LINUX

Beginner's Crash Course

Linux for Beginner's Guide to
Linux Command Line,
Linux System & Linux Commands

QUICK START GUIDES

Linux

Beginner’s Crash Course

Linux for Beginner’s Guide to Linux Command Line, Linux System & Linux Commands

Table of Contents

Introduction

Linux was once thought of as an underdog operating system, and for some time it remained in obscurity overshadowed by the mighty Windows. But more recently, Linux has been winning accolades among software developers and businesses. This Unix variant, once only known to geeks and nerds, is now gaining popularity among ordinary home computer users, too.

Linux is free of charge and runs on different platforms. It has been amassing a dedicated user base and is increasingly drawing interest from:

- People who are acquainted with Unix and interested in running it on personal computers
- People who are interested in experimenting with the principles of operating systems
- People who want an operating system which gives them maximum control
- People who are not satisfied with Microsoft Windows

Managing Linux is more difficult than using Windows, but Linux is more flexible and has more configuration options.

Corporations also seem to be conquering their irrational suspicion of Linux and are adapting Linux for many business applications. Because Linux is open source, companies need not worry about licensing when making such adaptations. That makes it a pretty attractive proposition.

Chapter 1: Linux & Other Operating Systems

If you are a computer programmer you already know about Linux. If not, you may have read about it or heard about it from one of your friends. Either way, Linux is interesting, and the more you learn about Linux, the more interesting questions you'll have about it.

What is Linux?

In general, there are two types of PC users. Most people look at the computer as an amazing tool for cool stuff like browsing, playing games, watching movies and listening to music. They might even know about the hardware components like RAM, processors, and audio and video cards. But they don’t really care about the complexity behind these simple processes. They open a browser, type the address and somehow the website opens. They are not interested in knowing how it happens.

Then there is a second set of people who are actually interested in knowing how things work internally. Almost everyone considers an operating system as additional software. It is not. An operating system, just like hardware components, is an important part of a computer. And when needed, the operating system can be upgraded or swapped for a more suitable one.

Advantages of Linux

You might think that people who talk about Linux are exaggerating how awesome it is, but in fact it does have a number of real advantages over other operating systems. These include the following:

Crash free

Linux is a crash-free operating system. When you're using Linux, you don't experience freezing of the cursor on the screen, sudden system crashes or error messages that won't go away until you reboot your system. The programs or applications that run on the Linux operating system might crash, but the operating system will still be functioning. No application can take it down when it crashes. In Linux, unlike other operating systems, you can simply clean up after the program crash and carry on. The crash-free reliability of Linux is a big asset when dealing with operations like spaceflight and nuclear power where you cannot risk system crashes which might result in huge expenses, injury or death.

Security

The next advantage of Linux is its security. The security of the Linux operating system is far better than Windows, which has many security flaws. The reason for this is the years of extensive computer science research that have gone into Linux development. In Linux, if you do not have the appropriate permissions, you will not be allowed to access a particular piece of hardware or a particular folder in the system. Privacy can be set to individual owners who can only access their own files. The root user will have access to the complete system and can limit access for other users.

Free & shareable

Another major benefit of Linux is that it is a free operating system. Updates will be downloaded free of charge, and when a new version of the operating system becomes available, you can simply replace your current version at no additional cost. In fact, you can actually download any version of Linux you want. As it is free software, it can be shared among a group of people. Default applications like image viewers, music players, image editors, etc. can all be shared. This is not possible with the Windows or most other operating systems, wherein copying these kinds of applications and sharing them with others will make you a software pirate. But with Linux, it is completely legal – and actually encouraged – to share software with other users.

Compatibility
Another good thing about Linux is that it can work even with older hardware. You don't need a high performance, cutting edge PC like you do with the latest versions of other operating systems. For example, the new version of Windows will need new hardware for it to run. The Linux operating system can work even on obsolete hardware. So you won't need to buy a new computer if you want to run Linux. You can see the difference for yourself if you install Linux and any other operating system on the same hardware. The Linux operating system will work faster and better than power-hungry operating systems like Windows. The Linux group encourages users to recycle and reuse components to make the most of existing hardware. You can actually bring an old computer system back to life by installing Linux on it. Even if you don't need it, you can give it to a friend or family member who doesn't own a PC.

Flexibility
The Linux operating system is extremely flexible. Linux works on almost 60% of computers and can use broadband Internet connections. You can turn your old hardware into a web server, email server, firewall or anything else you have in mind. Doing this using other operating systems would cost you a fortune because you would need a seriously high-performance computer.

The Linux community
But the best feature of Linux is undoubtedly the Linux community. Linux is not just a computer operating system. When you start using Linux, you become a part of a huge community that is spread across the globe. Being a part of the Linux community has its own advantages. One of these is that you are never far from finding a solution for whatever problem you may have. Most of the people in this community are Linux evangelists and love to help others solve their problems. New members of the Linux community are known as newbies, which sounds derisory but is actually a good opportunity to learn from professional Linux programmers. These programmers believe that knowledge should be shared freely and are glad to help other people learn Linux. When someone advertises his status as a newbie, people will

make time to help him. After all, those professionals were once newbies as well.

Features of Operating Systems

An operating system can be considered as an interface or intermediary between the computer hardware and the user. Operating systems provide users with an environment that allows them to execute different programs efficiently and conveniently. Technically speaking, an operating system is software that manages the hardware on which it is being run and controls the execution of all kinds of programs. An operating system is responsible for the allocation of services and resources like processors, memory, devices and information. Linux is one such operating system. The important functions of any operating system are given below:

- Memory management
- Device management
- Processor management
- Security
- File management
- Control over system performance
- Error detection
- Job accounting
- Coordination between users & other software

Memory management

Memory management is managing the main or primary memory. Memory can be considered as a very large array of bytes or words that has an address of its own.

The main memory provides the CPU with direct access and faster storage. So if you want to execute a program, it should be stored in the main memory of the computer. Every operating system, including Linux, performs the following memory management activities:

- Keeping track of the main memory, including used and unused memory. For used memory, the operating system also tracks which users are using it.
- The operating system will decide how to allocate memory for different processes when multiprogramming.
- Allocating memory whenever a certain process requests extra

memory.

- Operating systems also take care of memory de-allocation when a particular process is terminated or when it no longer needs memory.

Processor management

In a multiprogramming environment, the operating system is the program that decides the processor allocation time for each process. For processor management, every operating system will perform the following activities:

- Keeping track of process status and processor status using the traffic controller program.
- Allocating the processor to a given process.
- Allocating the processor when it is not required.

Device management

Devices communicate with each other using their own drivers, but the OS is responsible for this communication. An operating system's device manager performs the following activities:

- Keeping track of all the available devices. The I/O controller is the program responsible for performing this task.
- The device manager is what decides the allocation of a particular device to process. It is also responsible for the allocation of time for which the process needs the device.
- It efficiently allocates devices to processes.
- De-allocation is also handled by the device manager.

File management

For easy usage and navigation, file systems are organized into directories in which files and other directions are stored. The operating system performs the following file management activities:

- It keeps track of file status, uses, location, information etc. The file system is a collective of these facilities.
- The operating system's file manager is the program that decides the allocation of resources.
- This also decides who gets to use the resources.
- De-allocation of resources is handled by the file manager.

Other important activities

Here are few more important activities that are performed by an operating system:

- Security – By using a password or similar technique, the operating system will prevent any unauthorized access to data and programs.
- Control over system performance – An operating system will have control over the system performance by recording the delays sent by the service as a request and the response given by the system.
- Job accounting – The operating system is responsible for keeping track of the resources and time used by different users on various jobs.
- Error detecting aids – For error detection, the operating system will produce places, error messages, dumps and a few other errors detecting and debugging aids.
- Coordination between other software and users – The operating system takes care of the coordination and assignment of assemblers, compilers, interpreters and any other software to other users who use the computer system.

Types of operating systems

Operating systems have been in use from the very first generation of computers, and they have continuously evolved over this period of time. Here are a few of the most important types.

Batch operating systems

When using a batch operating system, the user does not interact directly with the computer. Every job that a user prepares is transferred to an off-line device like punch cards and then submitted to the computer operator. For better performance and speed, these jobs are batched together into a group. The computer will run this group of jobs all at once. Programmers give their programs to the operator, and the operator sorts these programs according to their type into batches that have similar requirements. Though this is a good practice, the batch operating system has several disadvantages:

- There is no interaction between the job and the user.
- The CPU is idle most of the time because of the difference in speeds between the CPU and the mechanical I/O devices, which are obviously much slower.
- It can be difficult to provide the properties desired.

Timesharing operating systems

Timesharing is a technique where multiple terminals are located on one computer system and can be used at the same time. Multitasking or timesharing is a logical extension of multiprogramming. The simultaneous sharing of processor time among different users is called timesharing. The main difference between timesharing systems and multiprogramming batch systems is that multiprogramming batch systems aim to maximize processor efficiency, whereas the main objective for timesharing systems is minimizing the response time.

Here, jobs are frequent. They can switch and be executed by the processor, meaning that users can get immediate responses. For example, in transaction processing, the processor executes the programs in the quantum of computation, i.e., in a short burst. The response time for every user is a few seconds at the most.

The operating system uses multiprogramming and scheduling of the CPU to provide each user with a small fraction of time. All computer systems that were originally designed as batch operating systems were later modified into timesharing systems.

The following are some advantages of timesharing operating systems:

- Users get quick responses.
- Software duplication is avoided.
- CPU idle time is reduced.

But timesharing operating systems also have some disadvantages:

- Reliability is low.
- Integrity and security of data and programs is placed at risk.

Distributed operating systems

Distributed operating systems deal with multiple users and multiple real-time applications using multiple CPUs. Efficiency of job performance is increased because data processing tasks are distributed among different processors accordingly.

Processes inter-communicate with each other using various lines like high-speed buses. This can also be referred to as a loosely coupled system. The processors in distributed operating systems are referred to as computers, sites, nodes and so on, and their function and size may vary.

