Linux Commands

date

Displays the current time and date.

Examples

```
chirag@chirag-VirtualBox:/$ date
Mon 17 Feb 14:39:56 IST 2020
```

cal

Displays a calendar of the current month.

Examples

```
chirag@chirag-VirtualBox:/$ cal
   February 2020
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
```

df

Displays current amount of free space on your disk drives.

Examples

```
chirag@chirag-VirtualBox:/$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              982896       0    982896   0% /dev
tmpfs             203880    1224    202656   1% /run
/dev/sda1       20509264 5830812  13613596  30% /
tmpfs            1019380   20316    999064   2% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
tmpfs            1019380       0   1019380   0% /sys/fs/cgroup
tmpfs             203876      56    203820   1% /run/user/1000
```

free

Display the amount of free memory.

Examples

```
chirag@chirag-VirtualBox:/$ free
              total        used        free      shared  buff/cache   available
Mem:        2038764     1252396      223656       36840      562712      61742
8
Swap:        969960       13824      956136
```

pwd

To display the current working directory, we use the pwd (print working directory) command.

Examples

```
chirag@chirag-VirtualBox:/$ pwd
/
```

ls

To list the files and directories in the current working directory, we use the ls command.

Options

Option	Description
-a, --all	List all files, even those with names that begin with a period, which are normally not listed (i.e., hidden).
-d, --directory	Ordinarily, if a directory is specified, ls will list the contents of the directory, not the directory itself. Use this option in conjunction with the -l option to see details about the directory rather than its contents.
-F, --classify	This option will append an indicator character to the end of each listed name (for Examples, a forward slash if the name is a directory).
-h, --human-readable	In long format listings, display file sizes in human-readable format rather than in bytes.
-l	Display results in long format.
-r	Display the results in reverse order. Normally, ls displays its results in ascending alphabetical order.
-S	Sort results by file size.
-t	Sort by modification time.

Examples

```
chirag@chirag-VirtualBox:/$ ls
bin    dev   initrd.img      lib64       mnt   root  srv       tmp  VBox.log
...

chirag@chirag-VirtualBox:/$ ls -l
total 970068
drwxr-xr-x   2 root root      4096 Feb  4 10:32 bin
drwxr-xr-x   3 root root      4096 Feb  4 10:33 boot
drwxrwxr-x   2 root root      4096 Feb  4 10:24 cdrom
drwxr-xr-x  16 root root      3860 Feb 17 14:32 dev
drwxr-xr-x 126 root root     12288 Feb 17 14:29 etc
drwxr-xr-x   3 root root      4096 Feb  4 10:31 home
...
```

---

file

When invoked, the file command will print a brief description of the file's contents.

Syntax

file *filename*

Examples

```
chirag@chirag-VirtualBox:~/Downloads$ file code.deb
code.deb: Debian binary package (format 2.0)
```

---

cd

To change your working directory we use the cd command.

cd Shortcuts

Shortcut	Result
cd	Changes the working directory to your home directory.
cd -	Changes the working directory to the previous working directory.
cd ~username	Changes the working directory to the home directory of *username*
cd ~	Changes the working directory to the home directory of current user
cd /	Changes the working directory to your root directory.
cd ..	Changes the working directory to the parent of current directory.

```
chirag@chirag-VirtualBox:~/Documents$ pwd
/home/chirag/Documents
chirag@chirag-VirtualBox:~/Documents$ cd /
chirag@chirag-VirtualBox:/$ pwd
/
```

```
chirag@chirag-VirtualBox:/$ cd ~
chirag@chirag-VirtualBox:~$ pwd
/home/chirag
```

less

The less command is a program to view text files.

Syntax

less *filename*

less Keybinds

Command	Action
PageUp or b	Scroll back one page
PageDown or Space	Scroll forward one page
Up Arrow	Scroll Up one line
Down Arrow	Scroll down one line
G	Move to the end of the text file
g	Move to the beginning of the text file
n	Search for the next occurrence of the previous search
h	Display help screen
q	Quit less
/*characters*	Search forward to the next occurrence of characters

mkdir

The mkdir command is used to create directories.

Syntax

mkdir *directory...*

Examples

```
chirag@chirag-VirtualBox:~/Documents/TestFolder$ mkdir dir1 dir2 dir3
chirag@chirag-VirtualBox:~/Documents/TestFolder$ ls
dir1  dir2  dir3
```

cp

The cp command copies files or directories.

Syntax

- **cp** *item1 item2*

Copy the single file or directory item1 to file or directory item2

- **cp** *item... directory*

Copy multiple items (either files or directories) into a directory.

cp Options

Option	Meaning
-a, --archive	Copy the files and directories and all of their attributes, including ownerships and permissions. Normally, copies take on the default attributes of the user performing the copy.
-i, --interactive	Before overwriting an existing file, prompt the user for confirmation. I f this option is not specified, cp will silently overwrite files.
-r,--recursive	Recursively copy directories and their contents. This option (or the -a option) is required when copying directories.
-u, --update	When copying files from one directory to another, copy only files that either don't exist or are newer than the existing corresponding files in the destination directory.
-v, --verbose	Display informative messages as the copy is performed.

