**linux command with examples**

[December 11, 2012](https://codedecoder.wordpress.com/2012/12/11/linux-command-example-ubuntu/) by [arunyadav4u](https://codedecoder.wordpress.com/author/arunyadav4u/) [19 Comments](https://codedecoder.wordpress.com/2012/12/11/linux-command-example-ubuntu/#comments)

I am using Ubuntu for last 5 years and doing a lot of things from command line. Many of the day to day command is now on my tip, but many of more I have used is out of my mind, and I have to google it back, So I have decided to note it down here for my future reference. I will keep adding to it whenever I get time or use new command.

**Note :**

Linux command can take a no of options. These options can be passed in both shorthand form or expanded form. with shorthand form use single hyphen – and with expanded form use double hyphen –  -, say you can get help option to a command as -h or – -h

Some options do not need any argument while other need some argument to be passed with them. you can pass all the options separately, but it is better to pass argument less options grouped together and passing option with argument separately. Say,  **$ mkdir -pv -m 777 myfolder** is more concise then **$ mkdir -p -v -m 777 myfolder**. In the first case we have grouped **-p -v** together as **-pv**. Since -m option need further argument here we have passes 777, we have kept it separate from -pv

**|** , known as pipe is very important for filtering result when used with grep. | basically, pass the result of first function as a block to the next function. Example,  **ls | grep myfile** will list only those files and folder which have myfile in there name. Here ls will return list of all file and folders which will passed to grep, which will do the filtering.

**1 ->** **history** # will show you all the command you have typed in past

**Syntax :** history #it do not take any argument

Example:

**$ history** # will list all comand used in past

**$ history | grep netbeans** # since we have piped the result of history to grep it will return only those command , which contain word history in them

**2 ->** **man** #show complete detail of a command

**syntax :** man command\_name

It is basically short form of manual and will tell you detail about other commands you pass to it as argument. It will explain you what that particular command do, how it work, what options it will take and so on. **Basically, if you remember name of any command, you can get all detail of it with man command.**

Example

**$ man man** # it will tell about how to use man itself

**$ man sudo** # will tell you all about sudo command

**$ man dpkg** # will tell you all about dpkg command

3 -> **help** #show tha available options for a command

**Syntax:** command\_name – -help or command\_name -h

unlike man , it will just tell you about different options a command can take. Since, most of the time we are most concerned about the available options a command can take rather then the whole detail, It is used more often then the man command. Also, most of the software , say netbeans comes with help option which will help you on how to execute them. On other hand man will work only with ubuntu command.

NOTE : If you know name of a command, you can get all available options with **-h** or **– -help** option. Remember that for some command **-h** not work but you have to pass **– -help**. for example:  **rm -h** will not work, you have to use **rm – -help** to know about rm options

Example:

**$ cd /netbeans-6.9.1/bin** #move to netbeans bin directory

**$ ./netbeans – -help** # you want to start netbeans and want to know available options. It has listed down all options for you  
General options:  
  –help                          show this help   
  –jdkhome <path>      path to Java(TM) 2 SDK, Standard Edition  
  -J<jvm\_option>        pass <jvm\_option> to JVM

  –cp:p <classpath>    prepend <classpath> to classpath  
  –cp:a <classpath>    append <classpath> to classpath

**4  -> ls** # Listing files and folders

**Syntax :** ls [options] #options are optional, if you do not pass it will take the default, you can see complete list of options

with ls – -help

Examples:

**$ ls** #will list all folder and files but not hidden files or folder

**$ ls -a** # will list hidden files and folders also

**$ ls -B** #will not list the backup file ending with ~

**$ ls -l** # will do complete listing with all details like owner, date, permission etc. but will not show hidden file

**$ ls -l -a -B** #will do all the above

**5 ->** **mkdir** # creating directory

**Syntax:** mkdir [options] directory\_name #options are optional, if you do not pass it will take the default, you can see list of options with mkdir – -help

Example:

**$ mkdir myfolder**

**$ mkdir -p myfolder1/myfolder2/myfolder3** # -p option tell to create nested directory with parent child relation. Thus myfolder3 will be created in myfolder2 and so on

**$ mkdir -p myfolder**# the p option means parent , if the directory already exist do not throw error but use the existing directory as parent

**$ mkdir -pv -m 777 myfolder** # it will not complain if directory exist, set the permission to 777 and also print the task it performing on terminal

**6 -> cd** #changing directory

**Syntax :** cd directory\_name # you can pass directory\_name or .. or ~ to cd

Examples:

