines from many files which contain the given pattern.

## 2.3 sed

**sed** is used to find, substitute or delete a pattern (which is written in the form of regular expressions) from a file or output from a Linux command.

- **Search a pattern:-** In order to find a pattern in a file we use the following command:

  ```
  > sed -n  '/myword/p' myfile.txt
  ```

  This prints all the lines of a file named ”myfile.txt” which contains the word ”myword”. We can also specify the pattern by rules following the syntax of regular expressions (see Table 4.

- **Replace a pattern:-** We can substitute one pattern in a file by another using the following:

  ```
  > sed -n  's/oldpattern/newpattern/g' myfile.txt
  ```

- **Delete :-** We can delete all the lines from a file which contain given pattern

  ```
  > sed  '/myword/d' myfile.txt
  ```

  This removes all the lines which contain the word “myword” from the file “myfile.txt”.

Note that in place of backslash we can use other delimiters also. For example

```
>  sed s:oldword:newword:  <oldfile.txt> newfile.txt
```

A nice introduction of sed is given at the following website :
http://www.grymoire.com/Unix/Sed.html

## 2.4 awk

**awk** can be used to manipulates files and print the output of linux command in a useful ways. I found **awk** useful for the following problems:

- To print a given columns of data files. For example the following command:

  ```
  >  awk '{print $1}'   filename.txt'
  ```

print the first column of the file named "filename.txt" on the screen. We can print any column using the number of that column after the $ sign. It is interesting to see that this command can be used in a c-programming style also. For example the above command also can be written in the following way:

```
> awk '{printf("%$ \",$1)}'}  filename.txt'
```

What is interesting is that **awk** accepts full pieces of c-programs also. For example, if we want to print all the number between 1 to 30 we use the following command:

```
> awk 'BEGIN{for(i=0; i < 30; i++) printf("%d\n",i)}'}
```

This piece is very useful to create quickly tables.

In **awk** rows are indexed by a variable "NR". For example if we want to put line number before every line of file named "filename.txt" we use the following command:

A nice introduction of awk is given at the following website :
http://www.gnu.org/software/gawk/manual/gawk.html

## 3 Exercise: Level I

1. Find out what are the processes running on your computer and what amount of CPU and RAM they are consuming. Kill a given process.
2. Find out the information about the CPU and RAM your computer has.
3. Find out what amount of disk space is there on your computer and what amount of disk space your various files and directories are consuming.
4. Find out if a given computer is connected to network and if yes then transfer a file from that computer to your computer.
5. Change the permissions of a given file or directory.
6. Find out which Linux is running on your computer.
7. Show the content of an ASCII file in different ways e.g., full content, top, end, partial etc.
8. Find the location of a given file on your computer.
9. Edit an ASCII file using **vi, emacs, pico** etc.
10. Find out the number of lines in a file, sort a file and compare two files.
11. To direct the out of command in a file.
12. Use two or more commands at a time using pipe.

## 4 Exercise : Level II

1. Print all the lines of a file which contain a given pattern using **grep** and **sed**.
2. Substitute a given pattern by a new pattern in a file using **sed**
3. Delete a line with a given pattern from a file using **vi** or **sed**.
4. Print the given columns of a data file using **awk**.
5. Produce a data file for a given function using **awk**.
6. Interchange the rows and columns of a file using **awk**.
7. Use the sed and awk command together using pipe.
8. Sort the data of a given file using **awk**.

## 5 Reading assignment

- Read the man and help pages of the commands you have used.
- Read about the resource requirements of Linux and Windows.
- Explore the history of linux and people behind it.