Examples

Command	Result
cp *file1 file2*	Copy file1 to file2. **If file2 exists, it is overwritten with the contents of file1.** If file2 does not exist, it is created.
cp *-i file1 file2*	Same as above, except that if file2 exists, the user is prompted before it is overwritten.
cp *file1 file2* **dir1**	Copy file1 and file2 into directory dir1. dir1 must already exist.
cp dir1/* *dir2*	Using a wildcard, all the files in dir1 are copied into dir2. dir2 must already exist
cp -r *dir1 dir2*	Copy directory dir1 (and its contents) to directory dir2. If directory dir2 does not exist, it is created and will contain the same contents as directory dir1.

mv

The mv command performs both file moving and file renaming, depending on how it is used. In either case, the original filename no longer exists after the operation. mv is used in much the same way as cp.

Syntax

- **mv** *item1 item2*

Move or rename file or directory item1 to item2 or

- **mv** *item... directory*

Move one or more items from one directory to another.

Examples

Command	Result
mv *file1 file2*	Move file1 to file2. I f file2 exists, it is overwritten with the contents of file1. If file2 does not exist, it is created. In either case, file1 ceases to exist.
mv -i *file1 file2*	Same as above, except that if file2 exists, the user is prompted before it is overwritten.
mv *file1 file2* dir1	Move file1 and file2 into directory dir1. dir1 must already exist.
mv *dir1 dir2*	Move directory dir1 (and its contents) into directory dir2. If directory dir2 does not exist, create directory dir2, move the contents of directory dir1 into dir2, and delete directory dir1.

rm

The rm command is used to remove (delete) files and directories.

Syntax

rm *item...*

rm Options

Options	Meaning
-i, --interactive	Before deleting an existing file, prompt the user for confirmation. **If this option is not specified, rm will silently delete files.**
-r, --recursive	Recursively delete directories. This means that if a directory being deleted has subdirectories, delete them too. To delete a directory, this option must be specified.
-f, --force	Ignore nonexistent files and do not prompt. This overrides the --interactive option.
-v, --verbose	Display informative messages as the deletion is performed

Examples

Command	Result
rm *file1*	Delete file1 silently.

rm -i *file1*	Before deleting file1, prompt the user for confirmation.
rm -r *file dir1*	Delete file1 and dir1 and its contents.
rm -rf *file1 dir1*	Same as above, except that if either file1 or dir1 does not exist, rm will continue silently.

```
chirag@chirag-VirtualBox:~/test$ ls
dir1  myFile
chirag@chirag-VirtualBox:~/test$ rm myFile
chirag@chirag-VirtualBox:~/test$ ls
dir1
chirag@chirag-VirtualBox:~/test$ cd dir1
chirag@chirag-VirtualBox:~/test/dir1$ ls
myFile-link
chirag@chirag-VirtualBox:~/test/dir1$ less myFile-link
myFile-link: No such file or directory
chirag@chirag-VirtualBox:~/test/dir1$ cd ..
chirag@chirag-VirtualBox:~/test$ rm -i dir1
rm: cannot remove 'dir1': Is a directory
chirag@chirag-VirtualBox:~/test$ rm -ri dir1
rm: descend into directory 'dir1'? y
rm: remove symbolic link 'dir1/myFile-link'? y
rm: remove directory 'dir1'? y
```

ln

The ln command is used to create either hard or symbolic links.

Syntax

- **ln** *file link*

Create a hard link

- **ln** -s *item link*

Create a symbolic link where item is either a file or a directory (It's relative or absolute path).

Hard Links

Hard links are the original Unix way of creating links; symbolic links are more modern. By default, every file has a single hard link that gives the file its name. When we create a hard link, we create an additional directory entry for a file. Hard links have two important limitations:

- A hard link cannot reference a file outside its own filesystem. This means a link cannot reference a file that is not on the same disk parti- tion as the link itself.
- A hard link cannot reference a directory.

Example

```
chirag@chirag-VirtualBox:~/test$ ls -li
total 4
1190489 -rw-rw-r-- 1 chirag chirag 6 Feb 18 13:17 myFile
```

```
chirag@chirag-VirtualBox:~/test$ ln myFile myFile-link
chirag@chirag-VirtualBox:~/test$ ls -li
total 8
1190489 -rw-rw-r-- 2 chirag chirag 6 Feb 18 13:17 myFile
1190489 -rw-rw-r-- 2 chirag chirag 6 Feb 18 13:17 myFile-linkchirag@chirag-Vir
tualBox:~/test$ ls -li
total 4
1190489 -rw-rw-r-- 1 chirag chirag 6 Feb 18 13:17 myFile
chirag@chirag-VirtualBox:~/test$ ln myFile myFile-link
chirag@chirag-VirtualBox:~/test$ ls -li
total 8
1190489 -rw-rw-r-- 2 chirag chirag 6 Feb 18 13:17 myFile
1190489 -rw-rw-r-- 2 chirag chirag 6 Feb 18 13:17 myFile-link
```

Symbolic Links

Symbolic links were created to overcome the limitations of hard links. Symbolic links work by creating a special type of file that contains a text pointer to the referenced file or directory. In this regard they operate in much the same way as a Windows shortcut. A file pointed to by a symbolic link and the symbolic link itself are largely indistinguishable from one another. For example, if you write some- thing to the symbolic link, the referenced file is also written to. However, when you delete a symbolic link, only the link is deleted, not the file itself. If the file is deleted before the symbolic link, the link will continue to exist but will point to nothing. In this case, the link is said to be *broken*.