Some advantages of distributed operating systems are given below:

- A resource sharing facility allows users at one site to access and use resources that are present on a different site.
- Data exchange speed is increased using electronic mail.
- Even if one site in a distributed operating system fails, other sites may be unaffected and can potentially continue to operate.
- Customers receive better services.
- The load on the host computer is reduced.
- The data processing delay is reduced.

Network operating systems

A network operating system runs on a server and provides the server with capabilities like data management, group management, user management, application management, security management and a few other networking functions. The network operating system is responsible for file sharing and printer access among all the computers in the network. These networks are usually private networks or local area networks, but can be of any type. Linux can be used as a powerful and efficient network operating system.

The advantages of network operating systems are that:

- The stability of the centralized server is high.
- The server can manage security.
- Any hardware or software updates can be integrated easily into the system.
- It is possible to remotely access the server from a different type of system or a different location.

Disadvantages of network operating systems include:

- Initial costs and maintenance costs are high.
- Servers are dependent on a central location for most operations.
- Frequent updates and regular maintenance are required.

Real-time operating systems

A real-time operating system can be defined as a data processing system where the required time interval for processing and response to inputs is so quick that it can control the environment. All real-time processing is online, but it is not necessary for an online system to be a real-time system. The time taken by a system to respond to an input and to give the output with the updated information is called the response time. When compared to the response time of online processing, the response time in this method is very quick.

There are two different types of real-time operating system:

Hard real-time systems

Hard real-time systems guarantee on-time completion of critical tasks. Secondary storage is absent or limited. Data can be stored in ROM format. There is no virtual memory in hard real-time systems.

Soft real-time systems

Soft real-time systems are less restrictive than hard real-time systems. A soft real-time system prioritizes critical tasks and retains those priorities until each task is completed. When compared to hard real-time systems, soft real-time systems have limited utility. For example, advanced scientific projects like planetary rovers and deep sea exploration can use soft real-time systems, and they also have virtual reality and multimedia applications.

Chapter 2: Components, Features & Architecture of Linux

The Linux operating system is the most popular version of the UNIX operating system. Linux was basically designed to be compatible with UNIX, and the functionalities of both operating systems are very similar.

Components

The Linux operating system has three main components:

- Kernel – The kernel is considered to be the core of the Linux operating system and is responsible for the system's major activities. The kernel interacts directly with the hardware and has various modules within it. The kernel hides all the low-level hardware details by providing the required abstraction to the application programs or system.
- System Library – System libraries can be considered as special programs or functions with which the system utilities or the application programs access the features of the kernel. Almost all of the functionality that these libraries can implement is also implemented by operating systems. The system libraries do not require the code of the kernel module for access rights.
- System Utility – System utilities are programs that are responsible for performing specialized and individual tasks.

Features

Here are some of the most important features of the Linux operating system:

- Portable – Portability means software can work on different types of hardware in the same way. Linux kernel and application programs can be installed on any kind of hardware platform.
- Open Source – Linux source code is freely available. Development is a community-based project. Multiple teams work in collaboration to enhance the capability of the Linux operating system, and it is continuously evolving.
- Multi-User – Linux is a multiuser system, meaning multiple users can access system resources like memory/ RAM/ application programs at the same time.
- Multiprogramming – Linux is a multiprogramming system, meaning multiple applications can run at the same time.
- Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.
- Shell – Linux provides a special interpreter program that can be used to execute operating system commands. It can be used for various types of operations, application programs etc.
- Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following layers are part of the Linux system architecture:

- Hardware– All the peripheral devices, like hard disks, CPU and RAM, are part of the hardware layer.
- Kernel – The kernel is the main or core component of the operating system. It communicates directly with the system hardware, providing the upper layer components with low level services.
- Shell – The shell is an interface between the user and the kernel. The shell accepts comments given by the user and executes kernel functions.
- Utilities – Utility programs are those that provide users with most of the operating system's functionalities.

Chapter 3: The Linux Kernel

In this chapter, we will learn about the Linux kernel. The Linux kernel is the computer operating system that is similar to UNIX. It is being used widely throughout the world. The Linux operating system is completely based on the Linux kernel. It is deployed on traditional computer systems through Linux distributions and also on embedded devices like routers. The Android mobile operating system, which works on mobile and tablet computers, is also built using the Linux kernel.

In 1991, Linus Torvalds, a computer science student from Finland, created the Linux kernel for his personal use. Initially he had no intention of using it on other platforms, but he gradually developed it so it could run on different computer architectures. The new operating system gained huge attention from developers, who were interested in adapting code from other projects involving free software to develop it further. Nearly 12,000 software developers and 1,200 companies contributed to the development of the Linux kernel.

The Linux kernel is free and open source software, which means that the source code of the kernel can be accessed and modified by anyone. LKML, or the Linux Kernel Mailing List, hosts the discussions on the development of the Linux kernel.

The kernel is the interface between the user applications and the CPU, memory and other devices.

The GNU Compiler Collection (GCC) supports a C language version (developed by incorporating several changes and extensions into standard C) which has been used to write the Linux kernel. Along with C, assembly language has also been used to write several small parts of the code. Since GCC supports several extensions of C, it was for some time the only compiler having the capability to build the Linux kernel.

Kernel sources

The kernel sources can be found in most Linux distributions. The Linux kernel that you install onto your system was developed using these sources. They can be found at ftp://ftp.cs.helsinki.fi and are shadowed on other websites. If you don't have Internet access, you can get hold of a CD ROM from vendors who offer snapshots of prominent websites for a reasonable price. Some vendors also offer subscription services with periodic updates. Your local Linux user group is another excellent alternative.

Several directories can be found at the highest level of the source tree /usr/src/Linux:

Arch: The kernel code specific to the architecture is present in this subdirectory. It is further divided into subdirectories, one for each architecture. Alpha and i386 are examples of the subdirectories.

Include: The include files required for building the kernel code are present in this subdirectory. It is also divided into one subdirectory for each architecture. Editing the kernel makefile and then rerunning the configuration program of the kernel can change the architecture.

Init: The kernel's initialization code is present in this directory. To understand the working of a kernel, this could be a good place to start.

MM: Files related to memory management are present in this directory. The code related to the memory management for each architecture is present in arch/*/mm/.

Drivers: This contains the device drivers of the system, which are classified into different classes.

IPC: The code for inter-process communication is present in this directory.

Modules: The modules are created and held by this directory.

FS: The code of the file system is present in this directory. This directory has several subdivisions, one for each file system.

Kernel: This consists of the kernel code. The kernel code specific to the architecture can be found in arch/*/kernel.

Net: This is the networking code of the kernel.

Lib: The library code of the kernel is present in this directory. The library code specific to an architecture is present in arch/*/lib/.

Scripts: The scripts that are used during the configuration of the kernel are present in this directory.

Chapter 4: Linux Processes

task_struct

For Linux to deal efficiently with the processes taking place in the system, a data structure called task_struct is used to represent each of those processes. (In Linux, the terms process and task are interchangeable.) The data structure task_struct has an array of pointers known as the task vector.

The task vector's size limits the number of processes happening in the system. The maximum number of entries is 512 by default. Creation of a new process results in the allocation of a new task_struct from the memory. Adding it into the task vector completes the creation. The current pointer points to the current running processes so that they are easier to find.

In addition to the normal process, Linux also supports real-time processes. These real-time processes are so named because they are expected to respond very swiftly to external requests. The scheduler treats them a bit differently than the normal processes. The complex data structure task_struct is made simpler by dividing it into the following functional areas:

State

The state of a process gets changed after its execution, depending on the circumstances. Each Linux process has one of the following states:

<u>Running</u>

In this state, the process is ready to run or is currently running in the system. Ready to run means the process is ready to be taken up by one of the system processors for execution.

<u>Waiting</u>

The waiting state means the process is waiting for a resource or for the occurrence of an event. Waiting processes are differentiated into two types: interruptible and uninterruptible. Signals can interrupt certain waiting processes, which are termed interruptible; waiting processes which the signals cannot interrupt are termed uninterruptible.

<u>Stopped</u>

A stopped process has, of course, been stopped, usually by receiving a signal. A process needs to be in the stopped state to be debugged.

<u>Zombie</u>

This is a dead process consisting of the data structure task_struct in the task vector. Hence, it is known as a zombie.

Scheduling information
This is the information which the scheduler uses to decide the order in which the processes should run.

Identifiers
A numerical process identifier identifies every process in a system.

Inter-process communication (IPC)

The important IPC mechanisms supported by Linux are signals, pipes, message queues, shared memory, semaphores etc.; you will learn them in detail in following chapters.

Links

Processes are not independent of one another in a system running on Linux. All processes have parent processes. The only process that does not have a parent process is the initial process. No new processes are created; instead, they are cloned from processes that have occurred previously. The task structure that represents a process points to the parent process and also to the siblings (processes that share the same parent). The pstree command can be used to view the relationships that run between the processes:

```
init(1)-+-crond(98)
    |-emacs(387)
    |-gpm(146)
    |-inetd(110)
    |-kerneld(18)
    |-kflushd(2)
    |-klogd(87)
    |-kswapd(3)
    |-login(160)---bash(192)---emacs(225)
    |-lpd(121)
    |-mingetty(161)
    |-mingetty(162)
    |-mingetty(163)
    |-mingetty(164)
    |-login(403)---bash(404)---pstree(594)
    |-sendmail(134)
    |-syslogd(78)
    `-update(166)
```

Times & timers

Another important duty of the kernel is keeping track of the time taken for creating a process. It also keeps track of the CPU time consumed by the process during its lifespan. As the clock ticks, the kernel keeps on updating the time spent by the current process in the system.

File systems

Linux is very flexible when it comes to supporting a wide variety of file systems. This helps it coexist with other popular operating systems. Linux supports a total of 15 file systems. They are:

- ext
- ext2
- vfat
- affs
- minix
- sysv
- hpfs
- ncp
- umsdos
- iso9660
- ufs
- xia
- msdos
- smb
- proc

Many more file systems will be added in future. The processes are free to open or close any file they need.