**$ cd myfolder** # take you inside myfolder

**$ cd .. #** take you out from current folder to the previous folder

**$ cd ~** # take you out of every previous folder to the root of terminal

**7-> touch** # it will create a file if not present and if already present will modify its modification or access time depending on argument you pass to it, but not overwrite its content

**Syntax :** touch [options] file\_name # touch –help will show you all available option. If you do not pass any option you can create multiple file with single touch command

Examples :

$ touch test1 #will create test1 file

$ touch test1 test2 test3 test4 # will create test2 test3 test4 as new file. Since test1 is already there it will just modify its access time

$ touch -a test1 # will change access time of test1 file

$ touch -m test1 # will change modification time of file test1

$ touch -am test1 # will change both access as well as the modification time

$ touch -d '1 May 2005 10:22' test1 # with d option you can set any date you want but maintain the formate of string, say '1 05 2005 10:22' otherwise '2005 1 May 10:22' will give error. however, you can ignore time if you want

$ touch -d '1 May 2005' test1 # it will set only the date, if you do not pass year then it will take current year i,e '1 May' will also work

$ touch -r test2 test1 # -r is reference option i,e it will set datetime of file test2 to test1 file

**8 ->** **rm** # removing file or folder, by default it remove only files to remove a folder, you need to pass additional options

**Syntax:** rm [options] file\_or\_directory\_name #options are optional, if you do not pass it will take the default, you can see list of options with rm – -help

Example :

**$ rm my\_file** # will remove my\_file

**$ rm -f my\_file** # will not throw error if my\_file do not exist i,e it will ignore the removal

**$ rm -fi my\_file** # Will ignore if file is not there and ask you to press Y to delete the file i,e it will become interactive

**$ rm -r my\_folder** # The -r option must be passed to delete a folder other while it will not delete it

**$ rm -rf my\_folder** # will delete the folder and will not complain if the folder do not exist

**9 -> rmdir** #it will list empty directory

**Syntax :** rmdir [options] directory\_name # without argument it will delete all empty directory name passed to it. It will throw error if directory is not empty. for that case use rm as above

Examples:

**$ rmdir dir1 dir2 dir3** # it will delete the empty directory dir1 dir2 dir3

**$ rmdir -p dir1/dir2/dir3/dir4** # It will delete the parent also if deleting the child make it empty, so first dir4 will be deleted then dir3 then dir2 and so on

**10 -> mv** # It will move file or directory to other location. if the location of source destination is same the file or directory will be get renamed

**Syntax:** mv [options] source destination # If two files are provided as source and destination source will be renamed as destination. If you pass more then two argument and the last one is directory all the previous will be treated as source and will be moved into the last directory, **but if the last argument is not directory it will throw error.**

Examples :

**$ mv myfile.txt myfile.rb** # it will rename myfile.txt to myfile.rb i,e the name is changed.

**$ mv -bvi myfile.txt myfile.rb** # here we are passing some option b will backup the file before renaming, i will ask your yes no before renaming and v will print what going on the console

**$ mv test1 test\_folder myfile.rb /home/arun/my\_folder** # it will move the file test , myfile.rb and the test\_folder to /home/arun/my\_folder

**$ mv \*  /home/arun/my\_folder** # it will move everything in current folder to /home/arun/my\_folder

**11-> grep** # It is used to search a pattern specified by user in the given text or files or block passed from first command to it through the pipe symbol |

**Syntax:** grep [options] pattern [files] # the complete list of options can be accessed with grep – -help command. If you want to match more than one word pass  within ” “

**$ grep “Linux is good” file1 file2 file3** # It will list all lines of these three file containing “Linux is good” in it in separate line, each line will be preceded with name of the file in which they appear. If you do not want the file name you can use -h option as below

**$ grep -h “Linux is good” file1 file2 file3** # It will not precede the matched line with the file in which they are found

**$ grep -i “Linux is good” file1 file2 file3** # it tell to ignore the case i,e “linux is good” will also match

**$ grep “Linux is good” \*** #will search the text “Linux is good” in all the text file in the current directory

**$ grep -r “Linux is good” \*** #-r option will make it to search the text “Linux is good” in all the text file recursively in the current directory as well as all it child directory

**$ grep -c “Linux is good” file1 file2** # It will not print each line but just return the count of match in each file

**$ grep -n “Linux is good” file1 file2** # It will also print the line number of the line containing the pattern in each file

**$ grep -rni “Linux is good” \*** # will recursively search each directory, print the line no and will ignore the case

**$ ls -l | grep “foo”** # it will list only those file and directory which containn the word foo

**$ history | grep rails** # will list history of only those command which contain the word rails

**$ history | grep -in rails** #will ignore case,  and print line no.

**12 -> less** # can be used to see content of a file .It allow you to display a large data in block and let you to scroll down, basically, it keep the screen at first line and let you scroll down. If you do not use this, screen will set to the last line and you have to scroll up to see the first line. you can exit the content by pressing q .