Example

```
chirag@chirag-VirtualBox:~/test$ ls
dir1  myFile
chirag@chirag-VirtualBox:~/test$ cd dir1
chirag@chirag-VirtualBox:~/test/dir1$ ln -s ../myFile myFile-link
chirag@chirag-VirtualBox:~/test/dir1$ ls -l
total 0
lrwxrwxrwx 1 chirag chirag 9 Feb 18 13:27 myFile-link -> ../myFile
```

type

The type command is a shell builtin that displays the kind of command the shell will execute.

Syntax

type *command*

Examples

```
chirag@chirag-VirtualBox:~/test$ type type
type is a shell builtin
chirag@chirag-VirtualBox:~/test$ type cd
cd is a shell builtin
chirag@chirag-VirtualBox:~/test$ type ls
ls is aliased to `ls --color=auto`
chirag@chirag-VirtualBox:~/test$ type cp
cp is hashed (/bin/cp)
```

which

To determine the exact location of a given executable, the which command is used.

Syntax

which *command*

Examples

```
chirag@chirag-VirtualBox:~/test$ which rm
/bin/rm
chirag@chirag-VirtualBox:~/test$ which ls
/bin/ls
chirag@chirag-VirtualBox:~/test$ which less
/usr/bin/less
```

help

bash has a built-in help facility for each of the shell builtins. To use it, type help followed by the name of the *shell builtin* (bash supports a number of commands internally called shell builtins. The *cd* command, for example, is a shell builtin).

Syntax

help *command*

Example

```
chirag@chirag-VirtualBox:/$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR. The default DIR is the value of the
    HOME shell variable.
    ...
```

man

Most executable programs intended for command-line use provide a formal piece of documentation called a manual or man page. A special paging program called man is used to view them.

Syntax

man *program*

The **manual** that man displays is broken into sections and covers not only user commands but also system administration commands, programming interfaces, file formats, and more.

man Page Oraganization

Section	Contents
1	User commands.
2	Programming interfaces for kernel system calls.
3	Programming interfaces to the C library.
4	Special files such as device nodes and drivers.
5	File formats.
6	Games and amusements such as screensavers.
7	Miscellaneous.
8	System administration commands.

To search in a specific section of the manual we can use arguments:

man *section search_term*

Example

```
chirag@chirag-VirtualBox:/$ man 5 passwd
```

will display the man pag describing the file format of the */etc/passwd* file.

---

apropos

It is also possible to search the list of man pages for possible matches based on a search term using apropos command.

Syntax

apropos *search_term*

Example

```
chirag@chirag-VirtualBox:/$ apropos internet
Email::Valid (3pm)   - Check validity of Internet email addresses
ftp (1)              - Internet file transfer program
ippfind (1)          - find internet printing protocol printers
ippserver (1)        - a simple internet printing protocol server
ipptool (1)          - perform internet printing protocol requests
Mail::Internet (3pm) - manipulate email messages
...
```

The first field in each line of output is the name of the man page, and the second field shows the section. Note that the man command with the -k option performs exactly the same function as apropos.

---

whatis

The whatis program displays the name and a one-line description of a *man* page matching a specified keyword.

Syntax

whatis *command*

Example

```
chirag@chirag-VirtualBox:/$ whatis ls
ls (1)               - list directory contents
```

---

info

The GNU Project provides an alternative to man pages called info pages. Info pages are displayed with a reader program named *info*. Info pages are hyperlinked much like web pages. The info program reads info files, which are tree-structured into individual nodes, each containing a single topic. Info files contain hyperlinks that can move you from node to node. A hyperlink can be identified by its leading asterisk and is activated by placing the cursor upon it and pressing the ENTER key.

info Reader Keybindings

Command	**Action**
?	Display command help.
PageUp or BackSpace	Display previous page.
PageDown or SpaceBar	Display next page.
n	Display next node.
p	Display previous node.
u	Display the parent node of the currently displayed node, usually a menu.
Enter	Follow the *hyperlink* at the cursor location.
q	Quit.

---

alias / unalias

An alias is a command that we can define ourselves, built from other commands. To remove an alias use unalias command.

Syntax

alias name='*string*'

unalias *command*

Example

```
chirag@chirag-VirtualBox:/$ alias user-files="cd /usr; ls; cd -;"
chirag@chirag-VirtualBox:/$ user-files
bin  games  include  lib  libexec  local  sbin  share  src
/
chirag@chirag-VirtualBox:/$ type user-files
user-files is aliased to `cd /usr; ls; cd -;`
chirag@chirag-VirtualBox:/$ unalias user-files
chirag@chirag-VirtualBox:/$ type user-files
bash: type: user-files: not found
```

---

cat

The cat command reads one or more files and copies them to standard output.

