The Command line interface – a quick view

Toshaan Bharvani

The terminal, sometimes also referred to as the *shell*, is a program that allows a user to input commands with the keyboard and gives these commands to the operating system to be executed. Originally this was the only way to communicate with the operating system, however nowadays the graphical interface is more used.

For many users the command-line interface is considered user unfriendly, however the command-line interface does provide the user with many advantages, such as:

- easy information over low bandwidth connections,
- more configuration options then the graphical wizards,
- quicker access to most configuration files,
- possibility to add comments to configuration files

There are two disadvantages to the command-line interface, being firstly you need to know what you are doing, secondly, some mistakes can be fatal to your system.

CentOS comes standard with *gnome-terminal*, as the default terminal emulator, because it is the standard terminal window for Gnome. However, some people prefer other emulators, CentOS third party repositories provide some other terminal emulators. xterm is one of the other more popular terminal emulators. To launch your favourite terminal emulator, press [*Alt*] + [*F2*] and type *gnome-terminal* or *xterm*.

Before going into more details regarding some of the commands, some basic rules:

- Linux is CaSe-SeNsITiVe, this means that all input requires to be in the correct case, *gnome-terminal* and *Gnome-Terminal* are two completely different commands, the user *toshaan* and *Toshaan* are two different users. Therefore all commands are considered to be in lower-case, unless specified differently.
- Filenames should be preferably up to 256 characters without spaces or language specific characters. To include spaces you will need to escape to include spaces, so a folder with name *toshaanbharvani* will be addressed as it is written, however *Toshaan Bharvani* need to be addressed as *Toshaan\ Bharvani*.
- For DOS or Windows people, "\" equals "/" in Linux and there are no drive letters, just block device names, such as /dev/sda with partition numbers, such as /dev/sda1 and mount points like / (root directory), all other drives will be mounted within the root directory structure, for example /home can be mounted by /dev/sda2.

- Any line starting with "#" is considered as a comment, in some cases ";" is also used.

Basic commands

Change directory

```
cd /path/to/where/you/want/ ↵
   to/go
```

Display directory content

```
ls -la /path/to/display
```

Displays the content of the directory */path/to/display*. If no path is provided, it will display the content of the current directory. The option *-l* will display the complete listing and the option *-a* displays all files, including hidden ones.

Make a directory

```
mkdir -p /path/to/new/ ↵
   directory
```

The creation of the path */path/to/new/directory* will be made, however to ensure that even if the upper path is available, you add the parent option *-p*. This will enable mkdir to make the directories */path/to/new*, which are higher than */path/to/new/directory* and are required to create the directory */path/to/new/directory*.

File movement

```
cp -rv /path/to/source /path/ ↵
   to/destination

mv -v /path/to/source /path/ ↵
   to/destination
```

Display disk space usage

```
df -h
```

This command displays the disk space usage, with the *-h* option, it converts the values into human readable numbers.

Display directory space usage

```
du -hs /path/to/display
```

The command displays the space usage of the path */path/to/display*. The option *-h* displays the results in human readable numbers and the option -s summarizes all files and subfolders size into one figure.

Hardware report tool

```
dmidecode
```

Outputs hardware information about the system, it can be useful when you need to check the latest BIOS version with installed version, how many DIMM slots are available. The output of this command will span over a long buffer, sometimes more than what can be scrolled, therefore adding | *less* to the end of the command buffers the output into pages, making it easier to navigate and read.

Display system messages

```
dmesg
```

This command outputs the kernel messages. Again this output may be longer than what you can manage to read, adding | *less* to the end of the command buffers the output into pages, adding *emph{| tail*} displays only the latest messages.

Display files on screen

```
cat /path/to/file
```

`cat` displays the contents of the */path/to/file* on the screen buffer. Again adding | *less* to the end will buffer the output into pages. However, it might be easier to just use less.

Screen output pager

```
less /path/to/file
```

`less` enables you to output the file into page buffers, which are scrollable. Adding *less* to the end of another buffer output command by use of | *less* buffers the original command, then lets less page the output and displays it on the screen.

```
more /path/to/file
```

`more` is the opposite of `less` in that way, that it functions in the same way as `less`, but only provides page down scrolling.

Text search tool

```
grep <text pattern> /path/to/ ↵
   file
```

`grep` is a text search tool for the command-line. It allows users to search for text patterns in files or folders.

Kill an application

```
kill <applications process id>

killall <application name>
```

Both the `kill` and `killall` commands terminate an application. The `kill` command uses the process id of an application, whereas the `killall` uses the application name to kill the application.

Process overview

```
ps aux | grep <appliction
   name>
```

This will display the information of the application such as the user, process ID, cpu usage, memory usage...

```
top
```

This command displays all the Linux tasks running on the system and is installed by default. It gives a realtime overview of the running processes with process ID (pid), user, cpu percentage and memory percentage usage.

```
htop
```

`htop` is a ncurses based process viewer, like top. It is more colourful and provides somewhat the same information as top, but with allows you to scroll through the information.

