Unit 2

Processes In Linux:

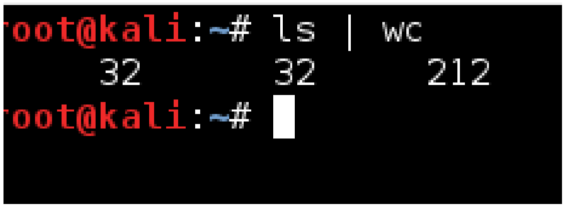
A RUNNING INSTANCE OF A PROGRAM IS CALLED A PROCESS. If you have two terminal windows showing on your screen, then you are probably running the same terminal program twice—you have two terminal processes. Each terminal window is probably running a shell; each running shell is another process. When you invoke a command from a shell, the corresponding program is executed in a new process; the shell process resumes when that process completes.

Process IDs
Each process in a Linux system is identified by its unique process ID, sometimes referred to as pid. Process IDs are 16-bit numbers that are assigned sequentially by Linux as new processes are created.

Connecting Processes with pipe:

To connect 2 processes or more correctly to pass the output of one command/ process as the input to another command pipes are used. We can simply write 2 commands next to each other for this purpose just by putting a | in between the commands.

There is no restriction on the number of commands you can use in pipeline. But you must consider the behaviour of the commands before placing them in a sequence.

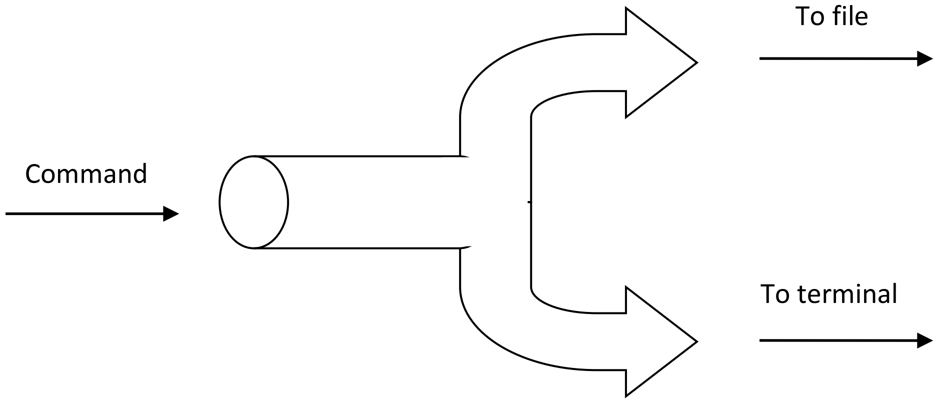


```
oot@kali:~# ls | wc
     32      32     212
oot@kali:~# 
```

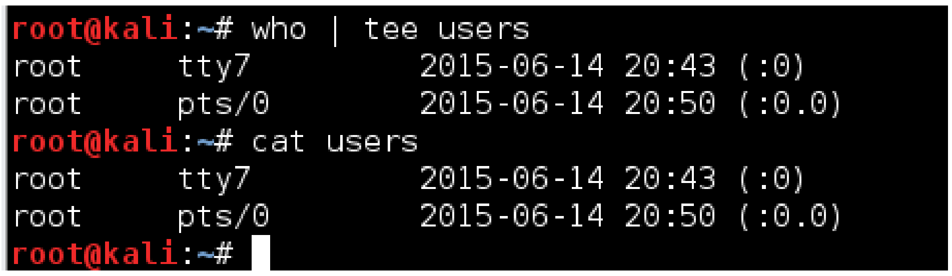
tee command:

The **tee** command is named after the T-splitter in plumbing, which splits water into two directions and is shaped like an uppercase T.

There are instances while working on linux when you want to save the output of a command to a file as well as see the output on the screen, this is achieved by using tee command.

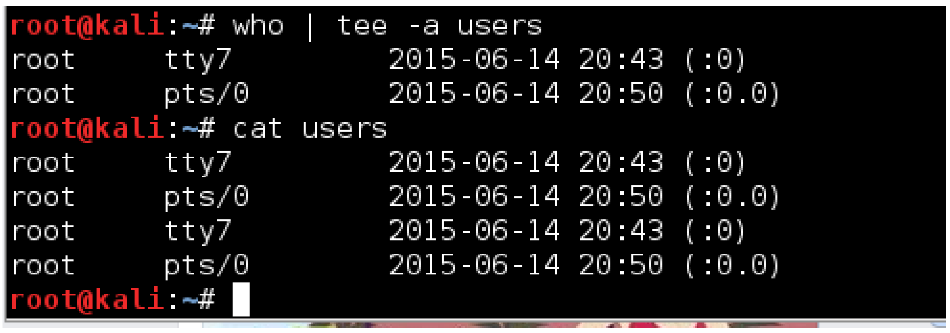


Example:



```
root@kali:~# who | tee users
root      tty7         2015-06-14 20:43 (:0)
root      pts/0        2015-06-14 20:50 (:0.0)
root@kali:~# cat users
root      tty7         2015-06-14 20:43 (:0)
root      pts/0        2015-06-14 20:50 (:0.0)
root@kali:~#
```

-a or –append option : This option is used to append to the file and not overwrite them.
Example:



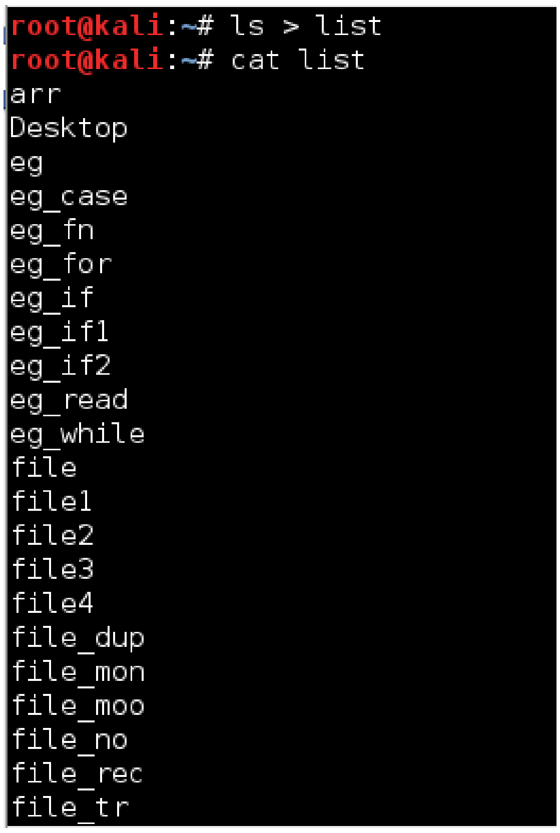
```
root@kali:~# who | tee -a users
root      tty7         2015-06-14 20:43 (:0)
root      pts/0        2015-06-14 20:50 (:0.0)
root@kali:~# cat users
root      tty7         2015-06-14 20:43 (:0)
root      pts/0        2015-06-14 20:50 (:0.0)
root      tty7         2015-06-14 20:43 (:0)
root      pts/0        2015-06-14 20:50 (:0.0)
root@kali:~#
```

Redirecting Input / output:

Standard Output

Most command line programs that display their results do so by sending their results to a facility called *standard output*. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used.

Example:

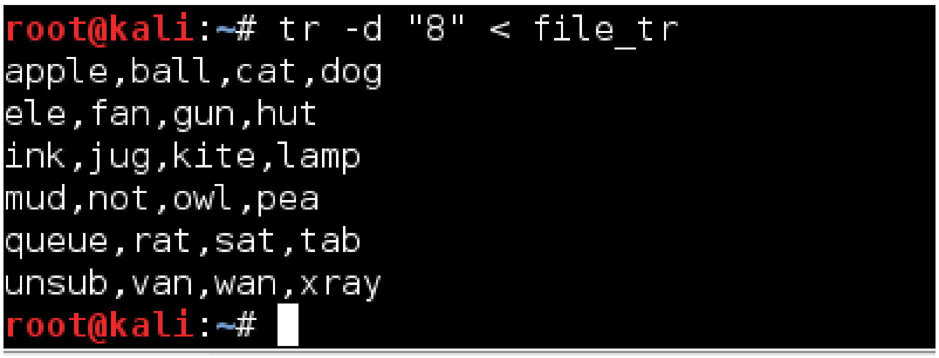


```
root@kali:~# ls > list
root@kali:~# cat list
arr
Desktop
eg
eg_case
eg_fn
eg_for
eg_if
eg_if1
eg_if2
eg_read
eg_while
file
file1
file2
file3
file4
file_dup
file_mon
file_moo
file_no
file_rec
file_tr
```

Standard Input

Many commands can accept input from a facility called *standard input*. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used.