Virtual memory

Most processes use virtual memory. It is the duty of the Linux kernel to track the mapping of virtual memory to the physical memory. Some processes, such as daemons and kernel threads, do not use virtual memory.

Processor specific context

One can think of a process as an aggregate of the current state of the system. When a process runs, it uses the registers of the processors, stacks etc., and that is the context of the process. After the suspension of the process, the context of the process must be saved in the task_struct data structure so it can be restored when the process is restarted.

How Linux organizes data

An effective operating system should organize its data well, and Linux does so. In this chapter, we will learn how. If you have worked with other operating systems, like Microsoft Windows, you'll probably find it easy to understand, because most operating systems organize their data in similar ways.

Devices

Linux receives, stores and sends its data to and from devices. Every device corresponds to a hardware unit like a serial port keyboard, but it is not necessary for a device to have a hardware counterpart. The kernel can create many pseudo-devices that act like devices but do not physically exist. A given hardware unit can have more than one device connected to it; for example, the Linux operating system will show the partitions of a disk drive as separate devices.

Some of the typical Linux devices are listed in the table given below:

Device	**Description**
atibm	**Bus mouse**
audio	**Sound card**
cdrom	**CD-ROM drive**
console	**Current virtual console**
fd n	**Floppy drive (n designates the drive; for example, fd0 is the first floppy drive)**
ftape	**Streaming tape drive not supporting rewind**
hd xn	**Non-SCSI hard drive (x designates the drive and n designates the partition; for example, hda1 is the first partition of the first non-SCSI hard drive)**

inportbm	**Bus mouse**
lp n	**Parallel port (n designates the device number; for example, lp0 is the first parallel port)**
modem	**Modem**
mouse	**Mouse**
nftape	**Streaming tape drive supporting rewind**
nrft n	**Streaming tape drive supporting rewind (n designates the device number; for example, nrft0 is the first streaming tape drive)**
nst n	**Streaming SCSI tape drive not supporting rewind (n designates the device number; for example, nst0 is the first streaming SCSI tape drive)**
null	**Pseudo-device that accepts unlimited output**
printer	**Printer**
psaux	**Auxiliary pointing device, such as a trackball, or the knob on IBM's ThinkPad**
rft n	**Streaming tape drive not supporting rewind (n designates th device number; for example, rft0 is the first streaming tape drive)**
scd n	**SCSI device (n designates the device number; for example, scd0 is the first SCSI device)**
sd xn	**SCSI hard drive (x designates the drive and n designates the partition; for example, sda1 is the first partition of the firs SCSI hard drive)**

sr n	**SCSI CD-ROM (n designates the drive; for example, sr0 is the first SCSI CD-ROM)**
st n	**Streaming SCSI tape drive supporting rewind (n designates the device number; for example, st0 is the first streaming SCSI tape drive)**
tty n	**Virtual console (n designates the particular virtual console; for example, tty0 is the first virtual console)**
ttyS n	**Modem (n designates the port; for example, ttyS0 is an incoming modem connection on the first serial port)**
zero	**Pseudo-device that supplies an inexhaustible stream of zero-bytes**

Chapter 5: Linux File Systems

Whether you are using Linux or Windows, you will have to perform a partition before you can store any data. Whenever you format a partition in Linux, the operating system will write special data that is known as the file system on that partition.

Every operating system supports its own set of file systems. Compared to Windows, Linux supports a wide range of file system types. Linux-supported file systems are listed in the table given below:

Filesystem	**Description**
coherent	**A filesystem compatible with that used by Coherent Unix**
ext	**The predecessor of the ext2 filesystem; supported fo compatibility**
ext2	**The standard Linux filesystem**
hpfs	**A filesystem compatible with that used by IBM's OS/2**
iso9660	**The standard filesystem used on CD-ROMs**
minix	**An old Linux filesystem, still occasionally used on flopp diskettes**
msdos	**A filesystem compatible with Microsoft's FAT filesystem used by MS-DOS and Windows**
nfs	**A filesystem compatible with Sun's Network File System**
ntfs	**A filesystem compatible with that used by Microsof Windows NT's NTFS filesystem**
sysv	**A filesystem compatible with that used by AT&T's Systen V Unix**
vfat	**A filesystem compatible with Microsoft's FAT3**

	filesystem, used by Windows 9x
xenix	**A filesystem compatible with that used by Xenix**

Chapter 6: Inter-Process Communication

Inter-process communication is a process where various communication protocols are used to share data between different processes. Servers and clients are two types of applications that use inter-process communication. In this application, the clients are mapped to single or multiple servers from whom they request information in the form of data. The server or servers send in the data. Different platforms provide different ways for the clients to access data. Below are the mechanisms that Linux provides for inter-process communication to take place:

1. The kernel or processes send in signals to the process that they wish to communicate with. Signals are the cheapest forms of inter-process communication that the Linux platform provides, and can be in the form of events or states. Events and states both infer or notify that some change has occurred in the process of one process or kernel.
2. "|" is called a pipe. It sets up by the shell that the platform uses to transfer input from one process to another. The two ends of the pipe are linked as a one to one communication mechanism with the help of file descriptors on both ends. These file descriptors are associated with single functionality that is read-only or write-only. The end from which the details are to be sent uses the reading file descriptor that reads the data from the process. Meanwhile, the end where the data has to be taken from will use the writing descriptor so that it can write the data received. Since the data flow is single-sided, the mechanism is known as a half- duplex mechanism. Processes that have a parent-child relationship where the parents create the pipes to their children through which they can communicate with them and also create pipes between their children so that they can communicate between themselves can use these pipes. This happens because the file descriptors are copied into their address spaces.
3. First-in first-out (FIFO) pipes are also known as named pipes, and are used in the same way as unnamed pipes. The only difference is that these are used when the processes are not related to each other. The

processes using these pipes communicate using FIFO files instead of file descriptors. FIFO files have the same mechanism as file descriptors: one FIFO file has reading functionality and the other has writing functionality. Therefore this mechanism is also half-duplex.

4. For single or multiple processes to read or write from single or multiple processes, we use message queues. This mechanism is similar to that of a post box. The process that wants to share the data comes and places the packet containing the data onto the message queue and leaves. The process that wants to take the data comes and takes the packet. This saves a lot of time, as the process sharing the data needn't wait for the other process. There are two different specifications for message queues. First are SysvV message queues, where the messages that are being sent contain a number so that the process receiving the message can take the data matching a particular number or accept everything except the packet that matches a particular number. The second, POSIX, is a mechanism that works on priority. It sends the packets in their priority order. The priority number is attached with the packets when the process is sending them. So when the other process receives the details the data in the highest priority packet is taken first.
5. There are counters that queue the order of processes that will receive the shared resources one after another. These counters are known as semaphores. They lock the shared resource so that only one process can access the shared resource. Once this process is done using the resource it is released. Then the process that is next in priority will get the resource, or the process that requested the resource irrespective of the priority will get the resource. This is similar to the SysV and POSIX mechanisms of the message queues, so these are known as SysV semaphores and POSIX semaphores.
6. The mechanism in which multiple processes share the memory space available is known as the shared memory mechanism. This helps two or more processes that share the memory to communicate more efficiently so that there is less interference of the kernel. There are also two types of shared memory, SysV shared memory and POSIX shared memory, and they work in the same way as above. That is, the priority is considered in one but not in the other.

Chapter 7: Linux Shell

As you already know, a computer can only understand zeros and ones, i.e., binary language. In the first generation of computing, instructions were given directly in binary language. But binary language is extremely hard to understand, and writing code in it is time consuming and difficult. To solve this problem, a special program called the shell was introduced to operating systems. A shell accepts commands given in English and checks if they are valid.

If a given command is valid, the shell passes it to the kernel. The shell can be considered as a program that simplifies the interaction between the user and the kernel. The shell is a command line interpreter. It can take input from a file or from a standard input device like a keyboard. Though the shell is not part of the kernel, it uses the kernel for creating files, executing programs etc.

Most basically, the shell is an interface between the user and the system kernel, but the shell can execute some basic commands without passing them to the kernel.

If you wish to know the available shells that are present in your system, use the following command:

```
$ cat /etc/shells
```

Though there are many shells available within an operating system, all of them are designed to do the same job. Different shells have different syntaxes and different built-in functions. In MS-DOS, command.com is a shell that the Windows operating system uses. It is less powerful than the shells present in Linux.

All shells take commands from the user through a file or an input device such as a keyboard and convey messages to the Linux operating system saying what the user requires.

If the user gives the command using an input device like a keyboard, it is known as the command line interface. The command given below shows the current shells on which you are working:

```
$ echo $SHELL
```

Shell scripts

All shells are interactive, which means that they accept commands from the user, and can execute them. You can either give your commands one after another or you can store them all in a sequence in a text file and give this to the shell to execute. Giving the commands all at once is called shell scripting.

Shell scripts are commonly defined as follows:

"A shell script is a series of commands written in a plain text file. A shell script is just like a batch file in MS-DOS, but it has more power than the MS-DOS batch file."

Reasons to write shell scripts

- You can give input to a shell script directly or through a file, and it will give the output on the screen.
- You can create your own commands using shell scripts.
- Shell scripting will save you a lot of time by taking all the commands at once.
- Shell scripting can be used for automating day-to-day tasks.
- You can also automate system administration using shell scripting.