**Syntax :** less [options] file\_name # you can see complete list of option with less – -help. Though, generally, I used it to see the content of file or a command from top and have not tried any options .

Examples:

**$ less myfile.txt** # It will show content of file from the top, need to scroll down to see more content. press q to exit the content

**$ ps -ef | less** #will show the result of ps -ef command from the top

**13 -> more** # It can be also used to display content of a file on terminal.

**syntax:** more [options] file\_name # By default It do not provide scroll down facility like less to see the whole content but show only that much content which fit into your screen. however, using different options you can set the no of line you want to see, which will activate the scrolling if the set line not come on the screen

Examples:

**$ more myfile** # will show that much content of file which fit in screen

**$ more -100 + 50 myfile** # will show 100 lines of file starting from line 50

**$ more +/”Iam arun” -100** # will show 100 line of the file from the line which contain “Iam arun”

**$ more -s -u -100  +50 myfile** #  s will cause to remove blank line from display and u will remove underlines

**$ ps -ef | more -10 + 20** # will display 10 lines of the result of command ps -ef starting from the 20 line

**14 -> cat** # will be used to create a file, read a file, concenate content of many files

**syntax :** cat [options] [input\_filename] [> or >>] [output\_filename] # all the things in [] is optional i,e cat can work without them also

NOTE :

1 -> If you do not provide output\_filename the output will be printed on terminal itself i,e terminal is the standard output for cat command

2 -> **>** and **>>** both are called append operator i,e write content to a file but > will erase the file of content if exist and write to it but >> will add the new content to the existing content , so it is safer to use **>>** instead of**>**

Example:

**$ cat >> test\_file1** # it will create **test\_file1** and when you press enter it will take you to the next line where you write to it, to exist press enter then press ctrl and d  
I have pressed enter on last line to reach here  
again pressed eneter to come to next line and keep writing  
All content Iam writing will get written to test\_files1                                   
to exist writing we will press Enter followed by Ctrl D

**$ cat test\_file1** # it will show you content of file test\_file1, so you can see the content you have written on the terminal  
I have pressed enter on last line to reach here  
again pressed eneter to come to next line and keep writing  
All content Iam writing will get written to test\_files1  
to exist writing we will press Enter followed by Ctrl D

**arun@arun-yadav:~$ cat -n test\_file1** #the -n option will print the line no also  
  1    I have pressed enter on last line to reach here  
     2    again pressed eneter to come to next line and keep writing  
     3    All content Iam writing will get written to test\_files1  
     4    to exist writing we will press Enter followed by Ctrl D

**$ cat >> test\_file2** # will create a new file  
This is the second file # you added this line to test\_file2

**$ cat test\_file1 test\_file2 >> test\_file3** # will create file test\_file3 and append content of test\_file1 test\_file2 to it

**$ cat test\_file3** # it show the content of test\_file3, you can see that this file contain content of both the files  
I have pressed enter on last line to reach here  
again pressed eneter to come to next line and keep writing  
All content Iam writing will get written to test\_files1  
to exist writing we will press Enter followed by Ctrl D   
This is the second file

**15 -> echo** # you can use it to write a string to file or terminal itself

**syntax:** echo string [filename] # file name is optional if you do not pass it the output will be printed to the terminal

unlike cat, it can use to write single string to a file or terminal i,e you can write multiple lines as with cat command

Example:

**$ echo Iam Arun** # it will print the output to terminal  
Iam Arun  
**$ echo Iam Arun to test\_file >> test\_file** # it will append the string to the test\_file, if test\_file is not present >> operator will create it and the append to it  
**$ cat test\_file** # we will see the content of test\_file  
Iam Arun to test\_file  
**$ echo Iam adding one more line >> test\_file** # one more line is appended  
**$ cat test\_file** # you can see both the line  
Iam Arun to test\_file  
Iam adding one more line  
**$ echo this “>” operator is dangerous as it will erase and write > test\_file** # see the danger of using >, if the file exist it will erase its content before appending  
**arun@arun-yadav:~$ cat test\_file**  
this > operator is dangerous as it will erase and write # see earlier line is erased and only single line present