Example:



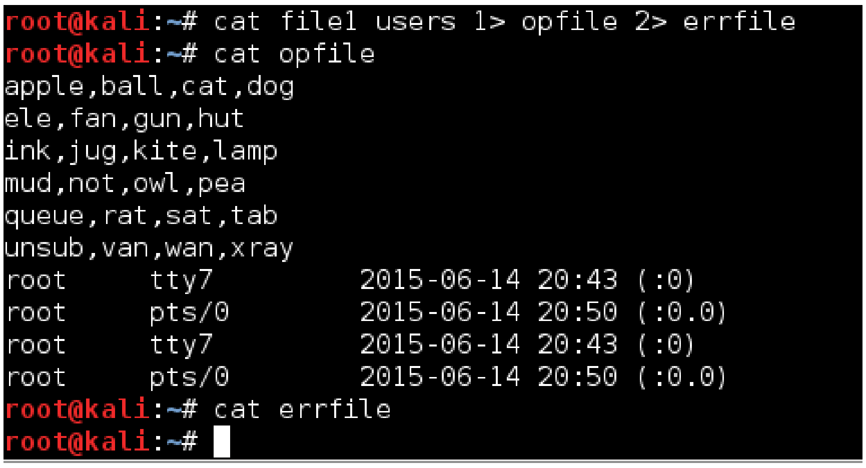
```
root@kali:~# tr -d "8" < file_tr
apple,ball,cat,dog
ele,fan,gun,hut
ink,jug,kite,lamp
mud,not,owl,pea
queue,rat,sat,tab
unsub,van,wan,xray
root@kali:~# 
```

Diagnostic output

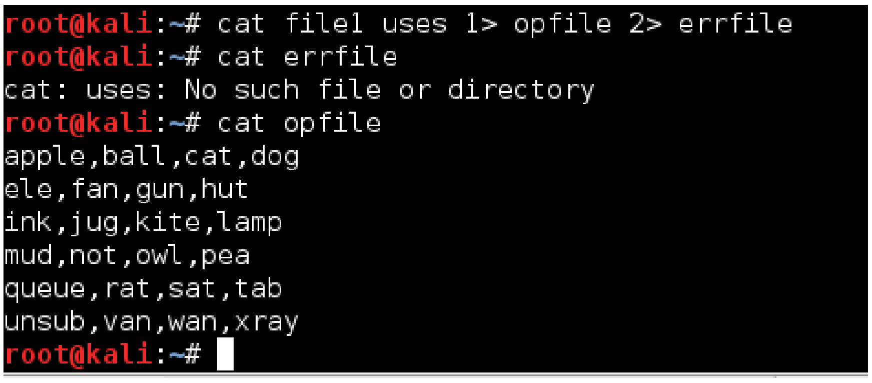
There are times when something goes wrong with the commands and an error is produced and sometimes the error is even not displayed on the terminal screen. To save the diagnostic output of a command we may redirect it into a file by using file descriptors. A file descriptor is an abstract indicator used to access a file or other input/output resource. There are three types of file descriptors given as follows:

- **0 (zero):** file descriptor for the standard input.
- **1 (one):** File descriptor for the standard output.
- **2 (two):** File descriptor for the diagnostic output.

Example:



```
root@kali:~# cat file1 users 1> opfile 2> errfile
root@kali:~# cat opfile
apple,ball,cat,dog
ele,fan,gun,hut
ink,jug,kite,lamp
mud,not,owl,pea
queue,rat,sat,tab
unsub,van,wan,xray
root     tty7         2015-06-14 20:43 (:0)
root     pts/0        2015-06-14 20:50 (:0.0)
root     tty7         2015-06-14 20:43 (:0)
root     pts/0        2015-06-14 20:50 (:0.0)
root@kali:~# cat errfile
root@kali:~# 
```



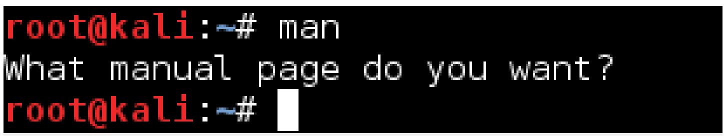
```
root@kali:~# cat file1 uses 1> opfile 2> errfile
root@kali:~# cat errfile
cat: uses: No such file or directory
root@kali:~# cat opfile
apple,ball,cat,dog
ele,fan,gun,hut
ink,jug,kite,lamp
mud,not,owl,pea
queue,rat,sat,tab
unsub,van,wan,xray
root@kali:~# 
```

man Manual Help:

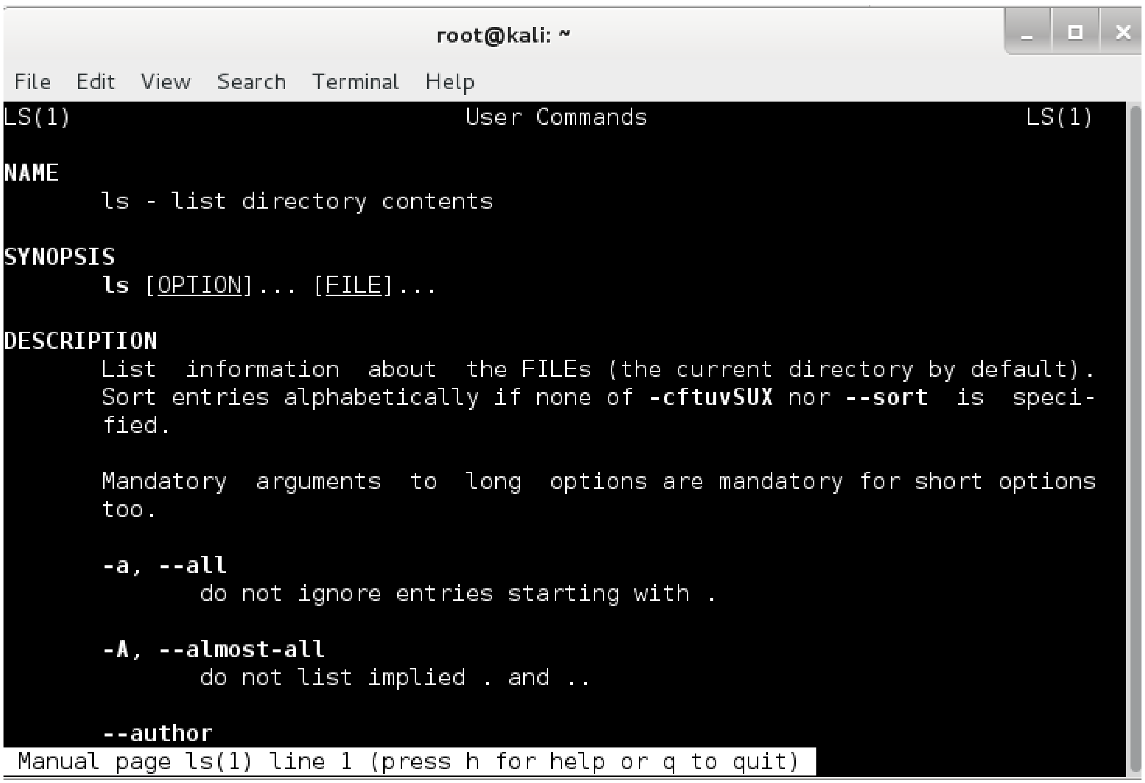
man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.
Each argument given to **man** is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section number, if provided, will direct **man** to look only in that section of the manual. The default action is to search in all of the available sections, following a pre-defined order and to show only the first page found, even if page exists in several sections.

Every command does not have all sections, but three sections namely NAME, SYNOPSIS and DESCRIPTION are generally seen in all the man pages. NAME presents one line introduction to the command, SYNOPSIS shows the syntax and DESCRIPTION provides a detailed description.

Example:



```
root@kali:~# man
What manual page do you want?
root@kali:~# 
```



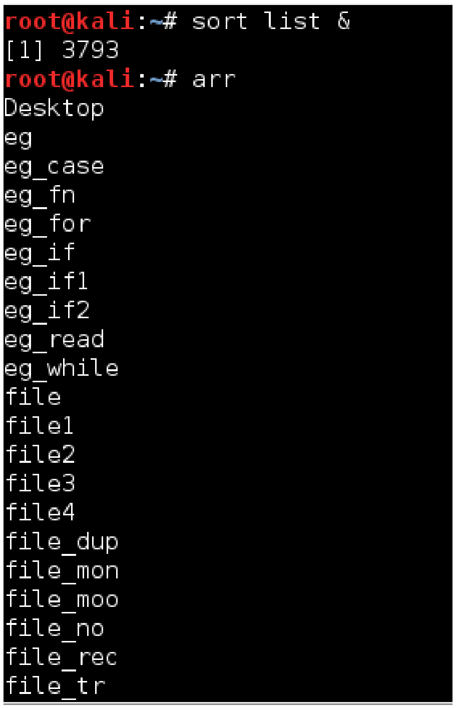
```
LS(1)                         User Commands                        LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
 Manual page ls(1) line 1 (press h for help or q to quit)
```

Background Processing:

A multitasking system lets a user do more than one job at a time. Since there can be only one job in the foreground, the rest of the jobs have to run in the background. There are two ways of doing this- with the shell's '**&**' operator and the **nohup** command. The latter permits you to log out while your jobs are running, but the former doesn't allow that.