How to write shell scripts

When writing a shell script, follow these steps:

- Use an editor such as mcedit or vi.
- After writing your set of commands in the shell script, you should set the execute permissions the script prepared with *chmod permission your-script-name*. For example, *$ chmod +x your-script-name* or *$ chmod 755 your-script-name*. The system admin or owner should have read/write/execute permission (7). When setting the permission for a group you should only use read and execute (5).
- Now you should execute your written script following the syntax given below:

syntax: bash your-script-name

sh your-script-name

./your-script-name

Examples:

```
$ bash bar

$ sh bar

$ ./bar
```

- Note that ./ indicates your current working directory and the dot (.) indicates the execution of the given command file present in the current shell without initializing a new copy of the shell. The syntax for using the dot is . *command-name*; for example, *$. foo*.
- Now you can write your own new shell script called First Shell using the vi command list (or any other editor):

```
$ vi first

#

# My first shell script

#

clear

echo "First Shell"
```

- After saving the script, you can run it with *$ ./first*, but first you have to set the permission with *$ chmod 755 first*. Now when you input *$ ./first* the screen will be cleared and then the message given in your shell script will be printed:

```
Output: First Shell
```

● **Script Command(s)**	**Meaning**
$ vi first	**Start vi editor**
# **# My first shell script** **#**	**# followed by any text is considered a comment A comment gives more information about th script; it is a logical explanation about a shel script.** ***Syntax:*** **# comment-text**

clear	clear the screen
echo "First Shell"	To print message or value of variables on screen we use the echo command. The general form o echo command is as follows: *syntax:* echo "Message"

Basic command line editing

For editing and recalling commands, you can use the key combinations given below:

CTRL + L: Clears the screen.

CTRL + W: Delete the word starting at cursor.

CTRL + U: Clear the line i.e. Delete all words from command line.

Up and Down arrow keys: Recall commands.

Tab: Auto-complete files, directory, command names etc.

CTRL + R: Search through previously used commands.

CTRL + C: Cancel currently running commands.

CTRL + T: Swap the last two characters before the cursor.

ESC + T: Swap the last two words before the cursor.

CTRL + H: Delete the letter starting at cursor.

Executing a command

To execute a command, just type it on the terminal and press the enter key. For example, the date command will give the current date and time as output, so when you type *date* and press enter you get something like: Wed Jan 27 05:35:56 IST 2015.

Bash shell commands

Here is an alphabetical list of all the bash commands from http://ss64.com/bash/:

alias	Create an alias •
apropos	Search Help manual pages (man -k)
apt-get	Search for and install software packages (Debian/Ubuntu)
aptitude	Search for and install software packages (Debian/Ubuntu)
aspell	Spell Checker
awk	Find and Replace text, database sort/validate/index
basename	Strip directory and suffix from filenames
bash	GNU Bourne-Again SHell
bc	Arbitrary precision calculator language
bg	Send to background
break	Exit from a loop
builtin	Run a shell builtin
bzip2	Compress or decompress named file(s)
cal	Display a calendar
case	Conditionally perform a command
cat	Concatenate and print (display) the content of files
cd	Change Directory
cfdisk	Partition table manipulator for Linux
chgrp	Change group ownership
chmod	Change access permissions

chown	Change file owner and group
chroot	Run a command with a different root directory
chkconfig	System services (runlevel)
cksum	Print CRC checksum and byte counts
clear	Clear terminal screen
cmp	Compare two files
comm	Compare two sorted files line by line
cp	Copy one or more files to another location
cron	Daemon to execute scheduled commands
crontab	Schedule a command to run at a later time
csplit	Split a file into context-determined pieces
cut	Divide a file into several parts
date	Display or change the date & time
dc	Desk Calculator
dd	Convert and copy a file, write disk headers, boot records
	ddrescue Data recovery tool
declare	Declare variables and give them attributes •
df	Display free disk space
diff	Display the differences between two files
diff3	Show differences among three files
dig	DNS lookup
dir	Briefly list directory contents
dircolors	Color setup for `ls'
dirname	Convert a full pathname to just a path
dirs	Display list of remembered directories

dmesg	Print kernel & driver messages
du	Estimate file space usage
echo	Display message on screen •
egrep	Search file(s) for lines that match an extended expression
eject	Eject removable media
enable	Enable and disable builtin shell commands •
env	Environment variables
ethtool	Ethernet card settings
eval	Evaluate several commands/arguments
exec	Execute a command
exit	Exit the shell
expect	Automate arbitrary applications accessed over a terminal
expand	Convert tabs to spaces
export	Set an environment variable
expr	Evaluate expressions
false	Do nothing, unsuccessfully
fdformat	Low-level format a floppy disk
fdisk	Partition table manipulator for Linux
fg	Send job to foreground
fgrep	Search file(s) for lines that match a fixed string
file	Determine file type
find	Search for files that meet a desired criteria
fmt	Reformat paragraph text
fold	Wrap text to fit a specified width.

for	Expand words, and execute commands
format	Format disks or tapes
free	Display memory usage
fsck	File system consistency check and repair
ftp	File Transfer Protocol
function	Define Function Macros
fuser	Identify/kill the process that is accessing a file
gawk	Find and Replace text within file(s)
getopts	Parse positional parameters
grep	Search file(s) for lines that match a given pattern
groupadd	Add a user security group
groupdel	Delete a group
groupmod	Modify a group
groups	Print group names a user is in
gzip	Compress or decompress named file(s)
hash	Remember the full pathname of a name argument
head	Output the first part of file(s)
help	Display help for a built-in command •
history	Command History
hostname	Print or set system name
htop	Interactive process viewer
iconv	Convert the character set of a file
id	Print user and group id's
if	Conditionally perform a command
ifconfig	Configure a network interface

ifdown	Stop a network interface
ifup	Start a network interface up
import	Capture an X server screen and save the image to file
install	Copy files and set attributes
ip	Routing, devices and tunnels
jobs	List active jobs
join	Join lines on a common field
kill	Kill a process by specifying its PID
killall	Kill processes by name
less	Display output one screen at a time
let	Perform arithmetic on shell variables •
link	Create a link to a file
ln	Create a symbolic link to a file
local	Create variables •
locate	Find files
logname	Print current login name
logout	Exit a login shell •
look	Display lines beginning with a given string
lpc	Line printer control program
lpr	Off line print
lprint	Print a file
lprintd	Abort a print job
lprintq	List the print queue
lprm	Remove jobs from the print queue
ls	List information about file(s)

lsof	List open files
make	Recompile a group of programs
man	Help manual
mkdir	Create new folder(s)
mkfifo	Make FIFOs (named pipes)
mkisofs	Create an hybrid ISO9660/JOLIET/HFS filesystem
mknod	Make block or character special files
more	Display output one screen at a time
most	Browse or page through a text file
mount	Mount a file system
mtools	Manipulate MS-DOS files
mtr	Network diagnostics (traceroute/ping)
mv	Move or rename files or directories
mmv	Mass Move and rename (files)
netstat	Networking information
nice	Set the priority of a command or job
nl	Number lines and write files
nohup	Run a command immune to hangups
notify-send	Send desktop notifications
nslookup	Query Internet name servers interactively
open	Open a file in its default application
op	Operator access
passwd	Modify a user password
paste	Merge lines of files
pathchk	Check file name portability

ping	Test a network connection
pkill	Kill processes by a full or partial name.
popd	Restore the previous value of the current directory
pr	Prepare files for printing
printcap	Printer capability database
printenv	Print environment variables
printf	Format and print data •
ps	Process status
pushd	Save and then change the current directory
pv	Monitor the progress of data through a pipe
pwd	Print Working Directory
quota	Display disk usage and limits
quotacheck	Scan a filesystem for disk usage
quotactl	Set disk quotas
ram	ram disk device
rar	Archive files with compression
rcp	Copy files between two machines
read	Read a line from standard input •
readarray	Read from stdin into an array variable •
readonly	Mark variables/functions as readonly
reboot	Reboot the system
rename	Rename files
renice	Alter priority of running processes
remsync	Synchronize remote files via email
return	Exit a shell function

rev	Reverse lines of a file
rm	Remove files
rmdir	Remove folder(s)
rsync	Remote file copy (Synchronize file trees)
screen	Multiplex terminal, run remote shells via ssh
scp	Secure copy (remote file copy)
sdiff	Merge two files interactively
sed	Stream Editor
select	Accept keyboard input
seq	Print numeric sequences
set	Manipulate shell variables and functions
sftp	Secure File Transfer Program
shift	Shift positional parameters
shopt	Shell Options
shutdown	Shutdown or restart linux
sleep	Delay for a specified time
slocate	Find files
sort	Sort text files
source	Run commands from a file '.'
split	Split a file into fixed-size pieces
ssh	Secure Shell client (remote login program)
stat	Display file or filesystem status
strace	Trace system calls and signals
su	Substitute user identity
sudo	Execute a command as another user

sum	Print a checksum for a file
suspend	Suspend execution of this shell •
sync	Synchronize data on disk with memory
tail	Output the last part of file
tar	Store, list or extract files in an archive
tee	Redirect output to multiple files
test	Evaluate a conditional expression
time	Measure Program running time
timeout	Run a command with a time limit
times	User and system times
touch	Change file timestamps
top	List processes running on the system
tput	Set terminal-dependent capabilities, color, position
traceroute	Traceroute to Host
trap	Run a command when a signal is set(bourne)
tr	Translate, squeeze, and/or delete characters
true	Do nothing, successfully
tsort	Topological sort
tty	Print filename of terminal on stdin
type	Describe a command •
ulimit	Limit user resources •
umask	Users file creation mask
umount	Unmounts a device
unalias	Remove an alias •
uname	Print system information

unexpand	Convert spaces to tabs
uniq	Uniquely files
units	Convert units from one scale to another
unrar	Extract files from a rar archive
unset	Remove variable or function names
unshar	Unpack shell archive scripts
until	Execute commands (until error)
uptime	Show uptime
useradd	Create new user account
userdel	Delete a user account
usermod	Modify user account
users	List users currently logged in
uuencode	Encode a binary file
uudecode	Decode a file created by uuencode
v	Verbosely list directory contents (`ls -l -b')
vdir	Verbosely list directory contents (`ls -l -b')
vi	Text Editor
vmstat	Report virtual memory statistics
wait	Wait for a process to complete •
watch	Execute/display a program periodically
wc	Print byte, word, and line counts
whereis	Search the user's $path, man pages and source files for a program
which	Search the user's $path for a program file
while	Execute commands
who	Print all usernames currently logged in

whoami	Print the current user id and name (`id -un')
wget	Retrieve web pages or files via HTTP, HTTPS or FTP
write	Send a message to another user
xargs	Execute utility, passing constructed argument list(s)
xdg-open	Open a file or URL in the user's preferred application.
yes	Print a string until interrupted
zip	Package and compress (archive) files.
.	Run a command script in the current shell
!!	Run the last command again
###	Comment / Remark

Commands with a dot (•) following the description are default shell commands.