Syntax

cat *files...*

cat is often used to display short text files. Since cat can accept more than one file as an argument, it can also be used to join files together. Say we have downloaded a large file that has been split into multiple parts (multimedia files are often split this way on Usenet), and we want to join them back together. If the files were named:

movie.mpeg.001 movie.mpeg.002 ... movie.mpeg.099

we could rejoin them with this command:

```
chirag@chirag-VirtualBox:~$ cat movie.mpeg.0* > movie.mpeg
```

If cat is not given any arguments, it reads from standard input, and since standard input is, by default, attached to the keyboard, it's waiting for us to type something.

```
chirag@chirag-VirtualBox:~$ cat
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
```

In the absence of filename arguments, cat copies standard input to standard output, so we see our line of text repeated. We can use this behavior to create short text files(by redirecting cat's output to a file).

```
chirag@chirag-VirtualBox:~$ cat >  lazy_dog.txt
The quick brown fox jumped over the lazy dog.
chirag@chirag-VirtualBox:~$ cat lazy_dog.txt
The quick brown fox jumped over the lazy dog.
chirag@chirag-VirtualBox:~$ cat < lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

Using the **<** redirection operator, we change the source of standard input from the keyboard to the file lazy_dog.txt.

sort

The sort command is used to sort a list of data.

Example

```
chirag@chirag-VirtualBox:~$ cat names
Chirag
Amar
Bharat
chirag@chirag-VirtualBox:~$ sort names
Amar
Bharat
Chirag
```

uniq

uniq accepts a sorted list of data from either standard input or a single filename argument and, by default, removes any duplicates from the list. If we want to see the list of duplicates instead, we add the -d option to uniq.

Example

```
chirag@chirag-VirtualBox:~$ cat > names
Chirag
Amar
Chirag
Avinash
chirag@chirag-VirtualBox:~$ sort names | uniq
Amar
Avinash
Chirag
chirag@chirag-VirtualBox:~$ sort names | uniq -d
Chirag
```

wc

The wc (word count) command is used to display the number of lines, words, and bytes contained in files.The -l option limits its output to only report lines.

```
chirag@chirag-VirtualBox:~$ cat > names
Chirag Singh
Amar Maurya
Avinash Yadav
chirag@chirag-VirtualBox:~$ wc names
 3  6 39 names
chirag@chirag-VirtualBox:~$ ls /usr/bin /bin | sort | uniq | wc -l
1601
```

grep

grep is a powerful program used to find text patterns within files.

Syntax

grep *pattern files...*

Examples

```
chirag@chirag-VirtualBox:~$ ls -l | grep ^d
drwxrwxr-x 10 chirag chirag 4096 Feb 18 17:40 dir1
drwxrwxr-x 12 chirag chirag 4096 Feb 18 17:41 dir2
chirag@chirag-VirtualBox:~$ ls /usr/bin /bin | sort | uniq | grep zip
bunzip2
bzip2
bzip2recover
funzip
```

grep Options

Option	**Result**
-w	Match whole words only.
-i	Case-insensitive search.
-n	Print line numbers where matches are found.
-B *n*	Print *n* lines before the match.
-A *n*	Print *n* lines after the match.
-C *n*	Print *n* lines before and after the match
-r	Recursive search in a directory and it's sub-directories.
-l	List only the file names that contains the match.
-c	List only the file names and number of matches in the files that contains the match.
-P	Use perl flavour Regex.

head / tail

The head command prints the first 10 lines of a file, and the tail command prints the last 10 lines. By default, both commands print 10 lines of text, but this can be adjusted with the -n option.

Examples

```
chirag@chirag-VirtualBox:~$ ls /usr/bin | tail -n 3
zipsplit
zjsdecode
zlib-flate
```

tail has an option that allows you to view files in real time. This is use ful for watching the progress of log files as they are being written.Using the -f option, tail continues to monitor the file and when new lines are appended, they immediately appear on the display.

```
chirag@chirag-VirtualBox:~$ tail -f /var/log/syslog
Feb 18 18:01:46 chirag-VirtualBox systemd[1]: Started Run anacron jobs.
Feb 18 18:01:46 chirag-VirtualBox anacron[14939]: Anacron 2.3 started on 2020-
02-18
Feb 18 18:01:46 chirag-VirtualBox anacron[14939]: Normal exit (0 jobs run)
...
```

---

tee

The tee program reads standard input and copies it to both standard output (allowing the data to continue down the pipeline) and to one or more files. This is useful for capturing a pipeline's contents at an intermediate stage of processing.

Example

```
chirag@chirag-VirtualBox:~$ ls /usr/bin | tail -10 | tee last-10-cmd | gerp zi
p
chirag@chirag-VirtualBox:~$ ls /usr/bin | tail -10 | tee last-10-cmd | grep zi
p
zip
zipcloak
zipdetails
...
chirag@chirag-VirtualBox:~$ cat last-10-cmd
zenity
zip
zipcloak
...
```

---

echo

echo is a shell builtin that performs a very simple task: It prints out its text arguments on standard output.

Expansion

With expansion, you enter something, and it is expanded into something else before the shell acts upon it. for example * , can have a lot of meaning to the shell. The process that makes this happen is called *expansion*.

- **Pathname Expansion**

  The mechanism by which wildcards work is called pathname expansion. The shell expands the * into the names of the files in the current working directory before the echo command is executed.