&: No logging out

The & is the shell's operator used to run a process in the background. The parent in this case doesn't wait for the child's death. Just terminate the command line with an &; the command will run in the background.

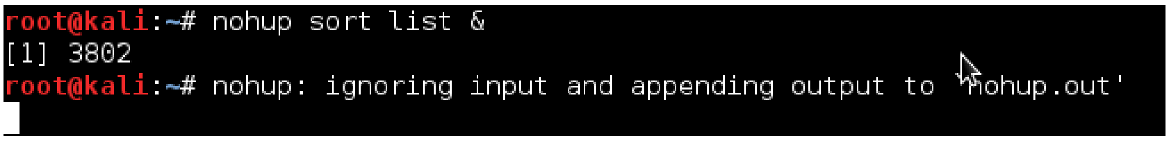


```
root@kali:~# sort list &
[1] 3793
root@kali:~# arr
Desktop
eg
eg_case
eg_fn
eg_for
eg_if
eg_if1
eg_if2
eg_read
eg_while
file
file1
file2
file3
file4
file_dup
file_mon
file_moo
file_no
file_rec
file_tr
```

The shell immediately returns a number- the PID of the invoked command. The prompt is returned and the shell is ready to accept another command even though the previous command has not been terminated yet. The shell, however, remains the parent of the background process. Using an &you can run as many jobs in the background as the system load permits.

nohup: Log out safely

Background jobs cease to run, however, when a user logs out. That happens because her shell is killed. And when the parent is killed, its children are also normally killed. The UNIX system permits a variation in this default behaviour. The **nohup** (no hangup) command, when prefixed to a command, permits execution of the process even after the user has logged out. You must use the & with it as well.



```
root@kali:~# nohup sort list &
[1] 3802
root@kali:~# nohup: ignoring input and appending output to 'nohup.out'
```

The shell returns the PID in this case too and some shells display a message as well. When the nohup command is run in these shells, nohup sends the standard output of the command to the file nohup.out. If you don’t get this message, then make sure that you have taken care of the output, using redirection if necessary. You can safely log out of the system without aborting the command.

Changing process priority with nice:

Processes in the UNIX system are usually executed with equal priority. This is not always desirable since high priority job must be completed at the earliest. UNIX offers the nice command, which is used with the & operator to reduce the priority of jobs. More important jobs can then have greater access to the system resources.

To run a job with low priority, the command name should be preferred with nice.

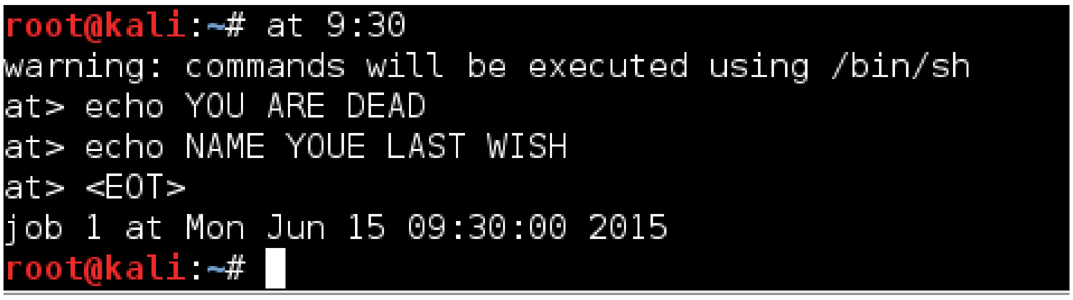
nice is a built in command in the C shell. nice values are system dependent and typically range from 1-19. Commands execute with a nice value that is generally in the middle of the range- usually 10. A higher nice value implies a lower priority. Nice reduces the priority of any process, thereby raising its nice value. You can also specify the nice value explicitly with the –n option.

A non-privileged user can not increase the priority of a process; that power is reserved for the super user. The nice value is displayed with the ps –o nice command.

at and batch command: Execute Later

at: one time execution

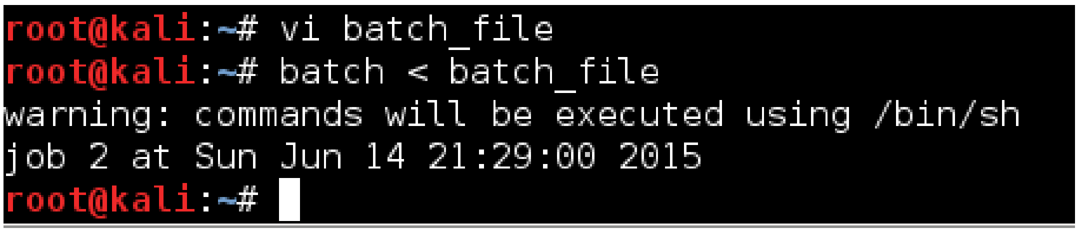
The **at** command schedules a command to be run once at a particular time. This can be any command that you normally have permission to run; anything from a simple reminder message, to a complex script. You start by running the **at** command at the command line, passing it the scheduled time as the option. It then places you at a special prompt, where you can type in the command (or series of commands) to be run at the scheduled time. When you're done, press **Control-D** on a new line, and your command will be placed in the queue.



```
root@kali:~# at 9:30
warning: commands will be executed using /bin/sh
at> echo YOU ARE DEAD
at> echo NAME YOUE LAST WISH
at> <EOT>
job 1 at Mon Jun 15 09:30:00 2015
root@kali:~# 
```

batch: Execute in Batch queue

The batch command also schedules jobs for later execution, but unlike **at**, jobs are executed as soon as the system load permits. The command does not takes any argument but uses an internal algorithm to determine the execution time. This prevents too many CPU hungry jobs from running at the same time.



```
root@kali:~# vi batch_file
root@kali:~# batch < batch_file
warning: commands will be executed using /bin/sh
job 2 at Sun Jun 14 21:29:00 2015
root@kali:~# 
```

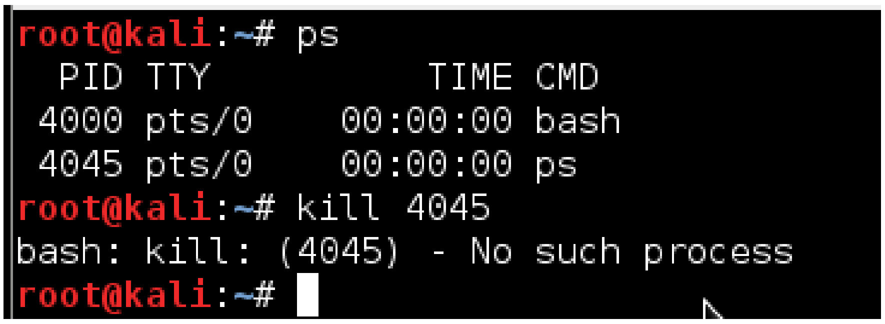
Any job scheduled with batch goes to a special at queue from where it can be removed with **at –r**.

kill : Premature Termination of a process

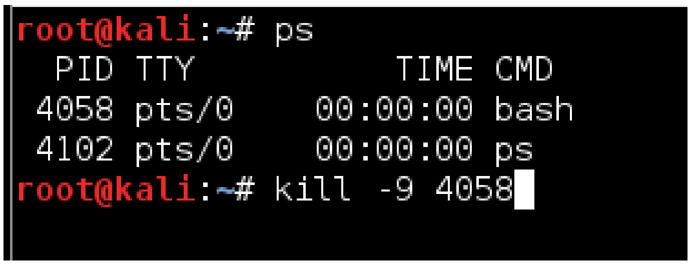
The kill command sends a signal, usually with the intention of killing one or more processes. kill is an internal command in most shells; the external **/bin/kill** is executed only when the shell lacks the kill capability. The command uses one or more PIDs as its arguments, and by default uses the SIGTERM (15) signal.

You can kill more than one process by giving their PIDs with kill command. If all the processes have the same parent then you can just kill the parent to kill all the processes.

Signal number 9 is a sure kill signal. The processes that cannot be killed in a normal way can be killed by providing 9 as a signal.