Chapter 8: Command Line Tutorial

Now you know what Linux is all about, it’s time to start actually using it. We’ll start with a basic command line tutorial, but first you need to know how to open the terminal. This isn’t hard, but all systems are different, so use this as a guideline only:

- Mac users should open Applications, Utilities and then click on Terminal. A shortcut would be to press Command+Space, which will open Spotlight, after which you can type in the word Terminal. Click on the result when it shows up.
- Linux users should look in Applications>System or Applications>Utilities. A shortcut is to right click on your desktop and look for the option to “Open in Terminal”. If it is there, click on it.
- Windows users can simply type the word Command in the search bar on their start menu. Click on the option for Command Prompt and it will take you straight to it.

As you know, the command line is where you type your commands and receive output, or feedback, as text. The command line starts with a prompt and the text you type is displayed after this prompt. Most of your work will be command work, for example:

```
1. ls -l /home/Simon
2. total 3
3. drwxr-xr-x  2 Simon users 4096 Mar 23 13:34 bin
4. drwxr-xr-x 18 Simon users 4096 Feb 17 09:12 Documents
5. drwxr-xr-x  2 Simon users 4096 May 05 17:25 public_html
6.
```

Let’s break that code down and see what it all means:

Line 1 always starts with the prompt with your username@bash. We also put a command in there – ls – and the command is always the first thing that you type. Following that are what we all command line arguments. Be aware that there must always be a space between the command and the first argument. The very first command line argument is (-l) and this is called an option.

These are used to change the way the command behaves. They are always listed before any other argument and start with a dash (-).

Lines 2 to 5 are outputs that come from running the command that you typed in. Nearly all commands will produce an output, and it will always be listed immediately under the command issued. A few commands just execute and don’t produce any information unless there is an error.

Line 6 is the prompt again. Once the command has been run, the terminal is ready for use again and will display the prompt. If there is no prompt showing, the command is most likely still running.

Shortcuts

While the terminal may seem a little overwhelming at first, there is nothing to be worried about. Linux is packed full of handy shortcuts that will make things much easier and can also stop you from making mistakes like typographical errors. I will go through some of them as I go through this chapter so make a note of them as you come across them.

Your first shortcut is this – whenever you enter a command it is stored in a history, and you can work your way through this history by using the up and down arrow keys. So you don't need to type out commands that you have already typed; if you need to enter the same one again, simply use the arrow keys to find it. If you want to edit a command, use the left and right arrow keys to put the cursor where you need it to make the change.

Basic navigation

The very first command we'll look at is pwd, which stands for Print Working Directory. (Many of the Linux commands are shortened to abbreviations and this makes it much easier to remember them.) This command does exactly what it says it does – prints the working directory to the screen. Give it a go:

```
pwd
```

```
/home/Simon
```

Many of the commands that you type in on the terminal will only work if you're in the right location. But as you will be moving around quite a bit, it will be easy to get lost. Simply typing pwd at the command prompt will tell you your location, so make use of it whenever you're in doubt.

The next thing you might want to know is what's in your current location. The command to find that out is ls, which is short for list.

```
ls
```

```
bin Documents public_html
```

Where pwd runs without arguments, ls has a bit more power. In this example, we haven't given it any arguments, so that we just get a list of what is in the current location. But you can do more with it, for example:

```
ls [options] [location]
```

The [] square brackets indicate that the text inside them is optional so the command can run with them or without them. The example below show the ls command run in a number of different ways just to give you an idea of what it can do:

```
ls
```

```
bin Documents public_html
```

This is ls in its absolute most basic format – the result is a list of the contents of the directory we are in.

```
ls -l
```

```
total 3
```

```
drwxr-xr-x  2 Simon users 4096 Mar 23 13:34 bin

drwxr-xr-x 18 Simon users 4096 Feb 17 09:12 Documents

drwxr-xr-x  2 Simon users 4096 May 05 17:25 public_html
```

Using ls with the command line option (-l) indicates that we want to do a long listing, which consists of:

- The first character indicates the file type – normal (-) or directory (d)
- The following nine characters are file or directory permissions
- The next field is the number of the blocks
- The next field indicates the file or directory owner
- The next field indicates the group that the directory or file belongs to
- Next is the size of the file
- Next is the file modification time
- Last is the name of the directory or file

```
ls /etc

a2ps.cfg aliases alsa.d cups fonts my.conf systemd

…
```

This time, ls is run with a command line argument, (/etc), which is telling ls that it shouldn't list the current directory; instead it should list the contents of the directory. (The ellipsis (…) indicates that some of the output has been cut out to save space; if you actually run this command, you will get a much longer listing of directories and files.)

```
ls -l /etc

total 3

-rwxr-xr-x  2 root root 123 Mar 23 13:34 a2ps.cfg

-rwxr-xr-x 18 root root 78 Feb 17 09:12 aliases

drwxr-xr-x  2 Simon users 4096 May 05 17:25 alsa.d

…
```

Now ls is run with a command line option and a command line argument, resulting in a long listing of the directory /etc.

Paths

In these last commands, we started to introduce paths. It is important that you learn what these are and how to use them as you cannot be proficient in the Linux language until you do so. When we talk about a directory or a file on the command line, we are talking about a path. Just like in the physical world, a path is a way of getting somewhere, in this case to a specific directory or file on the system.

Absolute & relative paths

We use two different paths – absolute and relative – to refer to a directory or file. Either can be used, as they will both take you to the same place on the system. To start with, you need to understand that Linux has what's known as a hierarchical structure. Right at the top is the root directory, indicated by a single slash (/). The root directory contains sub-directories, and these have their own sub-directories, and so on. Any of these sub-directories can contain files.

An absolute path specifies a particular file or directory relative to the root directory. They are identifiable by the slash that they all begin with. A relative path specifies a file or directory relative to your current location in the system and does not start with a slash.

Have a look at this example:

```
pwd
```

```
/home/Simon
```

Current location is confirmed by running pwd.

```
ls Documents
```

```
file1.txt file2.txt file3.txt
```

…

Now ls is run with a relative path. The current location is the Documents directory, but this command will give different results depending on where you are in the system. If you were in the home directory of another system user, it would provide you with a list of the contents of that user's Documents directory.

```
ls /home/Simon/Documents
file1.txt file2.txt file3.txt
…
```

Finally ls is run with an absolute path. This command is not reliant on where you are in the system; the result will be exactly the same wherever you are.

Building blocks

As you learn Linux, it will soon become apparent that there are loads of different ways to get things done. The following are called building blocks, and they will help you to build up your paths:

- ~ (tilde) – A shortcut for the home directory. For example, if your home directory is called /home/Simon, you can refer to the Documents directory with the path /home/Simon/Documents, or you could just type ~/Documents.
- . (dot) – A reference to the current directory. Take the earlier example where we referred to Documents with a relative path. We could also have written it as ./documents.
- .. (dotdot) – A reference to the parent directory. This can be used any number of times in a single path if you want to keep on traversing upwards through the hierarchy. For example, if you were in /home/Simon, you could simply type in ls../../ and you would get a listing of the root directory as your output.

Now you can start to see that you can refer to locations in a number of ways. As to which way you should use, the answer is you can use any of them. If you make a reference to a directory or file on the command line, you are referring to a path, and that can be made up with any of the elements we talked about. The best way is whichever way you find the easiest.

Have a look at these examples:

```
pwd
/home/Simon
ls ~/Documents
file1.txt file2.txt file3.txt
```

```
…
ls ./Documents
file1.txt file2.txt file3.txt
…
ls /home/Simon/Documents
file1.txt file2.txt file3.txt
…
ls ../../
bin boot dev etc home lib var
…
ls /
bin boot dev etc home lib var
…
```

Have a go at inputting these into the command line on your system – I promise that they will begin to make a bit more sense to you. Make sure you take the time to learn all about paths and really understand them; you will use them a lot.

Chapter 9 – Moving Around Linux

You can use a command called *cd* to move around in Linux. This stands for change directory, and it works like this:

```
cd [location]
```

If you run cd without using an argument, it will take you to your home directory automatically. But usually it is run with a single argument on the command line, i.e., the location of the directory that you want to change to. This is specified as a path and can be a relative or an absolute path using any of the building blocks that we talked about in the last chapter.

Have a look at these examples:

```
pwd

/home/Simon

cd Documents

ls

file1.txt file2.txt file3.txt

…

cd /

pwd

/

ls

bin boot dev etc home lib var

…

cd ~/Documents

pwd

/home/Simon/Documents
```

```
cd ../../
pwd
/home
cd
pwd
/home/Simon
```

Tab completion

Repeatedly typing out these paths can become tedious and you may end up making errors. However, there is a neat mechanism in the command line that can help with this. It’s called tab completion. When you begin to type out a path, no matter where you are on the command line and what command you are using, you can press the Tab key to invoke auto-complete. If nothing changes, it means that there are a number of possibilities for completing the path. Pressing the Tab button again will show those possibilities. Continue typing your path and then hit Tab again to narrow the possibilities down to the right one. Have a go on your own command line and see how you get on.

Files

In Linux, everything is a file. Directories, text files, even your keyboard (but only one that your system can read from) – all of them are files. The monitor that your system writes to is a file as well. This isn't going to have any effect on what you do, but you should keep it in mind to help you understand how Linux works.