Locate files

```
locate <name>
```

`locate` is a command to find files or folders based on a pre-cached database. The database is queried and whatever matches the pattern will be displayed.

```
whereis <application name>
```

`whereis` locates the binary file, source file and the man page file of certain applications.

Mounting locations

Mount a partition to a location:

```
mount /dev/blockname /mnt/drive
```

Mount an .iso file to a location:

```
mount -o loop /path/to/ ↵
   file.iso /mnt/cdrom
```

Mount a CIFS compatible share to a location:

```
mount -t cifs //ipaddress/ ↵
   sharename /mnt/share
```

nohup

```
nohup <command> &
```

The `nohup` command tells the executed command not to keep account with the hangup (HUP) signal and therefore allows the command to run, even when the user would get disconnected from the shell.

screen

```
screen
```

`screen` is a terminal multiplexer allowing multiple terminal sessions to run inside a single terminal. When the option *-ls* is added, all local screens are displayed.

The option *-S <name>* creates a screen with the given name and option *-r* reattaches to an existing screen. To logout of a screen type [*Ctrl*] + [*d*], to detach: [*Ctrl*] + [*a-d*].

Remote connection

Secure SHell

```
ssh -p <port number> ↵
   <username>@<server address>
```

SSH is a remote secure shell connection between a client machine and a server. The user gets shell access like he would have if he were in front of the computer.

Port forwarding with SSH

```
ssh -L <local port ↵
   number>:<host>:<remote port ↵
   number><username>@<server ↵
   address>
```

This creates a secure shell with port forwarding enabled. The *<local port number>* indicates the port number that the service will become available on the local system. The *<remote port number>* and the *<host>* are the connections on the remote side to be executed while being connected to the server with *<server address>*.

```
ssh -R <local port ↵
   number>:<host>:<remote port ↵
   number><username>@<server ↵
   address>
```

This creates a secure shell with port forwarding enabled. The *<local port number>* indicates the port number that the service will receive data on the server. The *<remote port number>* indicates the port that the data will be received on the client.

```
ssh -L 123:localhost:12 ↵
   administrator@192.168.1.1
```

This will make local port 123 connect to port 12 on 192.168.1.1

```
ssh -R 123:localhost:12 ↵
   administrator@192.168.1.1
```

This will make port 123 on server 192.168.1.1; listen to the localhost and forward it to the local port 12.

Secure Copy

```
scp -P <remote port> <source> ↵
   <target>
```

`scp` creates a secure file transfer shell, allowing the user to copy files from the source location to the target location. The syntax for the source and target is the same and depends on whether the location point locally or remotely. For local locations, the traditional syntax */path/to/file* applies. However, for remote locations, the syntax is as follows:

```
username@remotehost:/path/to/ ↵
   file
```

Thus `scp` allows you to copy files from one remote location to another remote location. The option *-P* (this is a capital P) sets the remote port.

Midnight Commander

Midnight Commander is command-line file manager inspired by Norton Commander.

```
mc
```

The screen is divided into two folder sections. When you type the below command, `mc` will connect to the remote server using the Files transferred over SHell protocol (FISH) protocol:

```
cd /#sh:remoteuser@ ↵
   remotemachine:/path/to/ ↵
   location
```

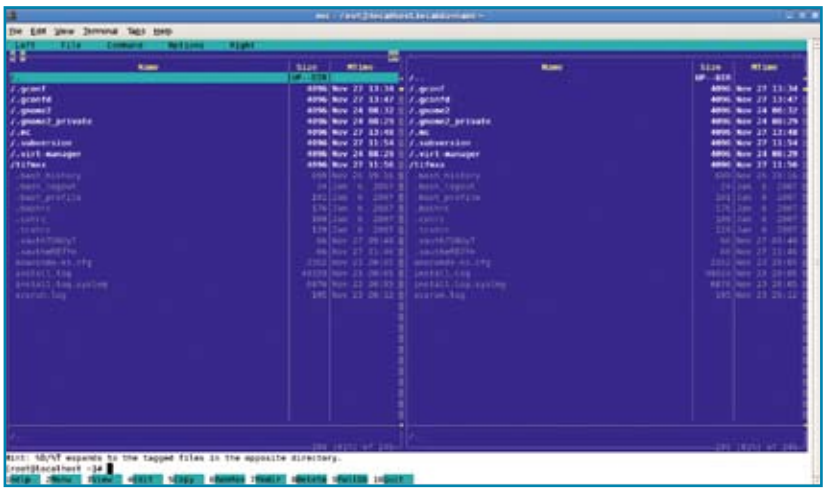

Figure 1. *Midnight Commander*