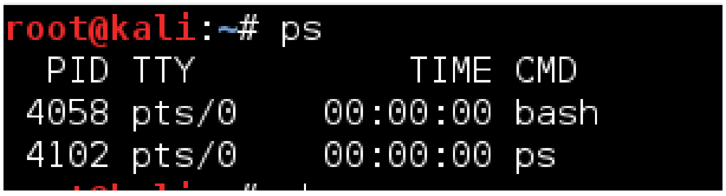


```
root@kali:~# ps
  PID TTY          TIME CMD
 4000 pts/0    00:00:00 bash
 4045 pts/0    00:00:00 ps
root@kali:~# kill 4045
bash: kill: (4045) - No such process
root@kali:~# 
```

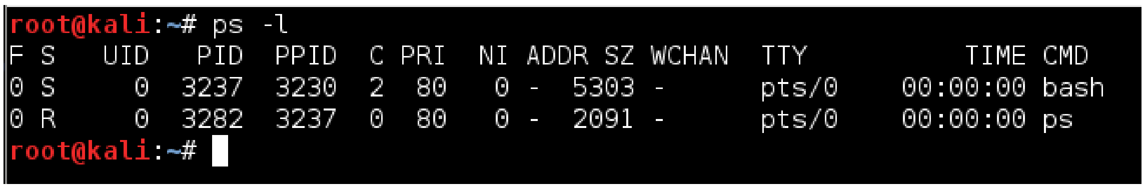

```
root@kali:~# ps
  PID TTY          TIME CMD
 4058 pts/0    00:00:00 bash
 4102 pts/0    00:00:00 ps
root@kali:~# kill -9 4058
```

ps: Process Status

The ps command on linux is one of the most basic commands for viewing the processes running on the system. It provides a snapshot of the current processes along with detailed information like user id, cpu usage, memory usage, command name etc.

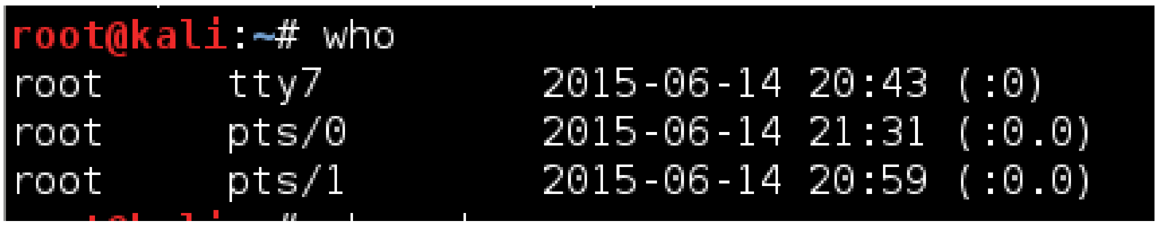

```
root@kali:~# ps
  PID TTY          TIME CMD
 4058 pts/0    00:00:00 bash
 4102 pts/0    00:00:00 ps
```

-l option: This option is used to show the long listing format in which all the information about the processes is shown

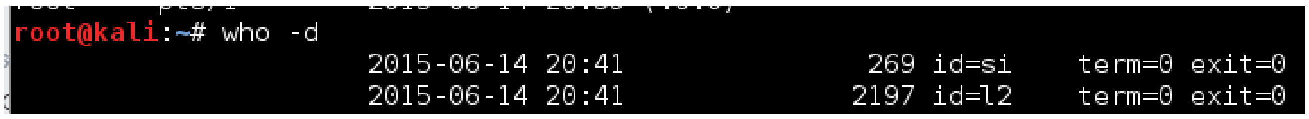

```
root@kali:~# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S     0  3237  3230  2  80   0 -  5303 -      pts/0    00:00:00 bash
0 R     0  3282  3237  0  80   0 -  2091 -      pts/0    00:00:00 ps
root@kali:~# 
```

who:

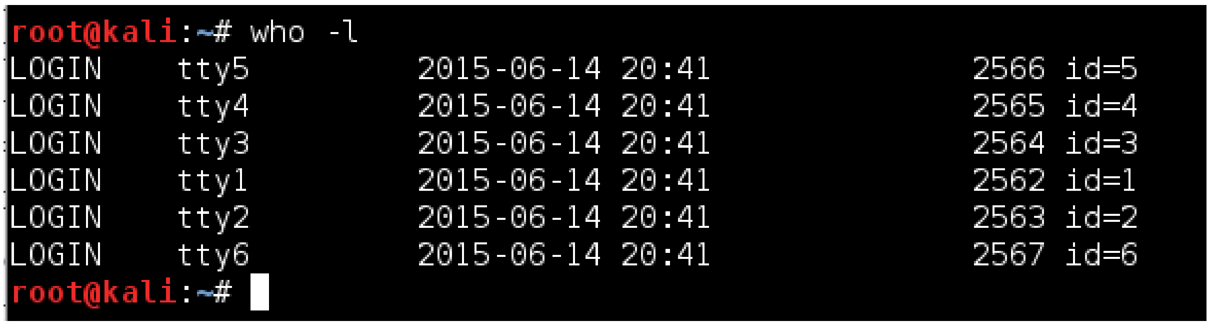
The **who** command prints information about all users who are currently logged in.


```
root@kali:~# who
root     tty7         2015-06-14 20:43 (:0)
root     pts/0        2015-06-14 21:31 (:0.0)
root     pts/1        2015-06-14 20:59 (:0.0)
```

-d option: displays dead process

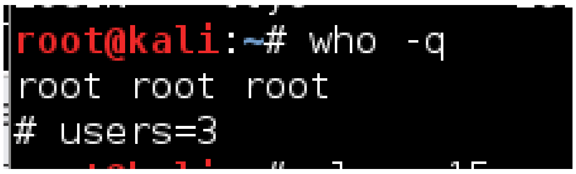

```
root@kali:~# who -d
                      2015-06-14 20:41               269 id=si    term=0 exit=0
                      2015-06-14 20:41              2197 id=l2    term=0 exit=0
```

-l option: Print system login processes.



```
root@kali:~# who -l
LOGIN    tty5         2015-06-14 20:41              2566 id=5
LOGIN    tty4         2015-06-14 20:41              2565 id=4
LOGIN    tty3         2015-06-14 20:41              2564 id=3
LOGIN    tty1         2015-06-14 20:41              2562 id=1
LOGIN    tty2         2015-06-14 20:41              2563 id=2
LOGIN    tty6         2015-06-14 20:41              2567 id=6
root@kali:~#
```

-q option: Displays all login names, and a count of all logged-on users.



```
root@kali:~# who -q
root root root
# users=3
```

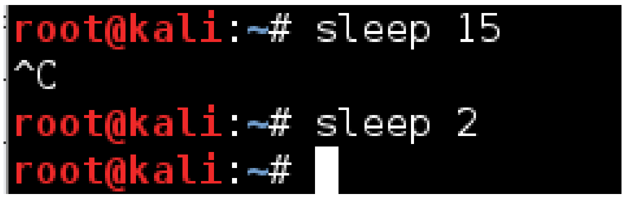
sleep command:

The **sleep** command pauses for an amount of time defined by NUMBER.

SUFFIX may be "**s**" for seconds (the default), "**m**" for minutes, "**h**" for hours, or "**d**" for days.

Some implementations require that NUMBER be an integer, but modern Linux implementations allow NUMBER to also be a floating-point value.

If more than one NUMBER is specified, **sleep** delays for the sum of their values.



```
root@kali:~# sleep 15
^C
root@kali:~# sleep 2
root@kali:~#
```

find : Locating files

The Linux **Find Command** is one of the most important and much used command in Linux systems. Find command used to search and locate list of files and directories based on conditions you specify for files that match the arguments. Find can be used in variety of conditions like you can find files by **permissions**, **users**, **groups**, **file type**, **date**, **size** and other possible criteria.

Syntax:

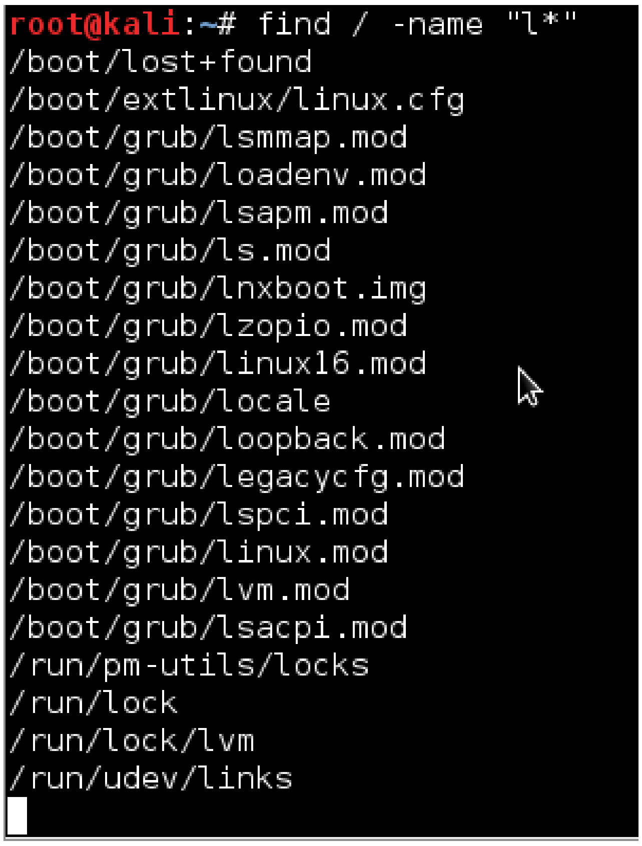
find *path_list selection_criteria action*

This is how find operates:

- First, it recursively examines all files in the directories specified in path_list.
- It then matches each file for one or more selection_criteria
- Finally, it takes some action on those selected files.