Extensionless system

This might take some getting used to, but as you work your way through the programming, it will start to make sense. File extensions are usually a set of characters, between two and four, after a dot at the end of the file name. The extension tells you what the file type is. Some common extensions are:

- file.exe – a file or program that is executable
- file.txt – a file that is plain text
- file.png – an image file
- file.gif – an image file
- file.jpg – an image file

In systems like Windows, the file extension is very important because the system uses it to see what file type it is dealing with. Linux is different in that it ignores the file extension, instead looking into the file itself to see what it is. As an example, you could take a .jpg file that you have on your system, perhaps a photo of your house, and change the extension to .txt. Linux would ignore that, look inside the file, and then treat it as the image file it actually is. But it isn't always easy to determine what file type a file should be considered, and that's where a little command named *file* comes in.

```
file [path]
```

Now, you might be thinking at this point, why has a command line been specified as a path instead of a file? Remember, when a file or a directory is specified on the command line, it is actually a path. Also, directories are nothing more than a specific type of file; a path is used to get us to a particular place in the system and that place is a file.

Linux is case sensitive

This is very important to remember, and is one of the biggest causes of the problems experienced by newcomers to Linux. Many operating systems are not sensitive to case, but Linux is, especially when you are referring to files. In Linux, it is perfectly possible to have two files or directories with names that are identical except for the cases, for example:

```
ls Documents

FILE1.txt File1.txt file1.TXT

…

file Documents/file1.txt

Documents/file1.txt: ERROR: cannot open 'file1.txt' (No such file or directory)
```

Linux sees all of these as different files. You should also be aware of case when you are dealing with command line options. For example, with the ls command, there are two options – S and s – which do two different things. One of the most common mistakes is to enter options uppercase where they should be lowercase and then wonder why the output isn't what you expected it to be.

Spaces in names

There is nothing wrong with having spaces in directory and file names, but you should be careful with them. A space on the command line is used to separate things – they are how we determine the program name and are able to identify the arguments on the command line. Let’s say that you wanted to move to a directory named Holiday Photos. This example would not work:

```
ls Documents

FILE1.txt File1.txt file1.TXT Holiday Photos

…

cd Holiday Photos

bash: cd: Holiday: No such file or directory
```

In this example, Holiday Photos is treated as two separate command line arguments. The command cd moves to the directory that is specified by the first argument only. To get round this and make it work, we need to let the terminal know that Holiday Photos is to be seen as a single command line argument. There are two equally valid ways we can do this.

Quotes

The first way is to enclose Holiday Photos in quote marks – single or double, it doesn’t matter which, so long as you use a matching pair. Anything that is written inside the quotes is treated as one single item:

```
cd 'Holiday Photos'

pwd

/home/Simon/Documents/Holiday Photos
```

Escape characters

Another way is to use something called an escape character. This is a backslash (\) and what this does is nullify the special meaning that the next character has:

```
cd Holiday\ Photos

pwd
```

/home/Simon/Documents/Holiday Photos

In this example, the space that is between the words Holiday and Photo would usually have a special meaning – to separate them as individual command line arguments. However, putting the backslash in front nullified that meaning. If you use tab completion, before you come across a space inserted in the directory name, the terminal will escape the space automatically for you.

Hidden files & directories

Linux has a very nice mechanism for telling you that a directory or file has been hidden. If the name of the file or directory starts with a dot (.) it is classed as hidden. There are no special commands or actions needed to hide a file or directory. There are a number of reasons why you might want to hide a file or a directory. For example, you might have another user who hides their configuration fees so they don’t get in your way.

To hide a file or directory, simply add a dot to the start of the name. If you have already created the file, simply rename it with a dot in the front. In the same way, you can remove the dot from a file or directory to unhide it.

One thing to keep in mind – the ls command will NOT list any directories or files that are hidden unless you modify the command by adding the command line option –a. Only then will it list anything that has been hidden.

```
ls Documents

FILE1.txt File1.txt file1.TXT

…

ls -a Documents

. .. FILE1.txt File1.txt file1.TXT .hidden .file.txt

…
```

Manual pages

Linux makes available a set of pages that tells you all about every single command– what they do, how they work, how to run them and the command line arguments that each one will accept. Some of them will be a little difficult to understand for a newcomer to Linux, but they are consistent. Once you get the hang of a few, they will start to mean something to you and you will find them easier to work with. Manual pages are invoked with this command:

```
man <command to look up>
```

For example:

```
man ls
```

Name

ls - list directory contents

Synopsis

ls [option] ... [file] ...

Searching

You can search through the manual pages for a specific keyword. This is helpful if you need a reminder of what command you need – when you know what you want to do but aren't certain of the command that you need to use. You might need to try this a few times before you find what you're after. You also may find that the particular keyword you are searching for shows up in a number of pages, and then you'll have to sift through them to find the right one. The syntax is:

```
man -k <search term>
```

You can also search inside of a manual page. While you are on a page and you want to search for something on that page, press the forward slash (/) button and follow it up with the search term. Press enter and your results will appear. If there are numerous instances of the search term, you can press 'n' to move through them.

More on running commands

Much of being good at Linux is knowing the right command line options to use to modify the way the commands behave to suit your requirements. Most commands have both a long and a shorthand version. For example, we talked about hidden files above. We can use the command –a or we can use –all to list all directory entries and any hidden files. The first is the shorthand version while the second is the longhand. You can use either version; it makes no difference because they both do the same thing. There is one advantage to using longhand, and that is that you may find it easier to remember what your commands are doing. However, an advantage of the shorthand method is that you can chain a number of commands together:

```
pwd

/home/Simon

ls -a

ls --all

ls -alh
```

Command line options written in longhand start with a pair of dashes (--) while the shorthand version starts with one single dash (-). The single dash can be used to invoke a number of options by putting all the letters that indicate these options together following the dash.

Making a directory

As time goes by, you will build up quite a lot of data. Because of this, you should create a system of directories to organize and manage it using Linux's hierarchical system. Get into the habit of creating new directories for new stuff and storing existing files in their correct directory.

Creating a directory is easy; we simply use the command mkdir, shorthand for make directory, as follows:

```
mkdir [options] <Directory>
```

Use mkdir, supply a directory name, and it will be created for you:

```
pwd

/home/Simon

ls

bin Documents public_html

mkdir linuxtutorialwork

ls

bin Documents linuxtutorialwork public_html
```

In the above example, we start off by checking our current location, then run a listing so that we know exactly what that directory contains. We then run the mkdir command and make a directory named linuxtutorialwork. Remember, when we give the directory name in the command line, we are giving a path.

In the following examples we see a few more ways to create directories:

```
mkdir /home/Simon/foo

mkdir ./blah

mkdir ../dir1

mkdir ~/linuxtutorialwork/dir2
```

There are some very useful options for mkdir. The first of these is –p, which is telling the command mkdir that it should create parent directories as necessary. The second is –v, which commands mkdir to tell us what it’s doing.

Here’s an example with the –p option:

```
mkdir -p linuxtutorialwork/foo/bar

cd linuxtutorialwork/foo/bar

pwd

/home/Simon/linuxtutorialwork/foo/bar
```

Now let’s look at the same example using the –v option as well:

```
mkdir -pv linuxtutorialwork/foo/bar

mkdir: created directory 'linuxtutorialwork/foo'

mkdir: created directory 'linuxtutorialwork/foo/bar'

cd linuxtutorialwork/foo/bar

pwd

/home/Simon/linuxtutorialwork/foo/bar
```

Removing a directory

It’s easy to make a new directory, and it is also very easy to delete one. Do keep in mind that there is no undo option when you delete something on Linux using the command line. So be careful and get into the habit of double-checking everything before you go ahead and delete. The command we use to delete a directory is rmdir, short for remove directory:

```
rmdir [options] <Directory>
```

There are two things to bear in mind here – first, rmdir supports both the -p and the -v options; and second, the directory has to be empty before you can delete it.

```
rmdir linuxtutorialwork/foo/bar
```

```
ls linuxtutorialwork/foo
```

Creating a blank file

A feature of many commands that are involved with the manipulation of data is that, if we refer to a directory that does not yet exist, they will create it for us. We can use this to create blank files with a command named *touch*:

```
touch [options] <filename>
```

For example:

```
pwd

/home/Simon/linuxtutorialwork

ls

foo

touch example1

ls

example1 foo
```

The touch command is used to modify the modification and access times on files. It isn’t always needed, but it can be helpful to use if you are testing a system that relies on file modification or file access times. What is important, as I said earlier, is that if you touch on a file or directory that does not exist, it will be created for you.

Much of what we do in Linux cannot be done automatically, but when you know how a certain command behaves and some of the deeper aspects of the system, you can use the command in creative ways to get the outcome you desire.

Copying a file or directory

Sometimes you might want to make a copy of a directory or a file, for example if you're about to make changes to it and you want a backup so that, if things do go wrong, all is not lost! The command for this is cp, which is short for copy:

```
cp [options] <source> <destination>
```

There are a number of options for this command, one of which I will demonstrate now:

```
ls

example1 foo

cp example1 barney

ls

barney example1 foo
```

Did you see that both the destination and the source are paths? These can be referred to using both relative and absolute paths, as per these examples:

```
cp /home/Simon/linuxtutorialwork/example2 example3

cp example2 ../../backups

cp example2 ../../backups/example4

cp /home/Simon/linuxtutorialwork/example2 /otherdir/foo/example5
```

When you use the cp command, it can be either a path to a file or a path to a directory. If it goes to a file, a copy of the source will be created and it will be named by the filename that is specified in the destination path. If the path goes to a directory, the file will be copied into that directory and be named the same as the source.

By default, cp only copies a file. If we use the option –r, which stands for Recursive, we can copy directories. Recursive means that you want to see a directory and you want to see all of the files and the directories contained in it. For the subdirectories, simply go into them and repeat the command, and so on:

```
ls
barney example1 foo
cp foo foo2
cp: omitting directory 'foo'
cp -r foo foo2
ls
barney example1 foo foo2
```

In this example, the files and the directories that are contained in the directory named foo are copied to foo2.