**16 -> pstree** # will return all the running process in tree structure with there name and process id

**Syntax :** pstree [options] [pid or username] # you can see the  complete list of option with  pstree – -help. the argument is user or process\_id whose tree you want to see

Example:

**$ pstree** #will list all the running process with all the user

**$ pstree root** # will show all the process running with root user

**$ pstree arun** # will list all the process running with the user arun

**$ pstree -p arun** # will print the PID i,e process id also with the name

**$ pstree -pn**  # will print the PID and also in the sorted order

**$ pstree -pnh** # the h option will highlight the current running process

**$ pstree -pnhu** # the u option will also print the user owning that process

**$ pstree -pnhul** # l option will not allow to truncate the process detail if it is too long

**17 -> pgrep** # It will return PID of the process

**Syntax :** pgrep process\_name # it will return the PID of process\_name. It will be helpful if you know the process name otherwise use pstree or ps to find PID

Examples:

**$ pgrep firefox** # will return PID of firefox

**$ pgrep ruby** # will return PID of ruby

**$ pgrep skype** # will return PID of skype

**18 -> ps** # it means process status and return all the running process with name and PID

**Syntax :**  ps [options] # you can see all the available options with ps – -help

Examples :

**$ ps** # will return currently active process

**$ ps -e** # will return all the running process

**$ ps -ef** # will return all running process with full detail

**$ ps -ef | less** # as process list is very long, we have piped the output of ps -ef to less command. It help you to display result in small blocks and let you to scroll down to see more.

**NOTE** : IF you want to kill some process through script, you will jut need the PID to pass it to kill command. say to kill apache 2 from script you can use below:

**kill -9 $(ps -e | grep apache2 | awk '{print $1}')**

**19 -> pkill** # will kill the process passed to it

**Syntax:** pkill process\_name # It would be helpful only if you know the name of process, you want to kill

Examples:

**$ pkill ruby**

**$ pkill firefox**

**20 -> kill** # will kill the process whose PID is passed to it

**syntax :** kill [signals] PID # there is 60 type of signal which can be passed to kill, but you have to just remember **9** which is called **SIGKILL**, the default signal passed is **15** which is called **SIGTERM**. Actually, kill do not kill the process itself instead, it just pass proper signal to a process which cause it to terminate i,e logic of termination is in every process itself. Say, for example when you close firefox browser, signal 9 get passed to the firefox process by our OS and it get terminated. The kill command just give you a way to pass signal to a process from the terminal.Thus by passing proper signal with kill you can control any state of a process. But mostly that is not needed. Mostly it is use to close any hanged process. If any process get hanged, you get its PID from the above pstree or pgrep or ps command and the pass it to kill to terminate that hanged process.

**NOTE: the complete list of signal available for kill can be listed with kill -l**

Examples:

**$ kill 485** #will terminate the process with PID 485. here the default signa 15 get passed, there may be chance that it fail to kill the process, say if it is making some system call

**$ kill -9 485** # it will be guaranteed that the process will be called. If the process is making any system call, it will wait and kill it when it return from the call

**21 -> killall** #will kill the process along with its child whose name passed to it. helpful if you know the process name

**syntax:** killall [options] process\_name #will kill the process and all its child

NOTE: **Killing by file name only works for executables  (i.e., runnable programs) that are kept open during execution**

Examples:

**$ killall nautilus** #will kill the nautilus process

**$ killall -e nautilus** # it will tell to kill whose name exactly match with nautilus. It will be helpful if your process name is more that 15 character, since killall by default match only upto that, so will delete all other process also whose first 15 character matches. the -e option will prevent this.

**$ killall -ew nautilus** # the w option will tell to wait till all the process are killed, otherwise killall check it every second by default and resend the signal if found some is still running, but since some has already get  killed, some signal become unresponsive. Also killall can kill all the process, including other killall process, except itself

**22 -> pwd** # will print the current working directory

**23 -> clear** # It will remove all previous command and output from the console

**24->SCP** # You can take it as server copy. scp command is used to copy files from one server to other.It can be from your local machine to some  other machine or vice versa. You can also use it to copy file between two different machine from your system.