The path_list comprises one or more subdirectories separated by whitespace. There can also be a host os selection_criteria that you can use to match a file, and multiple actions to dispose of the file. This makes the command difficult to use initially, but it is a program that every user must master since it lets her make file selection under practically any condition.

Example:

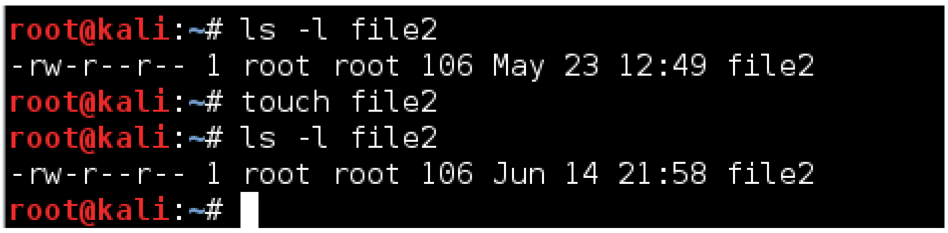


```
root@kali:~# find / -name "l*"
/boot/lost+found
/boot/extlinux/linux.cfg
/boot/grub/lsmmap.mod
/boot/grub/loadenv.mod
/boot/grub/lsapm.mod
/boot/grub/ls.mod
/boot/grub/lnxboot.img
/boot/grub/lzopio.mod
/boot/grub/linux16.mod
/boot/grub/locale
/boot/grub/loopback.mod
/boot/grub/legacycfg.mod
/boot/grub/lspci.mod
/boot/grub/linux.mod
/boot/grub/lvm.mod
/boot/grub/lsacpi.mod
/run/pm-utils/locks
/run/lock
/run/lock/lvm
/run/udev/links
```

touch: Changing the Time Stamps

The **touch** command updates the access and modification times of each FILE to the current system time.

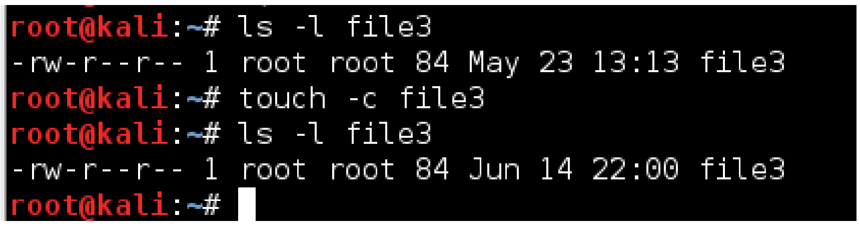
If you specify a FILE that does not already exist, **touch** creates an empty file with that name (unless the **-c** or **-h** options are specified)



```
root@kali:~# ls -l file2
-rw-r--r-- 1 root root 106 May 23 12:49 file2
root@kali:~# touch file2
root@kali:~# ls -l file2
-rw-r--r-- 1 root root 106 Jun 14 21:58 file2
root@kali:~#
```

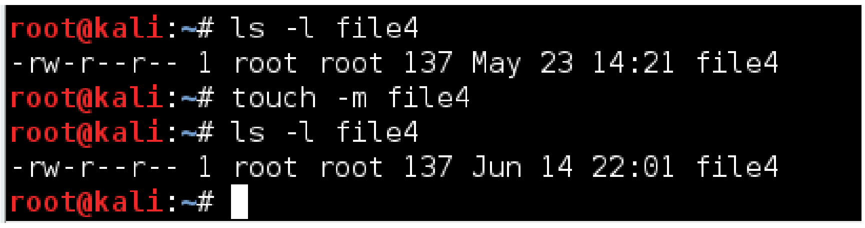
-a option: Change only the access time.

-c option: Do not create any files.



```
root@kali:~# ls -l file3
-rw-r--r-- 1 root root 84 May 23 13:13 file3
root@kali:~# touch -c file3
root@kali:~# ls -l file3
-rw-r--r-- 1 root root 84 Jun 14 22:00 file3
root@kali:~#
```

-m option: Change only the modification time.



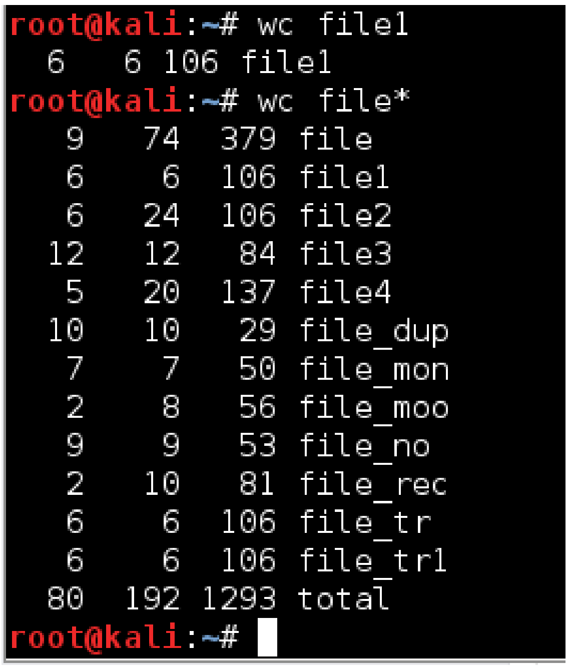
```
root@kali:~# ls -l file4
-rw-r--r-- 1 root root 137 May 23 14:21 file4
root@kali:~# touch -m file4
root@kali:~# ls -l file4
-rw-r--r-- 1 root root 137 Jun 14 22:01 file4
root@kali:~#
```

-t option: Use [[CC]YY]MMDDhhmm[.ss] instead of current time.

wc : word count

The wc (word count) command in Unix/Linux operating systems is used to find out number of newline count,word count, byte and characters count in a files specified by the file arguments. The syntax of wc command as shown below:

wc *options filename(s)*



```
root@kali:~# wc file1
  6   6 106 file1
root@kali:~# wc file*
   9   74  379 file
   6    6  106 file1
   6   24  106 file2
  12   12   84 file3
   5   20  137 file4
  10   10   29 file_dup
   7    7   50 file_mon
   2    8   56 file_moo
   9    9   53 file_no
   2   10   81 file_rec
   6    6  106 file_tr
   6    6  106 file_tr1
  80  192 1293 total
root@kali:~#
```

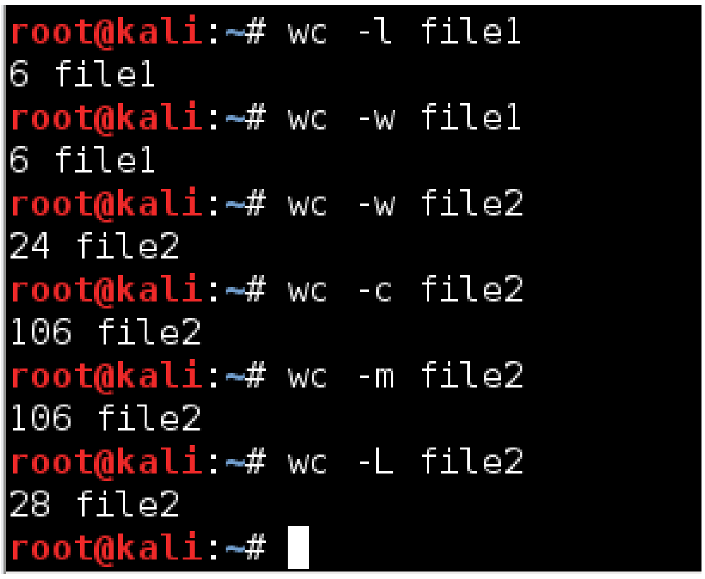
-l option : Prints the number of lines in a file.

-w option : prints the number of words in a file.

-c option : Displays the count of bytes in a file.

-m option : prints the count of characters from a file.