Moving a file or directory

If you want to move a directory or a file, you can do so with the command mv, which stands for move. It works in pretty much the same way as cp, with one small advantage – we don’t need to use the –r option to move a directory:

```
mv [options] <source> <destination>
```

For example:

```
ls
barney example1 foo foo2
mkdir backups
mv foo2 backups/foo3
mv barney backups/
ls
backups example1 foo
```

Let’s break that down. In line 3, a new directory called backups is created. In line 4, the directory called foo2 is moved into the new backups directory and renamed foo3. Then the file called barney is moved to the directory called backups. It retains the same name because we didn’t give a new destination name.

Renaming files & directories

As with the touch command, we can use the base behavior of the mv command in a slightly different way, so that we get a different outcome. As you can see from line 4 in the example above, we might need to give a new name for a directory or file, and when it is moved, it is renamed. If you specify that the destination is the same directory name as the source, but provide a different name, you have used the mv command to change the name of a file or directory, as in the following example:

```
ls
backups example1 foo
mv foo foo3
ls
backups example1 foo3
cd ..
mkdir linuxtutorialwork/testdir
mv linuxtutorialwork/testdir /home/Simon/linuxtutorialwork/fred
ls
backups example1 foo3 fred
```

So first, in line 3, the file called foo is renamed to foo3, using relative paths. Then with cd.. we go to the parent directory so that, in the next line, we can show that commands can be run on directories or files even if we are not in their current directory. In line 8, the directory called testdir is renamed to fred, using a relative path for the source and an absolute path for the destination.

Removing a file

The same as with rmdir: when we remove a file, the action cannot be undone – be very careful what you are doing! We use the command rm, short for remove, to delete or remove a file:

```
rm [options] <file>
```

For example:

```
ls
backups example1 foo3 fred
rm example1
ls
backups foo3 fred
```

Removing non-empty directories

The command rm has a number of options that will change the way it works. One very useful one is -r, meaning recursive, the same as in the command cp. When you run rm with that -r option, you can remove directories that still contain files and directories:

```
ls

backups foo3 fred

rmdir backups

rmdir: failed to remove 'backups': Directory not empty

rm backups

rm: cannot remove 'backups': Is a directory

rm -r backups

ls

foo3 fred
```

One good option to use with -r is -i, which means interactive. If you use this, whenever you give the command to remove a directory or file, you will get a prompt before each one is removed – this gives you the option of cancelling if you’ve made a mistake.

Wildcards

Wildcards are building blocks that let you come up with a pattern that defines a set of directories or files on Linux. Remember that when we refer to directories and files on the command line, it is a path that we are referring to. When we do this, we can use wildcards within the path, which allows us to turn it into a set of directories or files.

The basic wildcards are:

- * represents zero or more characters.
- ? represents a single character.
- [] represents a range of characters.

For the first example, we will use * to request a list of all entries that begin with the letter “b”:

```
pwd

/home/Simon/linuxtutorialwork

ls

barry.txt blah.txt bob example.png firstfile foo1 foo2

foo3 frog.png secondfile thirdfile video.mpeg

ls b*

barry.txt blah.txt bob
```

Under the hood

This is quite interesting. At first glance, you might assume that the ls command is receiving argument b* and is translating that to bring up the requested matches. However, the translation is being done by bash, which is the program that is responsible for giving us the command line interface. When we put in this command, it will spot that we have included wildcards. Then, before it can run the command, it will replace that pattern with every single directory or file that matches it. We give the command

```
ls b*
```

and it is translated into

```
ls barry.txt blah.txt bob
```

before the program is executed. The program itself will not see the wildcards and will not know that they have been used. This means that they can be used whenever you want on the command line.

Some more examples

Let's look at a few more examples of how wildcards behave. For these examples, assume that we are working from a directory called linuxtutorialwork that contains all of the files listed in the above example. We'll keep using the ls command, but please note that wildcards can be used with any command.

In this example, I am going to ask for all the files that end with the .txt extension. I have used an absolute path, but wildcards work on both absolute and relative paths:

```
ls /home/Simon/linuxtutorialwork/*.txt
```

```
barry.txt blah.txt
```

Now, let's add the ? operator into the mix. For the next example, I want to look for all files whose second letter is an "i". Note that we are building this patern with the use of more than one wildcard:

```
ls ?i*
```

```
firstfile video.mpeg
```

Or we could look for all the files that have a three-letter extension:

```
ls *.???
```

```
barry.txt blah.txt example.png frog.png
```

Lastly, let's look at the range operator, denoted by a pair of square brackets []. Unlike the first two wildcards we looked at, both of which specified any character, this range operator will limit you to a specific subset. For example, we can find all files that begin with an "s" or a "v":

```
ls [sv]*
```

```
secondfile video.mpeg
```

When we use ranges, we can also use a hyphen to include a set. In this example, we want to find all of the files that have a digit in their name:

```
ls *[0-9]*
```

```
foo1 foo2 foo3
```

And we can use the caret (^) to reverse a range. This means that it will look for files with characters that are NOT one of those given:

```
ls [^a-k]*
```

```
secondfile thirdfile video.mpeg
```

Some real-world examples

While these examples show you how wildcards actually work, you might be asking yourself what they are. Wildcards are used everywhere, and as you go through your learning, you will find many ways to use them for your own purposes. The following are a few examples of how they can be used. This is just a small part of what they can do – remember that a wildcard can be used whenever a path is specified on the command line. With a bit of clever thinking, you can use them for all sorts of things.

In this example, we are trying to see all of the file types of all of the files in a specific directory:

```
file /home/Simon/*
```

```
bin: directory

Documents: directory

frog.png: PNG image data

public_html: directory
```

Move all files of type either jpg or png (image files) into another directory:

```
mv public_html/*.??g public_html/images/
```

In the next example, we want to find out what the size and the modification time are of the bash_history file – this resides in the home directory of every user and holds the history of every command that a user has ever input on the command line. You can also use this to locate hidden files:

```
ls -lh /home/*/.bash_history
```

```
-rw------- 1 harry users 2.7K Jan 4 07:32 /home/harry/.bash_history
```

```
-rw------- 1 Simon users 3.1K Jun 12 21:16 /home/Simon/.bash_history
```

Permissions

There are three things that are dictated by a Linux permission – how you can read, write to, and execute a file. Each one is denoted with a single letter:

- r read – You may view the contents of the file.
- w write – You may change the contents of the file.
- x execute – You may execute or run the file if it is a program or script.

For each file, we can define three different sets of people for whom we can specify permissions:

- owner – A single person who owns the file. (Typically the person who created the file, but ownership may be granted to someone else by certain users.)
- group – Every file belongs to a single group.
- others – Everyone else who is not in the group or the owner.

So, that's three permissions and three different sets of people. Really, that's all there is to know about permissions, so let's have a look at how we can view them and change them.

Viewing permissions

Viewing permissions for a particular file involves using the long listing option with the ls command:

```
ls -l [path]
```

For example:

```
ls -l /home/Simon/linuxtutorialwork/frog.png
```

```
-rwxr----x1harryusers2.7KJan                                   4
07:32/home/Simon/linuxtutorialwork/frog.png
```

We look at the first 10 characters of the output to identify the permissions:

- Character 1 identifies the type of file. A dash (-) indicates a normal file, while a "d" indicates a directory
- Characters 2, 3 and 4 identify the specific permissions for the owner. Letters indicate the permission and a dash (-) indicates that there are no permissions. The example shows that the owner has all

three of the permissions.

- Characters 5 through 7 show permissions for the group. Here, group members are only permitted to read the file.
- The last three characters indicate permission for others. In the example, we can see that they have permission to execute files but cannot read or write to them.

Changing permissions

To change a permission on a directory or file, we use the chmod command, which stands for change file mode bits. The bits are the indicators for the permissions, and the syntax is:

```
chmod [permissions] [path]
```

Chmod has permission arguments that are made up of three components, i.e.:

- For whom are the permissions being changed? – UGOA = user, group, others, all
- Are we giving permission or removing it? – indicated with a + or a –
- Which permission is being set? – read, write, execute

Look at the following example to see if it is any clearer to you. We are granting permission for execute to the group and taking permission for write away from the user, or owner.

```
ls -l frog.png

-rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod g+x frog.png

ls -l frog.png

-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod u-w frog.png

ls -l frog.png

-r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
```

Instead of giving individual permissions, you can give multiple permissions at the same time:

```
ls -l frog.png
-rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
chmod g+wx frog.png
ls -l frog.png
-rwxrwx--x 1 harry users 2.7K Jan 4 07:32 frog.png
chmod go-x frog.png
ls -l frog.png
-rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png
```

While it might seem a little strange that the owner of a file can remove his own read, write and execute permissions for a file, there are good reasons for doing so. You might have a file that contains data that you don't want to be changed accidentally. While you can remove the permissions, you cannot remove your ability to set the permissions, and that means that you retain full control of every file that you own.

Setting permissions shorthand

The permission-setting methods we looked at above are easy enough, but can get a little tedious if you have to apply a set of permissions on a regular basis to specific files. You can use shorthand to do this. To understand how the shorthand system works, you must first know a little about how the numbers system works. The usual numbers system is decimal, a base 10 system. This means it has 10 symbols – 0 through 9. There is also the octal system, which has eight symbols – 0 through 7. Now, with three permissions and with each one being on or off, there are 8 combinations. These numbers can also be shown in binary, which has just two symbols – 0 and 1. The table below shows how we map from octal to binary:

OCTAL	**BINARY**
0	**000**
1	**001**
2	**010**
3	**011**

4	**100**
5	**101**
6	**110**
7	**111**

Now, all of the octal values can also be represented with three binary bits, and every combination of ones and zeros has been included. Now we have three permissions and three bits. The 1 represents on and the 0 represents off, and with that, we can use a single octal number to represent a specific set of permissions for a specific set of people. Using three numbers we can set permissions for the owner, the group and others. Let's have a look at an example – don't forget to refer to the above table to see how it all works:

```
ls -l frog.png

-rw-r----x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod 751 frog.png

ls -l frog.png

-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png

chmod 240 frog.png

ls -l frog.png

--w-r----- 1 harry users 2.7K Jan 4 07:32 frog.png
```

One convenient method is to remember a common sequence for different file types, e.g., 750 and 755 are used for scripts.