  ```
  chirag@chirag-VirtualBox:~$ echo *
  ```

```
abc dir1 dir2 Documents Downloads error.txt files last-10-cmd
...
chirag@chirag-VirtualBox:~$ echo D*
Documents Downloads
chirag@chirag-VirtualBox:~$ echo [[:upper:]]*
Documents Downloads Music Pictures Public Templates Videos
chirag@chirag-VirtualBox:~$ echo /usr/*/share
/usr/local/share
```

- **Tilde Expansion**

  When ~ is used at the beginning of a word, it expands into the name of the home directory of the named user or, if no user is named, the home directory of the current user.

```
chirag@chirag-VirtualBox:~$ echo ~
/home/chirag
chirag@chirag-VirtualBox:~$ echo ~foo
/home/foo
```

- **Arithmetic Expansion**

  The shell allows arithmetic to be performed by expansion. This allows us to use the shell prompt as a calculator.Arithmetic expansion uses the following form:

  $((expression))

```
chirag@chirag-VirtualBox:~$ echo $((2 + 2))
4
chirag@chirag-VirtualBox:~$ echo $(($((3 ** 2)) * 2 + 2))
20
chirag@chirag-VirtualBox:~$ echo 5 Squared Is $((5 ** 2))
5 Squared Is 25
```

- **Brace Expansion**

  With Brace Expansion, you can create multiple text strings from a pattern containing braces.

```
chirag@chirag-VirtualBox:~$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
chirag@chirag-VirtualBox:~$ echo num{1..5}
num1 num2 num3 num4 num5
chirag@chirag-VirtualBox:~$ echo {z..a}
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

- **Parameter Expansion**

  Environment Variables are expanded using Parameter Expansion. For example, the variable named USER contains your username. The *printenv* command is used to list all environment variables.

```
chirag@chirag-VirtualBox:~$ echo $USER
chirag
```

- **Command Substitution**

  Command substitution allows us to use the output of a command as an expansion. Command Substitution uses the following form:

  $(command) or ***\`command\`***

  ```
  chirag@chirag-VirtualBox:~$ echo $(ls)
  abc dir1 dir2 Documents Download
  chirag@chirag-VirtualBox:~$ ls -l $(which cp)
  -rwxr-xr-x 1 root root 141528 Jan 18  2018 /bin/cp
  chirag@chirag-VirtualBox:~$ echo `ls`
  abc dir1 dir2 Documents Downloads
  ```

Quoting

By quoting, we can control shell expansions in cases such as the following:

```
chirag@chirag-VirtualBox:~$ echo this is a     test
this is a test
chirag@chirag-VirtualBox:~$ echo The total is $100.00
The total is 00.00
```

In the first example, *word splitting* by the shell removed extra whitespace from the echo command's list of arguments. In the second example, parameter expansion substituted an empty string for the value of $1 because it was an undefined variable. The shell provides a mechanism called quoting to selectively suppress unwanted expansions.

- **Double Quoting**

  If you place text inside double quotes, all the special characters used by the shell lose their special meaning and are treated as ordinary characters. The exceptions are $, \, and \`.haracters.This means that word splitting, pathname expansion, tilde expansion, and brace expansion are suppressed, but parameter expansion, arithmetic expansion, and command substitution are still carried out.

  ```
  chirag@chirag-VirtualBox:~$ echo "This is a      test"
  This is a      test
  chirag@chirag-VirtualBox:~$ echo "2 + 2 is $((2 + 2))"
  2 + 2 is 4
  chirag@chirag-VirtualBox:~$ echo $(cal)
  February 2020 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1
  6 17 18 19 20 21 22 23 24 25 26 27 28 29
  chirag@chirag-VirtualBox:~$ echo "$(cal)"
     February 2020
  Su Mo Tu We Th Fr Sa
                     1
   2  3  4  5  6  7  8
   9 10 11 12 13 14 15
  16 17 18 19 20 21 22
  23 24 25 26 27 28 29
  ```

- **Single Quoting**

If we need to suppress all expansions, we use single quotes.

```
chirag@chirag-VirtualBox:~$ echo text ~/e*.txt {a..c} $(echo foo) $((2 +
 2)) $USER
text /home/chirag/error.txt a b c foo 4 chirag
chirag@chirag-VirtualBox:~$ echo "text ~/e*.txt {a..c} $(echo foo) $((2
+ 2)) $USER"
text ~/e*.txt {a..c} foo 4 chirag
chirag@chirag-VirtualBox:~$ echo 'text ~/e*.txt {a..c} $(echo foo) $((2
+ 2)) $USER'
text ~/e*.txt {a..c} $(echo foo) $((2 + 2)) $USER
```

- **Escaping Characters**

  Sometimes we want to quote only a single character. To do this, we can precede a character with a backslash, which in this context is called the *escape character*.

```
chirag@chirag-VirtualBox:~$ echo The total is \$100.00
The total is $100.00
```

---

id

To find out information about your identity, use the id command.