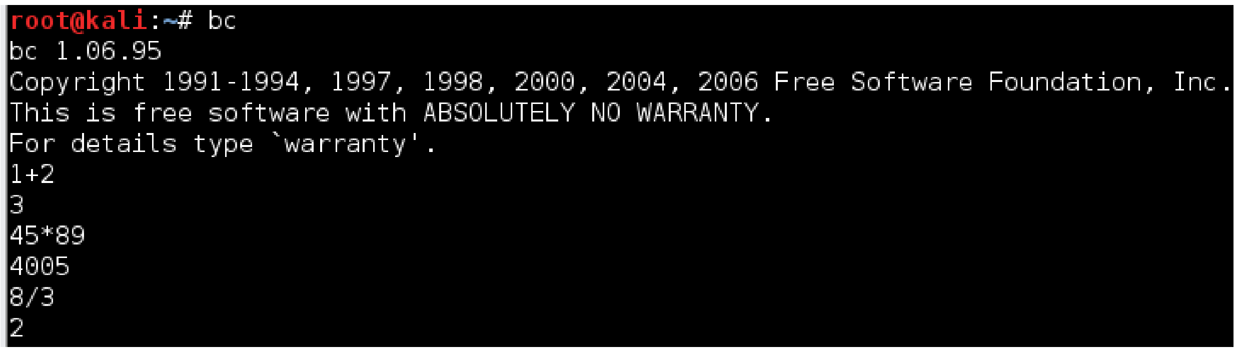
-L option: prints only the length of the longest line in a file.



```
root@kali:~# wc -l file1
6 file1
root@kali:~# wc -w file1
6 file1
root@kali:~# wc -w file2
24 file2
root@kali:~# wc -c file2
106 file2
root@kali:~# wc -m file2
106 file2
root@kali:~# wc -L file2
28 file2
root@kali:~#
```

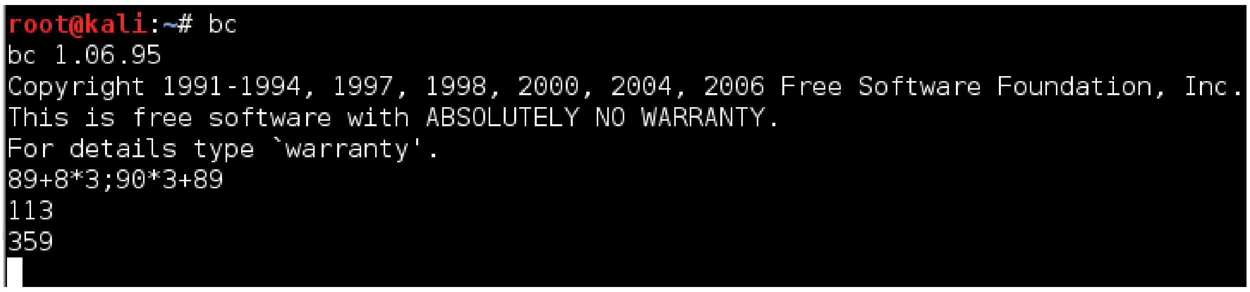
bc: The linux calculator

When you invoke bc without arguments, the cursor keeps on blinking and nothing seems to happen. bc belongs to a family of commands (called filters) that expect input from the keyboard when used without an argument. You can come out of the bc by using Ctrl+d.



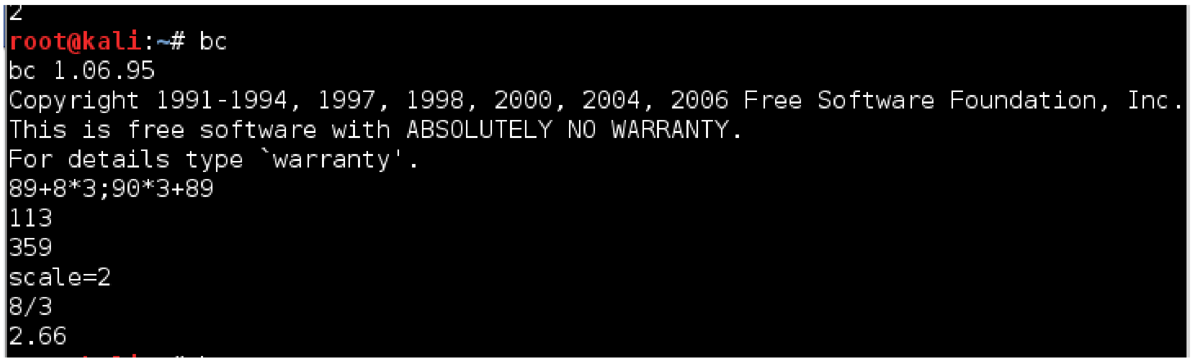
```
root@kali:~# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
1+2
3
45*89
4005
8/3
2
```

bc shows the output of the computation in the next line. To make multiple calculations you can give the expressions in the same line separated by a ; the output of each computation is however shown in different line.



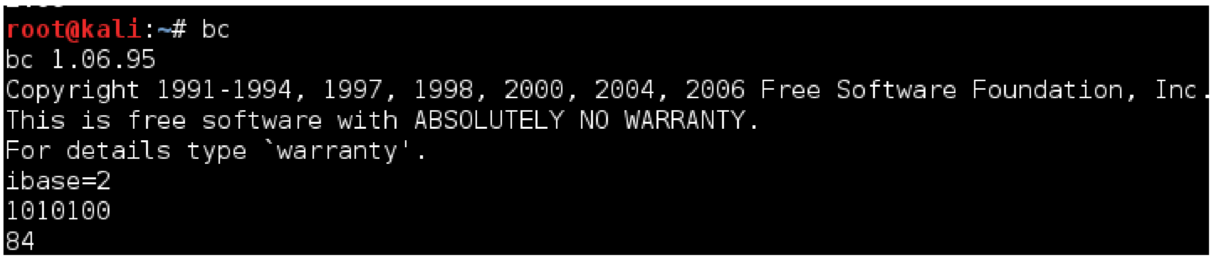
```
root@kali:~# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
89+8*3;90*3+89
113
359
```

In general bc performs only integer computation and truncates the decimal portion. However, to enable floating point computation, you have to set scale to the number of digit of precision before you key in the expression.

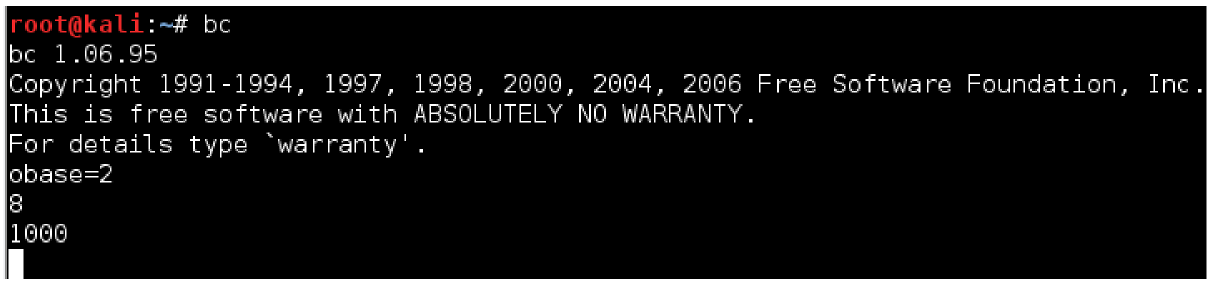


```
2
root@kali:~# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
89+8*3;90*3+89
113
359
scale=2
8/3
2.66
```

bc is also helpful in converting numbers from one base to another. You can provide ibase (input base) and obase (output) base to enter or get the number in a different base.



```
root@kali:~# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
ibase=2
1010100
84
```



```
root@kali:~# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
obase=2
8
1000
```

expr: Computation

expr syntax

expr *arg1 operator arg2* [*operator arg3* ...]

Evaluate arguments as expressions and print the results. Arguments and operators must be separated by spaces. In most cases, an argument is an integer, typed literally or represented by a shell variable. There are three types of operators: arithmetic, relational, and logical, as well as keyword expressions.

Arithmetic operators

Use these to produce mathematical expressions whose results are printed:

+

Add *arg2* to *arg1*.

-

Subtract *arg2* from *arg1*.

*

Multiply the arguments.

/

Divide *arg1* by *arg2*.

%

Take the remainder when *arg1* is divided by *arg2*.

Addition and subtraction are evaluated last, unless they are grouped inside parentheses. The symbols *, **(**, and **)** have meaning to the shell, so they must be escaped (preceded by a backslash or enclosed in single quotes).

Relational operators

Use these to compare two arguments. Arguments can also be words, in which case comparisons are defined by the locale. If the comparison statement is true, the result is 1; if false, the result is 0. Symbols **>** and **<** must be escaped.