Below will be the syntex :

**CASE 1:** Copy file from from your system to some other system:  
**scp  /path/to/local/file  username@hostname:/path/to/remote/file**

**CASE 2:** Copy file from another system to your system:

**scp  username@hostname:/path/to/remote/file  /path/to/local/file**

**CASE 3:** Copy file from some system to some other system from your machine:

**scp  username1@hostname1:/path/to/file  username2@hostname2:/path/to/other/file**

NOTE : to copy a **directory** instead of **file** in any of the above case use **scp -r** instead of just **scp**

**NOTE :** username@hostname is basically the url to which you ssh

**24-> lscpu**

This will tell you information about your CPU. Like you may want to know, you are **having 32 bit or 64 bit processor** (remember many a time at the time of download you do not know you should download 32 or 64 bit version).

Example :

**$ lscpu**  
Architecture:                          i686  
CPU op-mode(s):                   32-bit, 64-bit  
Byte Order:                           Little Endian  
CPU(s):                                   4  
On-line CPU(s) list:              0-3  
Thread(s) per core:              2  
Core(s) per socket:               2  
Socket(s):                              1  
Vendor ID:                          GenuineIntel  
CPU family:                         6  
Model:                                 42  
Stepping:                             7  
CPU MHz:                           3092.905  
BogoMIPS:                         6185.90  
Virtualization:                     VT-x  
L1d cache:                           32K  
L1i cache:                            32K  
L2 cache:                             256K  
L3 cache:                             3072K

Architecture showing to be i686(X86, i686, or i386 is same), which means your OS is of 32 bit. But the CPU op-mode showing that it can work in both 32 and 62 bit mod i,e your CPU is of 64 bit. So if you want you can install 64 bit Ubuntu OS on it, but if the mode is only 32 bit, you can’t

**NOTE :**

**X86, i686, or i386      means you are running a 32 bit kernel i,e OS.**  
**X86\_64 , amd64 , or X64 means you are running a 64 bit kernel i,e OS.**

**24 -> netstat**

It display information about all the **running internet connection be default**. if you pass **-a** option it will display inactive connection also. It take a number of options. You can find all the available option with below command:

**# netstat -h**

        -r, –route              display routing table  
        -i, –interfaces         display interface table  
        -g, –groups             display multicast group memberships  
        -s, –statistics         display networking statistics (like SNMP)  
        -M, –masquerade         display masqueraded connections

        -v, –verbose            be verbose  
        -W, –wide               don’t truncate IP addresses  
        -n, –numeric            don’t resolve names  
        -N, –symbolic           resolve hardware names  
        -e, –extend             display other/more information  
        -p, –programs           display PID/Program name for sockets  
        -c, –continuous         continuous listing

        -l, –listening          display listening server sockets  
        **-a, –all, –listening   display all sockets (default: connected)**  
        -o, –timers             display timers  
        -F, –fib                display Forwarding Information Base (default)  
        -C, –cache              display routing cache instead of FIB

You can use any of the above options

**Example:**

**root@arun-desktop:~# netstat -ltnp**  
Active Internet connections (only servers)  
Proto     Recv-Q     Send-Q         Local Address           Foreign Address     State       PID/Program name  
tcp        0                        0            127.0.0.1:631           0.0.0.0:\*               LISTEN      2322/cupsd        
tcp        0                        0            0.0.0.0:64283           0.0.0.0:\*               LISTEN      3176/skype        
tcp        0                        0            0.0.0.0:902             0.0.0.0:\*                  LISTEN      1135/vmware-authdla  
tcp        0                        0            127.0.1.1:53            0.0.0.0:\*               LISTEN      1460/dnsmasq      
tcp6       0                       0            ::1:631                         :::\*                    LISTEN      2322/cupsd        
tcp6       0                       0            :::443                            :::\*                    LISTEN      10916/apache2     
tcp6       0                       0            :::8140                          :::\*                    LISTEN      10916/apache2     
tcp6       0                       0            :::80                               :::\*                    LISTEN      10916/apache2

**25-> browsing folder as root user.**

sometime you want to open folder as root user, so that you can directly browse and edit a file. For example, say you want to edit standalone.xml file of jboss . If you go to that file through the UI, open it and edit it, you can’t as you have browsed it as normal user so don’t have permission to edit it. So in such case you can browse the file as root user with below command.

**root@arun-desktop:~# nautilus**

**arun@arun-desktop:~$ sudo nautilus** # for non root user

**26->Complete Listing**

Complete list of Linux command with detailed explanation is available [here](http://linux.about.com/od/commands/l/blcmdl.htm).

Reference :

<http://www.linfo.org/pipes.html>