Example

```
chirag@chirag-VirtualBox:~$ id
uid=1000(chirag) gid=1000(chirag) groups=1000(chirag),4(adm),24(cdrom),27(sudo
),30(dip),46(plugdev),118(lpadmin),128(sambashare)
```

When user accounts are created, users are assigned a number called a user ID, or uid. This is then mapped to a username. The user is assigned a primary group ID, or gid, and may belong to additional groups.

---

chmod

To change the mode (permissions) of a file or directory, the chmod command is used. Be aware that only the file's owner or the superuser can change the mode of a file or directory.

Permission Attributes

Attributes	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is	Allows files within a directory to be created, deleted, and renamed if the

	determined by directory attributes.	execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered; e.g., *cd directory*

chmod supports two distinct ways of specifying mode changes:

- **Octal Representation**

  With octal notation we use octal numbers to set the pattern of desired permissions. Since each digit in an octal number represents three binary digits, this maps nicely to the scheme used to store the file mode.

  ```
  chirag@chirag-VirtualBox:~$ ls -l foo
  -rw-rw-r-- 1 chirag chirag 0 Feb 19 01:15 foo
  chirag@chirag-VirtualBox:~$ chmod 600 foo
  chirag@chirag-VirtualBox:~$ ls -l foo
  -rw------- 1 chirag chirag 0 Feb 19 01:15 foo
  ```

- **Symbolic Representation**

  chmod also supports a symbolic notation for specifying file modes. Symbolic notation is divided into three parts: whom the change will affect, which operation will be performed, and which permission will be set.

Symbol	Meaning
u	Short for user but means the file or directory owner.
g	Group owner.
o	Short for others, but means world.
a	Short for all; the combination of u, g, and o.
+	Indicates that a permission is to be added.
-	Indicates that a permission is to be taken away.
=	Indicates that only the specified permissions are to be applied and that all others are to be removed.

  Examples

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. Equivalent to a+x .
o-rw	Remove the read and write permissions from anyone besides the owner and group owner.
	Set the group owner and anyone besides the owner to have read and write

go=rw	permission. If either the group owner or world previously had execute permissions, remove them.
u+x,go=rx	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

umask

The umask command controls the default permissions given to a file when it is created. It uses octal notation to express a mask of bits to be removed from a file's mode attributes. umask cannot be used to give executable permissions. The default umask is 0002.

Original file mode	--- rw- rw- rw-
Mask	000 000 010
Result	--- rw- rw- r--

Therefore, whenever a new file created it will be assigned the permissions same as the result. We can also change the umask.

```
chirag@chirag-VirtualBox:~$ umask
0002
chirag@chirag-VirtualBox:~$ > file; ls -l file
-rw-rw-r-- 1 chirag chirag 0 Feb 19 02:20 file
chirag@chirag-VirtualBox:~$ umask 0066
chirag@chirag-VirtualBox:~$ rm file; > file; ls -l file
-rw------- 1 chirag chirag 0 Feb 19 02:20 file
```

su

The su command is used to start a shell as another user. The command syntax looks like this:

su [-[l]] [user]

If the -l option is included, the resulting shell session is a login shell for the specified user. This means that the user's environment is loaded and the working directory is changed to the user's home directory. This is usually what we want. If the user is not specified, the superuser is assumed. -l can be abbreviated with only -. By default the superuser account is disable. You can create one using the following commands:

```
chirag@chirag-VirtualBox:~$ sudo passwd root
[sudo] password for chirag:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
chirag@chirag-VirtualBox:~$ su -
Password:
root@chirag-VirtualBox:~# exit
logout
chirag@chirag-VirtualBox:~$
```

It is also possible to execute a single command rather than starting a new interactive command by using su this way:

```
chirag@chirag-VirtualBox:~$ su -c 'ls -l /root/*'
Password:
-rw-r--r-- 1 root root 335 Feb  4 14:19 /root/10-monitor.conf
```

It is important to enclose the command in quotes, as we do not want expansion to occur in our shell but rather in the new shell.

---

sudo

The sudo command is like su in many ways but has some important additional capabilities. The administrator can configure sudo to allow an ordinary user to execute commands as a different user (usually the superuser) in a very controlled way. In particular, a user may be restricted to one or more specific commands and no others. Another important difference is that the use of sudo does not require access to the superuser's password. To authenticate using sudo, the user enters his own password.

One important difference between su and sudo is that sudo does not start a new shell, nor does it load another user's environment. This means that commands do not need to be quoted any differently than they would be without using sudo.

```
chirag@chirag-VirtualBox:~$ sudo ls -l /root/
total 4
-rw-r--r-- 1 root root 335 Feb  4 14:19 10-monitor.conf
```

---

chown

The chown command is used to change the owner and group owner of a file or directory. Superuser privileges are required to use this command. The syntax of chown looks like this:

chown [owner][:[group]] file...