=, ==

Are the arguments equal?

!=

Are the arguments different?

>

Is *arg1* greater than *arg2*?

>=

Is *arg1* greater than or equal to *arg2*?

<

Is *arg1* less than *arg2*?

<=

Is *arg1* less than or equal to *arg2*?

Logical operators

Use these to compare two arguments. Depending on the values, the result can be *arg1* (or some portion of it), *arg2*, or 0. Symbols **|** and & must be escaped.

|

Logical OR; if *arg1* has a nonzero (and nonnull) value, the result is *arg1*; otherwise, the result is *arg2*.

&

Logical AND; if both *arg1* and *arg2* have a nonzero (and nonnull) value, the result is *arg1*; otherwise, the result is 0.

Keywords

index *string character-list*

Return the first position in *string* that matches the first possible character listed in *character-list*. Continue through *character-list* until a match is found, or return 0.

length *string*

Return the length of *string*.

match *string regex*

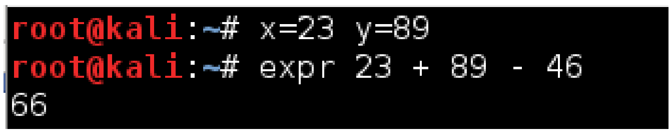
Same as *string* : *regex*.

quote *token*

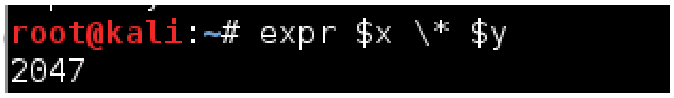
Treat *token* as a string, even if it would normally be a keyword or an operator.

substr *string start length*

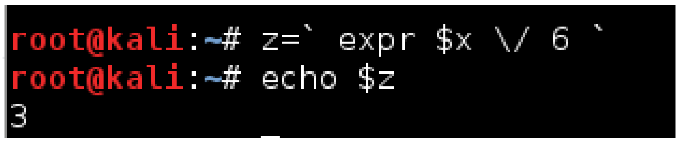
Return a section of *string*, beginning with *start*, with a maximum length of *length* characters. Return null when given a negative or nonnumeric *start* or *length*.



```
root@kali:~# x=23 y=89
root@kali:~# expr 23 + 89 - 46
66
```



```
root@kali:~# expr $x \* $y
2047
```



```
root@kali:~# z=` expr $x \/ 6 `
root@kali:~# echo $z
3
```

The vi Editor

The default editor that comes with the UNIX operating system is called vi (visual editor). [Alternate editors for UNIX environments include pico and emacs, a product of GNU.]
The UNIX vi editor is a full screen editor and has two modes of operation:

1. Command mode commands which cause action to be taken on the file, and
2. Insert mode in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert

mode, every character typed is added to the text in the file; pressing the <Esc> (Escape) key turns off the Insert mode.
While there are a number of vi commands, just a handful of these is usually sufficient for beginning vi users.

To Get Into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

* **vi filename**	*edit filename starting at line 1*
vi -r filename	*recover filename that was being edited when system crashed*

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

* **:x<Return>**	*quit vi, writing out modified file to file named in original invocation*
:wq<Return>	*quit vi, writing out modified file to file named in original invocation*
:q<Return>	*quit (or exit) vi*
* **:q!<Return>**	*quit vi even though latest changes have not been saved for this vi call*

Moving the Cursor

Unlike many of the PC and MacIntosh editors, **the mouse does not move the cursor** within the vi editor screen (or window). You must use the the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in viand should be avoided.

If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two.

In the table below, the symbol ^ before a letter means that the <Ctrl> key should be held down while the letter key is pressed.

* **j *or* <Return>**	*move cursor down one line*

[*or* down-arrow]	
*** k [*or* up-arrow]**	*move cursor up one line*
*** h *or* <Backspace> [*or* left-arrow]**	*move cursor left one character*
*** l *or* <Space> [*or* right-arrow]**	*move cursor right one character*
*** 0 (zero)**	*move cursor to start of current line (the one with the cursor)*
*** $**	*move cursor to end of current line*
w	*move cursor to beginning of next word*
b	*move cursor back to beginning of preceding word*
:0<Return> *or* 1G	*move cursor to first line in file*
:n<Return> *or* nG	*move cursor to line n*
:$<Return> *or* G	*move cursor to last line in file*

Screen Manipulation

The following commands allow the vi editor screen (or window) to move up or down several lines and to be refreshed.

^f	*move forward one screen*
^b	*move backward one screen*
^d	*move down (forward) one half screen*
^u	*move up (back) one half screen*
^l	*redraws the screen*
^r	*redraws the screen, removing deleted lines*

Adding, Changing, and Deleting Text

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead use the commands in the following tables.

Perhaps the most important command is the one that allows you to back up and *undo* your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

*** u** *UNDO WHATEVER YOU JUST DID; a simple toggle*

The main purpose of an editor is to create, add, or modify text for a file.

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vieditor back into command mode.

* **i**	*insert text before cursor, until <Esc> hit*
I	*insert text at beginning of current line, until <Esc> hit*
* **a**	*append text after cursor, until <Esc> hit*
A	*append text to end of current line, until <Esc> hit*
* **o**	*open and put text in a new line below current line, until <Esc> hit*
* **O**	*open and put text in a new line above current line, until <Esc> hit*

Changing Text

The following commands allow you to modify text.

* **r**	*replace single character under cursor (no <Esc> needed)*
R	*replace characters, starting with current cursor position, until <Esc> hit*
cw	*change the current word with new text, starting with the character under cursor, until <Esc> hit*
cNw	*change N words beginning with character under cursor, until <Esc> hit; e.g., c5w changes 5 words*
C	*change (replace) the characters in the current line, until <Esc> hit*
cc	*change (replace) the entire current line, stopping when <Esc> is hit*
Ncc ***or*** **cNc**	*change (replace) the next N lines, starting with the current line, stopping when <Esc> is hit*

Deleting Text

The following commands allow you to delete text.

* **x**	*delete single character under cursor*
Nx	*delete N characters, starting with character under cursor*
dw	*delete the single word beginning with character under cursor*
dNw	*delete N words beginning with character under cursor; e.g., d5w deletes 5 words*
D	*delete the remainder of the line, starting with current cursor*

	position
*** dd**	*delete entire current line*
Ndd *or* dNd	*delete N lines, beginning with the current line; e.g., 5dd deletes 5 lines*

Cutting and Pasting Text

The following commands allow you to copy and paste text.

yy	*copy (yank, cut) the current line into the buffer*
Nyy *or* yNy	*copy (yank, cut) the next N lines, including the current line, into the buffer*
p	*put (paste) the line(s) in the buffer into the text after the current line*

Other Commands

Searching Text

A common occurrence in text editing is to replace one word or phase by another. To locate instances of particular sets of characters (or strings), use the following commands.

/string	*search forward for occurrence of string in text*
?string	*search backward for occurrence of string in text*
n	*move to next occurrence of search string*
N	*move to next occurrence of search string in opposite direction*

Determining Line Numbers

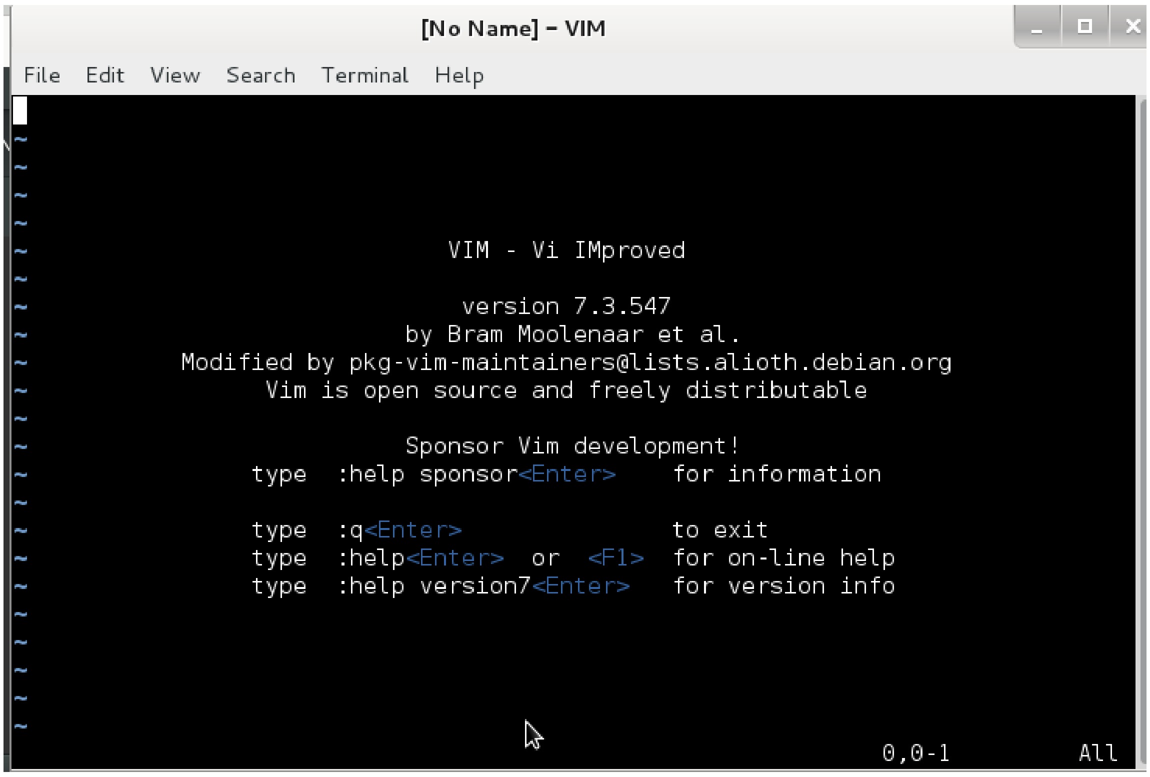
Being able to determine the line number of the current line or the total number of lines in the file being edited is sometimes useful.