Permissions for directories

We can use the same permissions for directories, but they will behave a little differently:

- r – You have the ability to read the contents of the directory (i.e., do an ls).
- w – You have the ability to write into the directory (i.e., create files and directories).
- x – You have the ability to enter that directory (i.e., cd).

Let’s look at some examples of how these work:

```
ls testdir
file1 file2 file3
chmod 400 testdir
ls -ld testdir
-r-------- 1 Simon users 2.7K Jan 4 07:32 testdir
cd testdir
cd: testdir: Permission denied
ls testdir
file1 file2 file3
chmod 100 testdir
ls -ld testdir
---x------ 1 Simon users 2.7K Jan 4 07:32 testdir
cd testdir
ls testdir
ls: cannot open directory testdir/: Permission denied
```

When we ran ls on lines 5 and 14, we included the option –d, which means directory. Normally, if ls is given an argument that is a directory, it will show the contents for the given directory as an output. However, in this example, what we have asked for is to see the permissions for the given directory, and that is what we have been shown.

Permissions can seem somewhat confusing to start with, but you just need to remember that these particular ones are for the directory and not the files that are contained in it. For example, if you have a directory that you do not have a read permission for, that contains a file that you do have the read permission for, then provided you know that the file is there and you know its name, you are still able to read the file:

```
ls -ld testdir
```

```
--x------- 1 Simon users 2.7K Jan 4 07:32 testdir
cd testdir
ls testdir
ls: cannot open directory .: Permission denied
cat samplefile.txt
Kyle 20
Stan 11
Kenny 37
```

The root user

There are just two people who can change the file or directory permissions: the owner for the specific directory or file and the root user. This super user has access everything and can do anything. Usually, only a system administrator has access to the root account, and he uses it to keep the system maintained. Normal users generally only have access to their own files and directories, those located in their home directory, and perhaps a couple of others that are being shared or that they are collaborating on with other users. This is how the stability and the security of the system are maintained.

Basic security

Your space is your home directory and it is up to you to make sure that it remains yours. Most users tend to give themselves full permissions for read, write and execute in their home directory and will not give permissions to the group or to others. However, everyone has a different setup.

To maintain security, you should refrain from giving write access to your home directory to anyone in the group or others. However, being able to execute without read permission can be handy at times. This allows others to get to your home directory, but they will not be able to see what is contained in it. One good example of this is when it is used for personal web pages. Systems typically run web servers and each user is given their own bit of space. If you put a directory into your home directory and call it public_html, the webserver will generally be able to read the contents and display them. However, because the webserver is a different user, it will not be able to access and read the files. This is where you might want to give execute permissions on the home directory so that the webserver can do its job.

Chapter 10: The vi Text Editor

In the last chapter we created some blank files. In this chapter we are going to look at a tool that helps us to put some content into those files and allows us to edit the content as well. vi is a nice text editor, something different from most of the text editors you have used in the past. It will take you a while to learn it but, once you have, you will see just how powerful it is.

Essentially, vi is a command line editor. As you know by now, the command line is somewhat different from the Linux GUI. It is just one window where you input text and output text, that is all. vi works within those strict limitations, making it an extremely powerful editor. It is a plain text editor, similar to TextEdit on the Mac or Notepad on Windows. It is NOT a word processor, but it is more powerful than either of the aforementioned built-in editors.

You do not need your mouse with vi; everything in done on the keyboard.

There are two separate modes – Insert (input) and Edit. In the Insert mode, you type in content to go into a file. In Edit, you move around inside the file, and edit the content, as by deleting, searching, copying, replacing, saving, etc. One of the most common mistakes that people make is to start inputting commands without being in Edit mode or to start typing in their input without being in Insert mode. Do check before you start. That said, if you do make one of these mistakes, it is easy to recover from it.

To run the vi text editor, you use a single command line argument, naming the file you want to edit:

```
vi <file>
```

If you don’t name a file, you can open it within, vi but it is actually easier to shut the editor down and start again. Do remember as well that you can use a relative or an absolute path to specify the file name.

First of all, with your terminal open, move to the directory that you made called linuxtutorialwork. We want to make some files, and we can store them here to keep them away from everything else you do.

Now we need to edit your first file, so type this into the command line:

```
vi firstfile
```

When this command is run, it will open up the file, but if the file does not exist, it will create it before opening it up. When you get into vi, it may look a little like this (depending on what system you are running):

```
~
~
~
~
~
"firstfile" [New File]
```

Always start in Edit mode. In this case, we now want to go into Insert mode, so press i. You will see from the bottom left corner of your screen when you go into Insert mode.

```
~
~
~
~
~
INSERT --
```

Type in some text, anything you like, and press Esc. This takes you back into Edit mode.

Saving & exiting

There are a couple of ways to do this; just choose whichever method suits you, but do make sure that you are in Edit mode first by glancing at the bottom left corner of the screen. If it is blank, you are OK; if it says INSERT you need to get back into Edit mode first. To do this, you could just press Esc – if you are already in Edit mode, it won't do anything. Then, to save and exit, type either of the following:

- :wq + Enter
- ZZ

Most of vi commands will be executed as soon a key sequence is entered. If a command begins with a colon (:) you will need to hit the Enter key for the command to be completed.

Save and exit out of the file that you are in now.

Other ways to view files

You can edit files in vi ,and you can also view files. However, there are a couple of other commands that are a bit better for doing that. The first is cat, short for concatenate. It is usually used to join two files together, but it can be used to view a file as well:

```
cat <file>
```

Running the cat command with just a single command line argument will let you see the file content on the screen, followed by the prompt. If you run the cat command with a command line argument, the cursor will move onto the next line but nothing else will happen. If you type in some text and then press Enter, cat would mirror what you input onto the screen. You can get out of this by pressing Ctrl+C, which is the Cancel signal in Linux. (You can use this to get out of most trouble that you get into in Linux.)

```
cat firstfile

here you will see

whatever content you

entered in your file
```

This is a neat command to use if you only have a small file to look at, but when the files are larger, the content will fly across your screen and you will only be able to view the last page. To view larger files, we use a different command called *less*:

```
less <file>
```

The less command lets you move through the file contents with the arrow keys. You can forward to the next page with the spacebar or you can press "b" to go back a page. When you are finished, just press "q" to quit.

Try using both of those commands on the file you created and see how you get on with them.

Navigating a file in vi

Now, go back to that file and input some more content. While you are in Insert mode, you can use the arrow keys to get around. Type in two more paragraphs and then press Esc so you go back into Edit mode.

The following list shows some of the commands that can be used to traverse through a file; practice with them and get to grips with how they work.

- Arrow keys – move the cursor around
- j, k, h, l – move the cursor down, up, left and right (similar to the arrow keys)
- ^ (caret) – move cursor to beginning of current line
- $ – move cursor to end of the current line
- nG – move to the nth line (e.g., 5G moves to 5th line)
- G – move to the last line
- w – move to the beginning of the next word
- nw – move forward n words (e.g., 2w moves two words forward)
- b – move to the beginning of the previous word
- nb – move back n words
- { – move backward one paragraph
- } – move forward one paragraph

Deleting content

You now know a few ways to move around in vi. Deleting works in a similar way; there are a number of different delete commands that you can use to define what you want to delete. The list below shows some of the ways to delete. Practice with them, but only after you have read the section on how to undo a deletion:

- x – delete a single character
- nx – delete n characters (e.g., 5x deletes five characters)
- dd – delete the current line
- dn – d followed by a movement command. Delete to where the movement command would have taken you (e.g., d5w means delete 5 words).

Undoing

It is a fairly easy process to undo a delete in vi; just use the letter "u".

- u – Undo the last action (you can keep pressing u to keep undoing)
- U (Note: capital) – Undo all changes to the current line

Taking it further

Now you know how to insert content, delete it, undo the delete, save it and then exit. This is how to do basic editing work in vi. While we can’t go into all the details of vi here, you should explore these other features of the text editor on your own:

- copy and paste
- search and replace
- buffers
- markers
- ranges
- settings

Have fun; learning vi will be painful at times, but once you have gotten to grips with it, you won’t want to use any other text editor.

Conclusion

Not many years ago, Linux was just a word that computer geeks used to enhance their resumes. But nowadays, you can find Linux in almost all of the data centers and servers in the world. The Linux operating system is pretty straightforward and it is not as complex as other operating systems. Linux is available free of charge, which makes it available for everyone who wishes to learn it. Once you install Linux on your computer, you become part of a huge Linux community where you can learn everything about Linux from people who are willing to help newbies like you.

In this tutorial we have covered the Linux kernel and shell scripting, along with topics like inter-process communication and a lot of commands with which you can perform most of the simple tasks in Linux. I have given you a basic overview of using Linux, with plenty of examples to try out.

Linux really isn't all that difficult to learn. Once you get into it, you will find that you pick it up quite easily, but – and I cannot stress this enough – you must practice. Do not think that, once you have learned something, it will stay with you forever – it won't if you don't use it. And don't forget that things are changing all the time and advances are constantly being made. If you do not keep up with those changes, you will soon fall behind and have to start all over again.

I want to thank you for downloading my book, and I hope you found this tutorial helpful.

Did You Like This Book?

Before you leave, I wanted to say thank-you again for buying my book.

I know you could have picked from a number of different books on this topic, but you

chose this one, so I can't thank you enough for doing that and reading until the end.

I'd like to ask you a small favor.

If you enjoyed this book or feel that it has helped you in anyway, then could you please

take a minute and post an honest review about it on Amazon?

Your review will help get my book out there to more people and they'll be grateful, as

will I.