Examples

Arguments	Results
bob	Changes the ownership of the file from its current owner to user bob.
bob:users	Changes the ownership of the file from its current owner to user bob and changes the file group owner to group users.
:admins	Changes the group owner to the group admins. The file owner is unchanged.
bob:	Change the file owner from the current owner to user bob and changes the group owner to the login group of user bob.

```
root@chirag-VirtualBox:~# cp file.txt ~chirag
root@chirag-VirtualBox:~# ls -l ~chirag/file.txt
```

```
-rw-r--r-- 1 root root 0 Feb 19 02:56 /home/chirag/file.txt
root@chirag-VirtualBox:~# chown chirag: ~chirag/file.txt
root@chirag-VirtualBox:~# ls -l ~chirag/file.txt
-rw-r--r-- 1 chirag chirag 0 Feb 19 02:56 /home/chirag/file.txt
```

passwd

To set or change a password, the passwd command is used. To change your own password just omit the user parameter.

Syntax

passwd *[user]*

ps

The most commonly used command to view processes is ps.

```
chirag@chirag-VirtualBox:~$ ps
  PID TTY          TIME CMD
20699 pts/4    00:00:00 bash
20751 pts/4    00:00:00 ps
```

Field	Meaning
PID	Each process is assigned a number called a process ID (PID). PIDs are assigned in ascending order, with init always getting PID 1.
TTY	TTY is short for teletype and refers to the controlling terminal for the process.
TIME	The TIME field is the amount of CPU time consumed by the process.
CMD	Process name.

Adding the x option (note that there is no leading dash) tells ps to show all of our processes regardless of what terminal (if any) they are controlled by. The presence of a ? in the TTY column indicates no controlling terminal. Using this option, we see a list of every process that we own.

```
chirag@chirag-VirtualBox:~$ ps x
  PID TTY      STAT   TIME COMMAND
  865 ?        Ss     0:00 /lib/systemd/systemd --user
  866 ?        S      0:00 (sd-pam)
  885 ?        Sl     0:01 /usr/bin/gnome-keyring-daemon --daemonize --login
  ...
```

Process STAT / States

State	Meaning
R	Running. The process is running or ready to run.

S	Sleeping. The process is not running; rather, it is waiting for an event, such as a keystroke or network packet.
D	Uninterruptible sleep. Process is waiting for I/O such as a disk drive.
T	Stopped. Process has been instructed to stop.
Z	A defunct or “zombie” process. This is a child process that has terminated but has not been cleaned up by its parent.
<	A high-priority process. It’s possible to grant more importance to a process, giving it more time on the CPU. This property of a process is called niceness. A process with high priority is said to be less nice because it’s taking more of the CPU’s time, which leaves less for everybody else.
N	A low-priority process. A process with low priority (a nice process) will get processor time only after other processes with higher priority have been serviced.

Another popular set of options is aux (Using the options without the leading dash invokes the command with “BSD-style”).

```
chirag@chirag-VirtualBox:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 225176  5524 ?        Ss   Feb18   0:05 /sbin/init sp
lash
root         2  0.0  0.0      0     0 ?        S    Feb18   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   Feb18   0:00 [rcu_gp]
...
```

BSD-Style ps Column Headers

Header	**Meaning**
USER	User ID. This is the owner of the process.
%CPU	CPU usage as a percent.
%MEM	Memory usage as a percent.
VSZ	Virtual memory size.
RSS	Resident Set Size. The amount of physical memory (RAM) the process is using in kilobytes.
START	Time when the process started. For values over 24 hours, a date is used.

top

To see a more dynamic view of the machine’s activity, we use the top command. The top program displays a continuously updating (by default, every 3 seconds) display of the system processes listed in order of process activity.

```
top - 04:05:35 up 13:57,  1 user,  load average: 0.33, 0.52, 0.50
Tasks: 172 total,   2 running, 139 sleeping,   0 stopped,   0 zombie
top - 04:09:31 up 14:00,  1 user,  load average: 0.63, 0.53, 0.51
Tasks: 172 total,   1 running, 140 sleeping,   0 stopped,   0 zombie
```

```
%Cpu(s): 21.7 us,  2.0 sy,  0.0 ni, 75.8 id,  0.5 wa,  0.0 hi,  0.0 si,  0.0 s
t
KiB Mem :  2038764 total,   591788 free,  1098920 used,   348056 buff/cache
KiB Swap:   969960 total,   459616 free,   510344 used.   786036 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1410 chirag    20   0 1289188 230760  45620 S 12.5 11.3  26:22.26 code
 1183 chirag    20   0 1416452  56868  26980 S  6.2  2.8   9:16.68 gala
 1366 chirag    20   0 1260776  79356  35196 S  6.2  3.9   5:18.13 code
 1400 chirag    20   0  624500  37520  19760 S  6.2  1.8  10:35.05 code
```

jobs

To launch a program so that it is immediately placed in the back- ground, we follow the command with an ampersand character (&):

```
chirag@chirag-VirtualBox:~$ xlogo &
[1]+  Done                    xlogo
[1] 21443
```