:.=	*returns line number of current line at bottom of screen*
:=	*returns the total number of lines at bottom of screen*
^g	*provides the current line number, along with the total number of lines, in the file at the bottom of the screen*

Saving and Reading Files

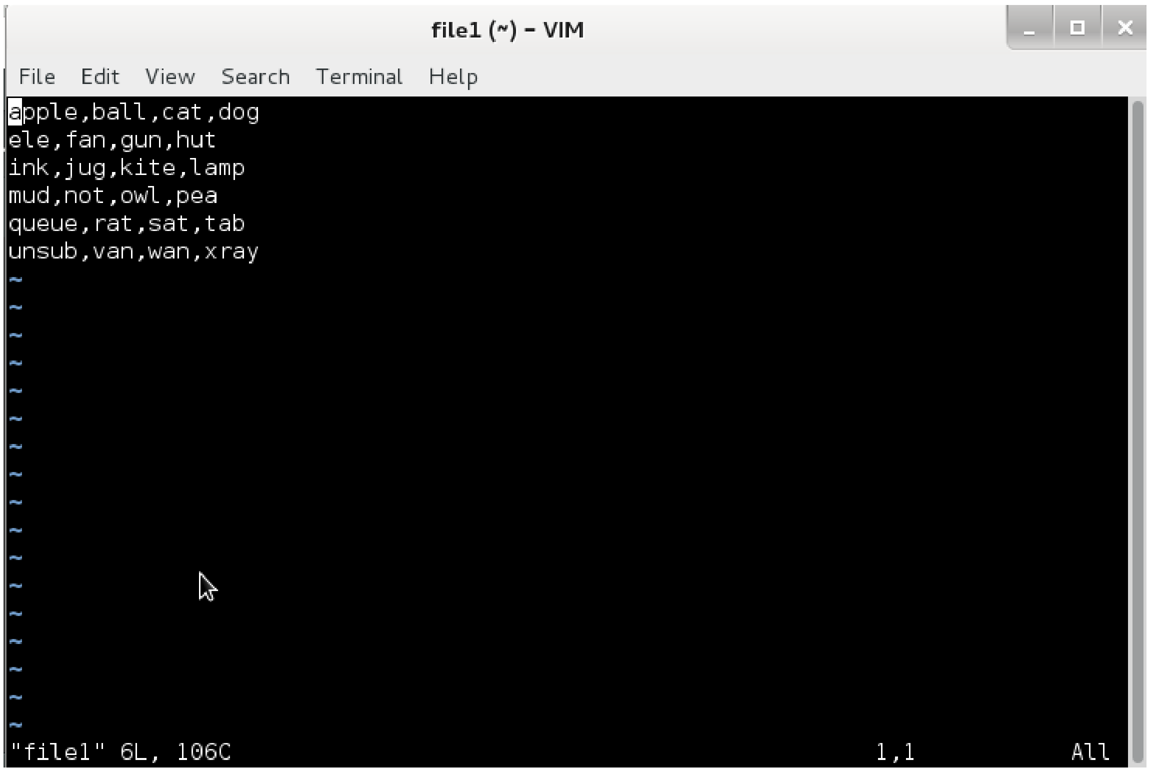
These commands permit you to input and output files other than the named file with which you are currently working.

:r filename<Return>	*read file named filename and insert after current line (the line with cursor)*
:w<Return>	*write current contents to file named in original vi call*
:w newfile<Return>	*write current contents to a new file named newfile*
:12,35w smallfile<Return>	*write the contents of the lines numbered 12 through 35 to a new file named smallfile*
:w! prevfile<Return>	*write current contents over a pre-existing file named prevfile*



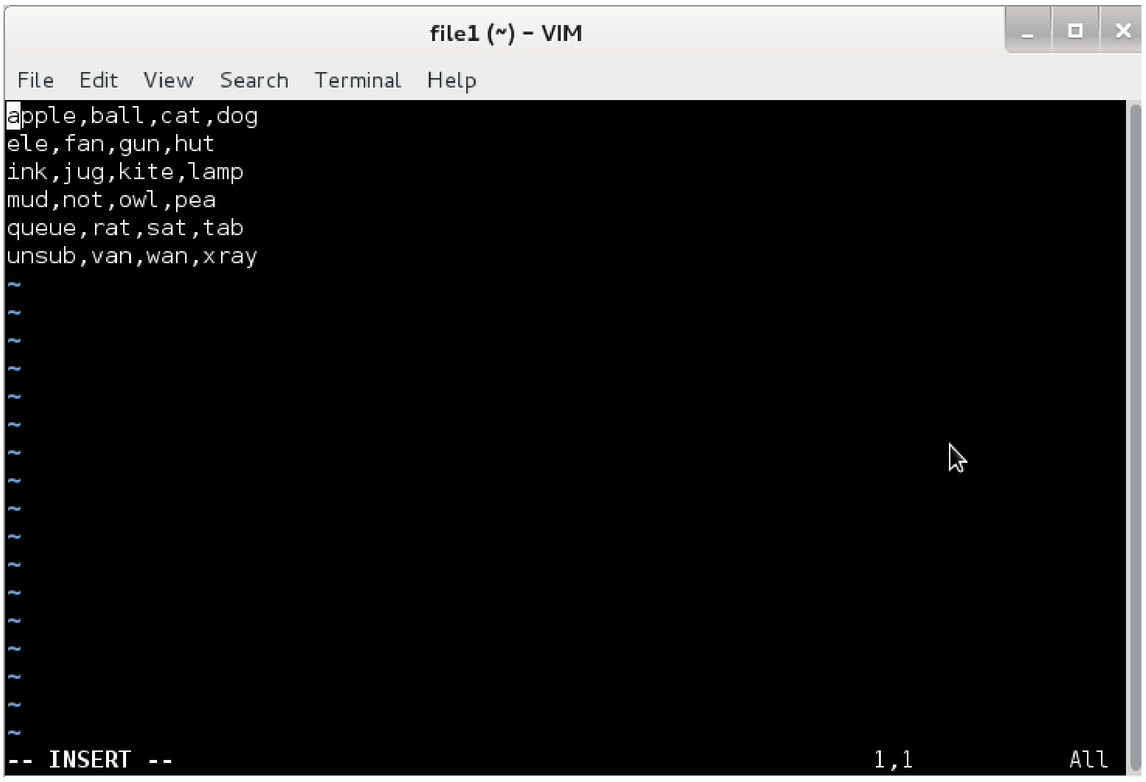
```
~
~
~
~
~                          VIM - Vi IMproved
~
~                           version 7.3.547
~                       by Bram Moolenaar et al.
~          Modified by pkg-vim-maintainers@lists.alioth.debian.org
~                Vim is open source and freely distributable
~
~                       Sponsor Vim development!
~               type  :help sponsor<Enter>    for information
~
~               type  :q<Enter>               to exit
~               type  :help<Enter>  or  <F1>  for on-line help
~               type  :help version7<Enter>   for version info
~
~
~
~
~
                                                       0,0-1         All
```

vi editor default screen



```
apple,ball,cat,dog
ele,fan,gun,hut
ink,jug,kite,lamp
mud,not,owl,pea
queue,rat,sat,tab
unsub,van,wan,xray
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"file1" 6L, 106C                                               1,1           All
```

File opened in command mode



```
apple,ball,cat,dog
ele,fan,gun,hut
ink,jug,kite,lamp
mud,not,owl,pea
queue,rat,sat,tab
unsub,van,wan,xray
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                   1,1           All
```

File opened in Insert mode