After the command was entered, the xlogo window appeared and the shell prompt returned, but some numbers were printed too. This message is part of a shell feature called *job control*. With this message, the shell is telling us that we have started job number 1 ([1]) and that it has PID 21443. The shell's job control facility also gives us a way to list the jobs that have been launched from our terminal. Using the *jobs* command, we can see the following list:

```
chirag@chirag-VirtualBox:~$ jobs
[1]+  Running                 xlogo &
```

fg

A process in the background is immune from keyboard input, including any attempt to interrupt it with a CTRL-C. To return a process to the foreground, use the *fg* command.

```
chirag@chirag-VirtualBox:~$ jobs
[1]+  Running                 xlogo &
chirag@chirag-VirtualBox:~$ fg %1
xlogo
^C
```

The command fg followed by a percent sign and the job number (called a jobspec). If we have only one background job, the jobspec is optional.

bg

Sometimes we'll want to stop a process without terminating it. This is often done to allow a foreground process to be moved to the background. To stop a foreground process, type CTRL-Z.

After stopping a process, we can either restore the program to the foreground, using the fg command, or move the program to the background with the *bg* command:

```
chirag@chirag-VirtualBox:~$ xlogo
^Z
[1]+  Stopped                 xlogo
chirag@chirag-VirtualBox:~$ jobs
[1]+  Stopped                 xlogo
chirag@chirag-VirtualBox:~$ bg %1
[1]+ xlogo &
chirag@chirag-VirtualBox:~$ jobs
[1]+  Running                 xlogo &
```

---

kill

The kill command is used to “kill” (terminate) processes. This allows us to end the execution of a program that is behaving badly or otherwise refuses to terminate on its own.The kill command doesn’t exactly “kill” processes; rather it sends them signals. Signals are one of several ways that the operating system communicates with programs.

Syntax

kill *[-signal] PID...*

Examples

```
chirag@chirag-VirtualBox:~$ xlogo &
[2] 21744
chirag@chirag-VirtualBox:~$ jobs
[2]+  Running                 xlogo &
chirag@chirag-VirtualBox:~$ kill -9 21744
[2]+  Killed                  xlogo
```

Pipelines

The ability of commands to read data from standard input and send to standard output is utilized by a shell feature called pipelines. Using the pipe operator | (vertical bar), the standard output of one command can be piped into the standard input of another.

command1 | command2

```
chirag@chirag-VirtualBox:~$ ls -l /usr/bin | less
```

Filters

It is possible to put several commands together into a pipeline. Frequently, the commands used this way are referred to as filters. Filters take input, change it somehow, and then output it.

```
chirag@chirag-VirtualBox:~$ ls -l dir1 dir2 | sort | less
```

The output of ls would have consisted of two sorted lists, one for each directory. By including sort in our pipeline, we changed the data to produce a single, sorted list.

Redirection

- Redirecting Standard Error

  ```
  chirag@chirag-VirtualBox:~$ ls /bin/usr 2> error.txt
  ```

- Redirecting Standard Output and Standard Error to One File

  ```
  chirag@chirag-VirtualBox:~$ ls /bin/usr > error.txt 2>&1
  chirag@chirag-VirtualBox:~$ ls /bin/usr &> error.txt
  ```

- Disposing of Unwanted Output

  The system provides a way to do this by redirecting output to a special file called /dev/null. This file is a system device called a ***bit bucket***, which accepts input and does nothing with it.

  ```
  chirag@chirag-VirtualBox:~$ ls /bin/usr 2> /dev/null
  ```

Tricks

- To truncate a file (or create a new, empty file) we can use:

  ```
  chirag@chirag-VirtualBox:~$ > names.txt
  ```

- List only hidden files.

  ```
  chirag@chirag-VirtualBox:~$ ls -d .[!.]?*
  ```

- List only directories.

  ```
  chirag@chirag-VirtualBox:~$ ls -F | grep "/$"
  chirag@chirag-VirtualBox:~$ ls -d */
  chirag@chirag-VirtualBox:~$ ls -l | grep "^d"
  ```

Wildcards

Wildcard	Matches
*	Any character

?	Any single character
[characters]	Any character that is the member of set *characters*
[!characters]	Any character that is not the member of set *characters*
[[:class:]]	Any character that is the member of specified *class*

Charcter Classes

Character Class	**Matches**
[:alnum:]	Any alphanumeric character
[:alpha:]	Any alphabetic character
[:digit:]	Any numeral
[:lower:]	Any lowercase letter
[:upper:]	Any uppercase letter

Linux File Types

Attribute	**File Type**
-	A regular file.
l	A directory.
d	A symbolic link. Notice that with symbolic links, the remaining file attributes are always rwxrwxrwx and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A character special file. This file type refers to a device that handles data as a stream of bytes, such as a terminal or modem.
b	A block special file. This file type refers to a device that handles data in blocks, such as a hard drive or CD-ROM drive.